Advanced search

Linux Journal Issue #125/September 2004

Listening to FM Radio in Software, Step by Step

Eric Blossom

Issue #125, September 2004

Get started in software-defined radio with a project that can tune in two FM stations at once.

My article “GNU Radio: Tools for Exploring the Radio Frequency Spectrum” [*LJ*, June 2004] provides an overview of how the GNU Radio system works and discusses a couple hardware options for getting the RF signals digitized and into the computer. This article takes a look at how to use GNU Radio to listen to FM radio.

The hardware setup used in this article is shown in Figure 1. It's the brute-force, no-frills approach and is good for explaining how everything works. Later in the article, we discuss the Universal Software Radio Peripheral (USRP) and what it can do for you.

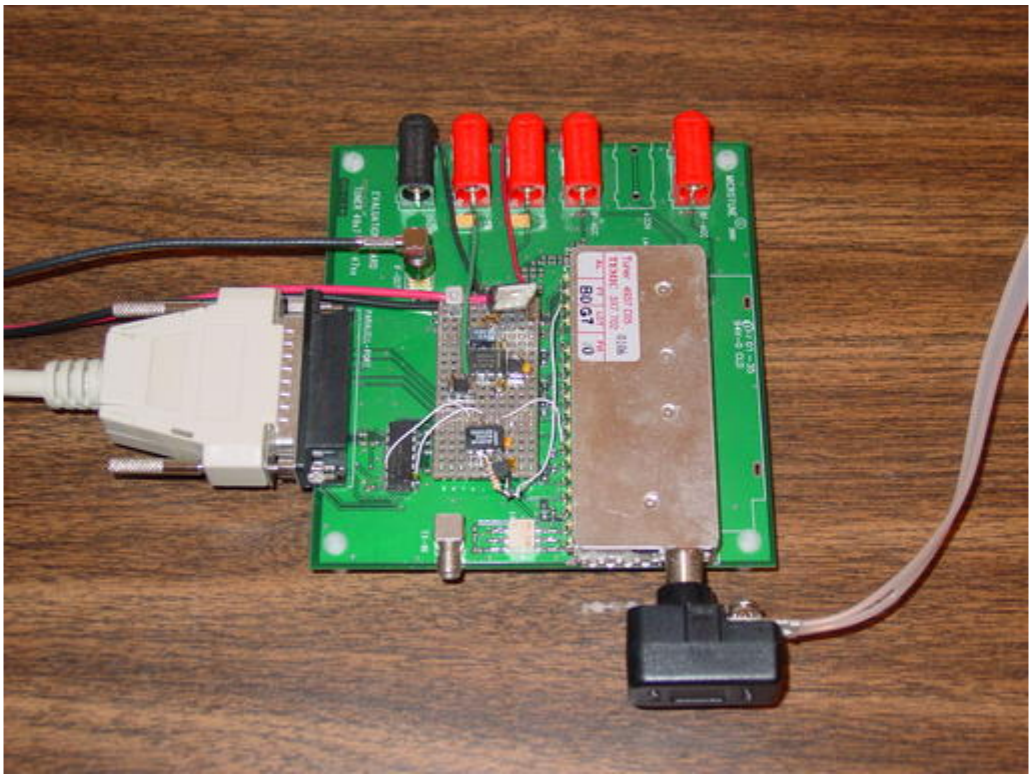


Figure 1. Cable Modem Tuner RF Front End

Our setup consists of a conventional FM dipole antenna, a cable modem tuner module mounted on an evaluation board and a 20M sample/second PCI analog-to-digital converter (ADC) card. The antenna plugs in to the input of the tuner module. The tuner module IF output is connected with a piece of coaxial cable to the ADC input on the back of the computer. The tuner module eval board is connected to the PC's parallel port so that we have a way of controlling the module.

The specific hardware we're using is a Microtune 4937 DI5 3X7702 cable modem tuner module and a Measurement Computing PCI-DAS 4020/12 ADC board. This particular tuner module is hard to get, but others, such as those from Sharp Microelectronics, ought to work fine (see the on-line Resources section).

The cable modem tuner functions as our RF front end and is responsible for translating the radio frequency signals that we're interested in down to a range that our ADC can deal with. In this case, the module translates a selectable 6MHz chunk of the spectrum in the range of 50MHz–800MHz down to a 6MHz chunk centered at 5.75MHz. For more background on these concepts, see the June article mentioned previously.

Getting Started

First off, let's take a look at what happens when we tune our front end to the middle of the FM band, say 100.1MHz. Figure 2 shows the received samples vs.

time. This view, the time domain, is what you'd see on an oscilloscope. It's not particularly enlightening, but it does show that our samples are in the range of –170 to –70, which is fine. In an ideal world, they would be symmetric about zero. For our purposes, the offset won't matter.

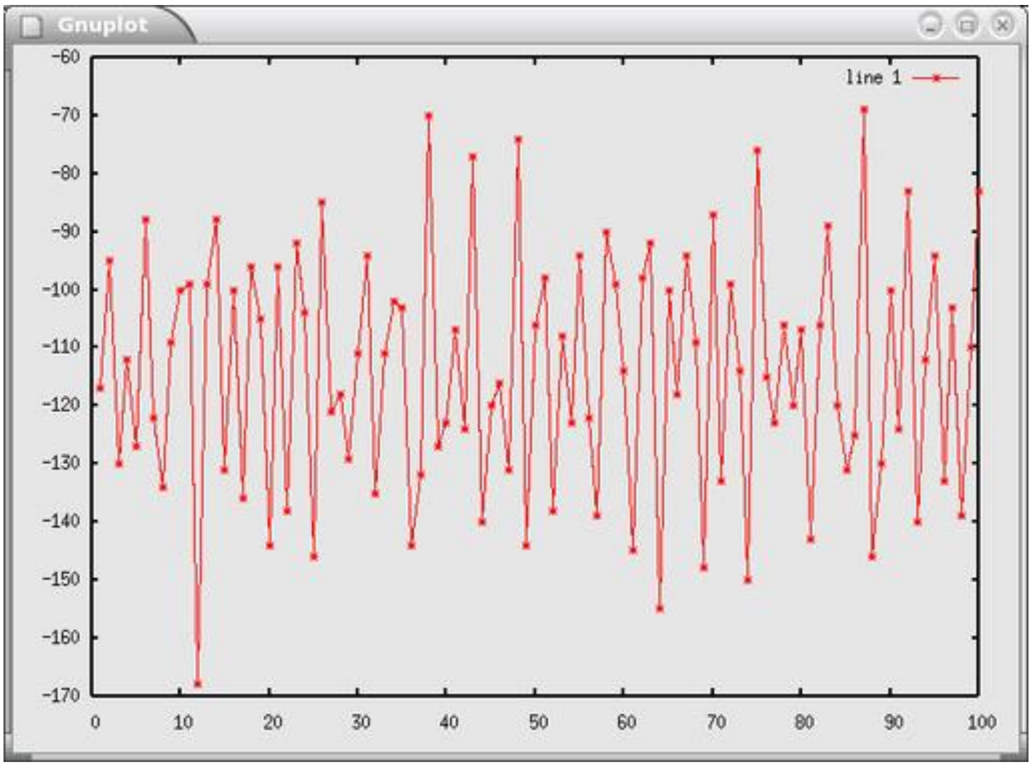


Figure 2. ADC Samples in the Time Domain

The frequency domain provides additional information. In this case, we grab 1,024 samples at a time and compute the discrete Fourier transform using the fast Fourier transform (FFT) algorithm. This gives us a representation of the frequencies that are contained in the input signal. Figure 3 shows the resulting spectrum. The x-axis is frequency, and the y-axis is power in decibels (10 * $\log_{10}$ power). The low end is at zero Hz, and the top end is at 10MHz, half our sampling rate.

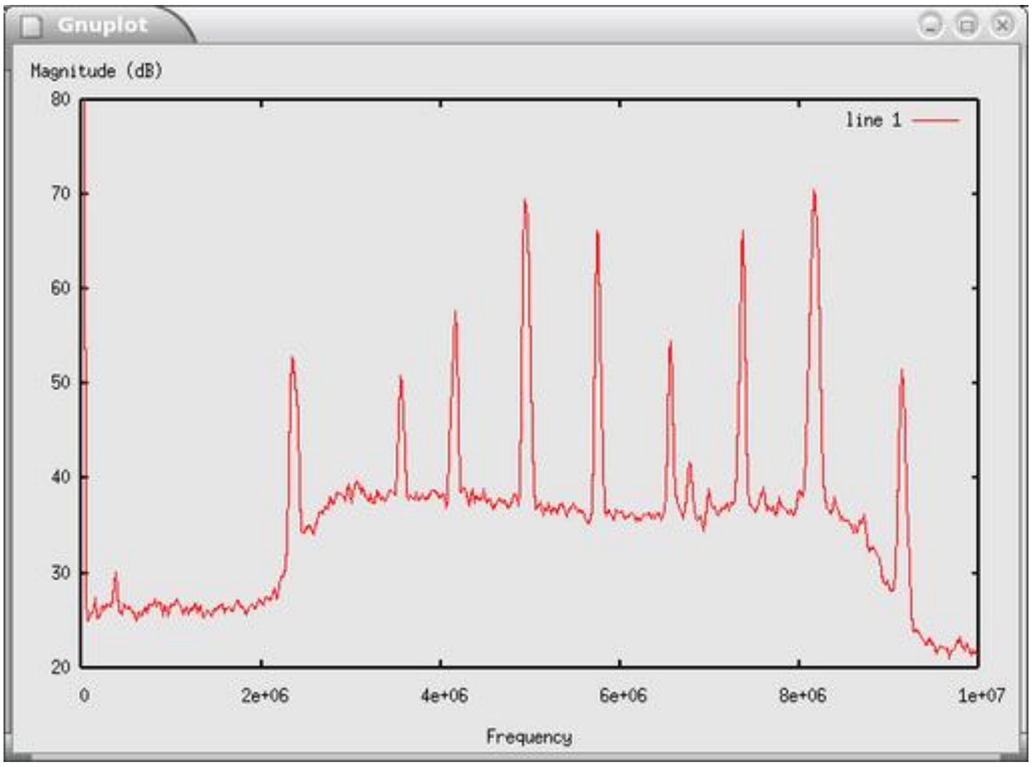


Figure 3. Fast Fourier Transform of FM Band with Nine Stations

Each of the spikes in Figure 3 is a radio station. Our software sees them all at once! To listen to a station, we need a way to separate it from all of the others, translate it to baseband (DC, 0Hz) and reverse the effect of the frequency modulation. We work through this step by step, but first let's talk about FM.

What Is Frequency Modulation?

To understand how an FM receiver works, it's helpful to know a bit about how FM signals are generated. With FM, the instantaneous frequency of the transmitted waveform is varied as a function of the input signal. Figure 4 shows m(t), the input signal (the message, music and so forth), and s(t), the resulting modulated output. To be rigorous, the instantaneous frequency at any time is given by the following formula:

$f(t) = k*m(t) + f_c$

m(t) is the input signal, k is a constant that controls the frequency sensitivity and f_c is the frequency of the carrier (for example, 100.1MHz). Remember that frequency has units of radians per second. As a result, frequency can be thought of as the rate at which something is rotating. If we integrate frequency, we get phase, or angle. Conversely, differentiating phase with respect to time gives frequency. These are the key insights we use to build the receiver.

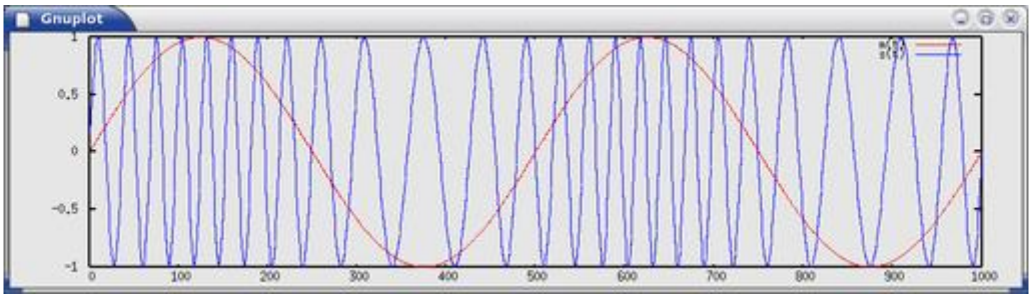


Figure 4. A Simple Frequency Modulated Signal

The Block Diagram

Figure 5 shows our strategy for listening to an FM station. If we remove the carrier, we're left with a baseband signal that has an instantaneous frequency proportional to the original message m(t). Thus, our challenge is to find a way to remove the carrier and compute the instantaneous frequency.

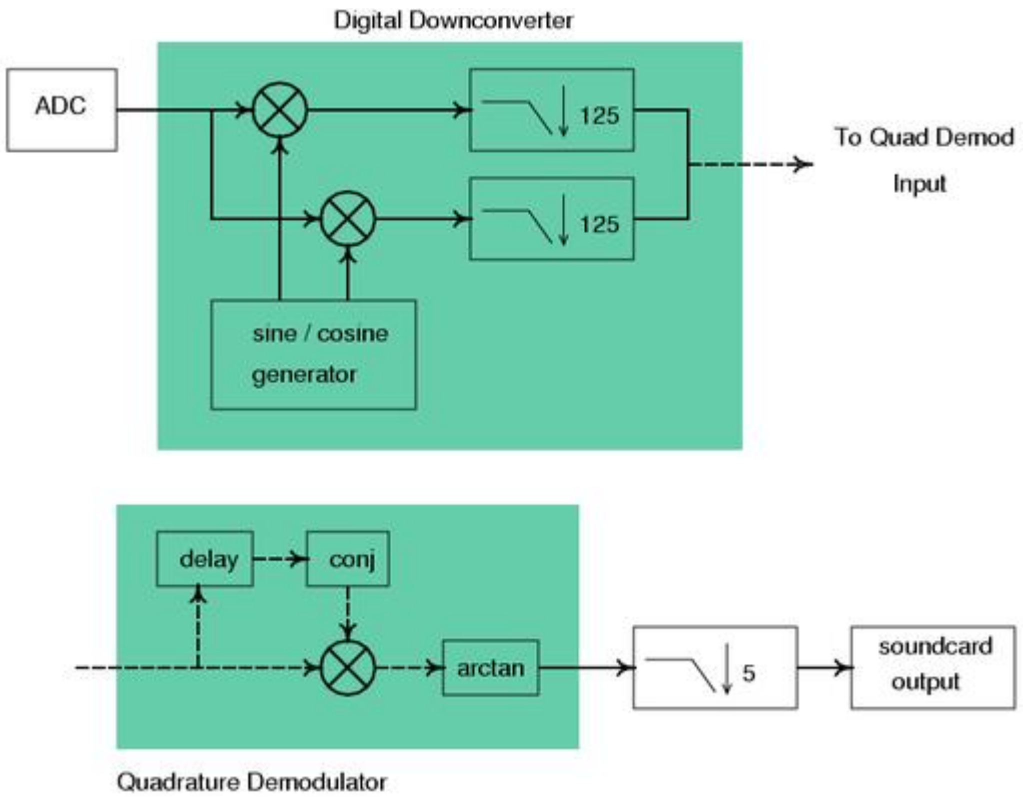


Figure 5. Block Diagram of FM Receiver

The first part is easy. We get rid of the carrier by using our software digital downconverter (DDC) block, freq_xlating_fir_filter_scf. This block is composed conceptually of a numerically controlled oscillator that generates sine and cosine waveforms at the frequency that we want to translate to zero, a mixer (that's a multiplier to us software folks) and a decimating finite impulse response filter. The scf suffix indicates that this block takes a stream of shorts on its input, produces a stream of complexes on its output and uses floating-point taps to specify the filter.

The digital downconverter does its job by taking advantage of a trigonometric identity that says when you multiply two sinusoids of frequency f_1 and f_2 together, the result is composed of two new sinusoids, one at f_1+f_2 and the other at f_1-f_2. In our case, we multiply the incoming signal by the frequency of the carrier. The output consists of two components, one at 2x the carrier and one at zero. We get rid of the 2x component with a low-pass filter, leaving us the baseband signal.

MIPS Are Us!

A straightforward implementation of the digital downconverter block in software is extremely expensive computationally. We'd be performing the sine and cosine generation and multiplication at the full input rate. On a Pentium 4, computing sine and cosine takes on the order of 150 cycles. Given a 20M sample/sec input stream, we'd be burning up 20e6 * 150 = 3e9 cycles/sec merely computing sine and cosine! Definitely a non-starter.

The good news is there's a better way to implement the DDC in software. This technique, described by Vanu Bose, et al., in “Virtual Radios” (see Resources), allows us to run all of the computation at the decimated rate by rearranging the order of the operations and using frequency-specific complex filter coefficients instead of real coefficients. The end result is a big win! We can do it in real time!

Quadrature Demodulation

The next job is to compute the instantaneous frequency of the baseband signal. We use the quadrature_demod_cf block for this. We approximate differentiating the phase by determining the angle between adjacent samples. Recall that the downconverter block produces complex numbers on its output. Using a bit more trigonometry, we can determine the angle between two subsequent samples by multiplying one by the complex conjugate of the other and then taking the arc tangent of the product. Listings 1 and 2 show the implementation of the quadrature_demod_cf block. Once you know what you want, it doesn't take much code. The bulk of the signal processing is the three-line loop in sync_work.

Listing 1. Quadrature Demodulator Header

```
/*
 * Copyright 2004 Free Software Foundation, Inc.
 *
 * This file is part of GNU Radio
 *
 * GNU Radio is free software; you can redistribute
 * it and/or modify it under the terms of the GNU
 * General Public License as published by the Free
 * Software Foundation; either version 2, or (at
 * your option) any later version.
 */

#ifndef INCLUDED_GR_QUADRATURE_DEMOD_CF_H
#define INCLUDED_GR_QUADRATURE_DEMOD_CF_H

#include <gr_sync_block.h>

class gr_quadrature_demod_cf;
typedef boost::shared_ptr<gr_quadrature_demod_cf>
  gr_quadrature_demod_cf_sptr;

gr_quadrature_demod_cf_sptr
gr_make_quadrature_demod_cf (float gain);

/*
 * quadrature demodulator: complex in, float out
 */
class gr_quadrature_demod_cf : public gr_sync_block
{
  friend gr_quadrature_demod_cf_sptr
    gr_make_quadrature_demod_cf (float gain);
  gr_quadrature_demod_cf (float gain);

  float         d_gain;

 public:

  int sync_work (
        int noutput_items,
        gr_vector_const_void_star &input_items,
        gr_vector_void_star &output_items);
};

#endif /* INCLUDED_GR_QUADRATURE_DEMOD_CF_H */
```

Listing 2. Quadrature Demodulator Implementation

```
/*
 * Copyright 2004 Free Software Foundation, Inc.
 *
 * This file is part of GNU Radio
 *
 * GNU Radio is free software; you can redistribute
 * it and/or modify it under the terms of the GNU
 * General Public License as published by the Free
 * Software Foundation; either version 2, or (at
 * your option) any later version.
 */
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <gr_quadrature_demod_cf.h>
#include <gr_io_signature.h>

gr_quadrature_demod_cf::gr_quadrature_demod_cf (
                                         float gain)
  : gr_sync_block (
       "quadrature_demod_cf",
      gr_make_io_signature(1,1,sizeof (gr_complex)),
      gr_make_io_signature(1,1,sizeof (float))),
    d_gain (gain)
{
  set_history (2);    // provide 1 sample look ahead
}

gr_quadrature_demod_cf_sptr
gr_make_quadrature_demod_cf (float gain)
{
  return gr_quadrature_demod_cf_sptr (
      new gr_quadrature_demod_cf (gain));
}
```

```
int
gr_quadrature_demod_cf::sync_work (
    int noutput_items,
    gr_vector_const_void_star &input_items,
    gr_vector_void_star &output_items)
{
  gr_complex *in = (gr_complex *) input_items[0];
  float *out = (float *) output_items[0];
  in++;         // ensure that in[-1] is valid

  for (int i = 0; i < noutput_items; i++){
    gr_complex product = in[i] * conj (in[i-1]);
    out[i] = d_gain * arg (product);
  }

  return noutput_items;
}
```

Tying this all together, Figure 6 shows the output of the digital downconverter, and Figure 7 shows the output of the quadrature demodulator. In Figure 7, you can see all the components of the FM waveform. From 0 to about 16kHz is the left plus right (L+R) audio. The peak at 19kHz is the stereo pilot tone. The left minus right (L-R) stereo information is centered at 2x the pilot (38kHz) and is AM-modulated on top of the FM. Additional subcarriers are sometimes found in the region of 57kHz–96kHz.

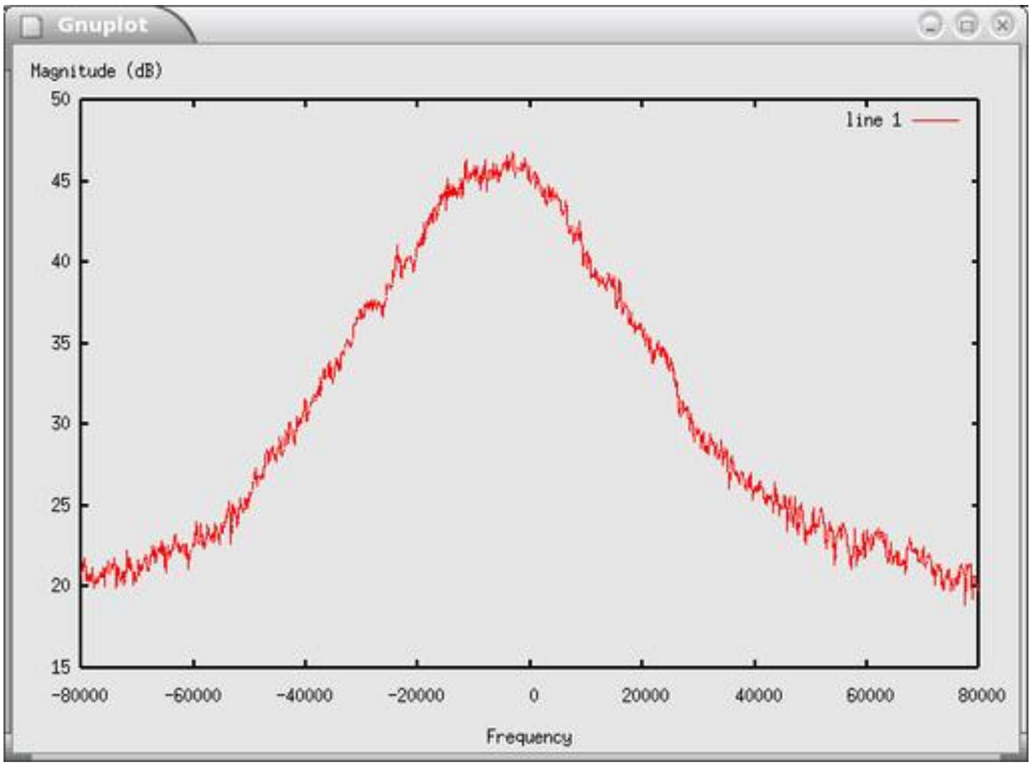


Figure 6. FFT at Output of Digital Downconverter

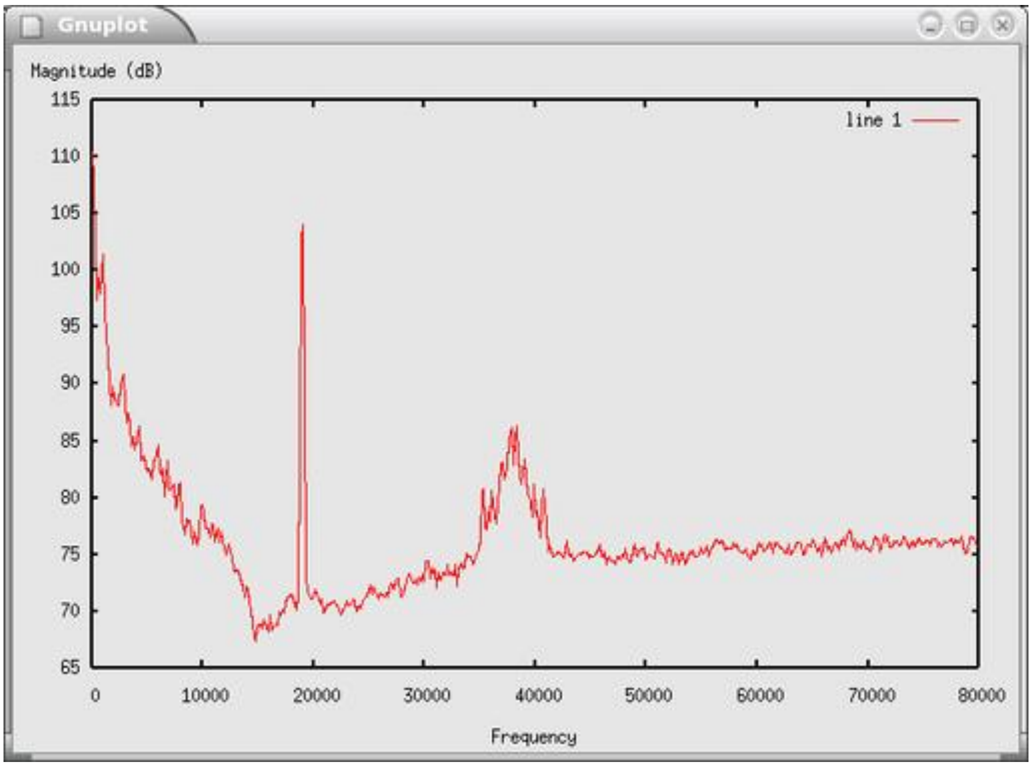


Figure 7. FFT of Demodulated FM Signal

To keep life simple, we low pass the output of the quadrature demodulator with a cutoff frequency of 16kHz. This gives us a monaural output that we connect to the sound card outputs.

A Multichannel Receiver

Listing 3, available from the *Linux Journal* FTP site (see Resources), is the Python code that implements the overall receiver. In fact, it can listen to two FM stations at the same time, one out the left speaker and one out the right! I'm not arguing that this is a particularly practical application, but it does illustrate some of the power of software radio. This idea of extracting multiple stations concurrently could be used as the basis of a multichannel TiVo-like device for radio.

The code is split into three functions. main handles the argument parsing, manages the RF front end and controls the main signal processing loop. If we're receiving a single station, we tell the RF front end to put the station right at the center of the tuner's output frequency, the IF. If we're receiving two stations, we ensure that they're within 5.5MHz of each other. This restriction is due to the SAW filter built in to the cable modem tuner. It's a bandpass filter centered at 5.75MHz that's about 6MHz wide, the width of a North American TV channel. In this case, we split the difference and tune the front end exactly halfway between the two stations. build_graph instantiates the common signal processing blocks and connects them together. In both the single and dual-station modes, we use a single high-speed analog-to-digital converter for input and a single sound card for output. For each station that we want to receive concurrently, we instantiate a digital downconverter, quadrature demodulator and low-pass filter.

There's More Than One Way to Do It!

The maximum number of stations that can be received concurrently is a function of the speed of your computer. Even with our fancy implementation, most of the CPU cycles still are burned in the freq_xlating_fir_filter blocks. What we've described could be called the dumb ADC/brute-force method. One way to free up computational resources is to move the digital downconversion into hardware. Companies such as Texas Instruments, Intersil and Analog Devices sell dedicated ASICs that do this. The strategy used in the Universal Software Radio Peripheral (USRP) is to code the digital downconverter in the Verilog hardware description language and then download the resulting bitstream over the USB into the FPGA. This gives us a combined hardware/software system that maximizes flexibility while still allowing us to off-load some of the more computationally intensive parts into hardware. For more information on the USRP, see the GNU Radio Wiki.

Summary

We've walked through a fully functional but stripped-down multichannel FM receiver. We managed to turn a couple thousand dollars' worth of hardware into the equivalent of two $5 transistor radios, and we learned a bunch in the process. For those of you interested in pursuing FM listening, the GNU Radio code base includes a substantially higher fidelity FM receiver (hifi_fm.py), along with all kinds of other goodies.

Right now, a lot of interesting work is being done with GNU Radio. Some are focusing on mobile ad hoc networking, others on the legacy amateur radio waveforms, some on software GPS and another group is working on designing the next-generation ground-to-space amateur satellite communication system. Although the GNU Radio toolkit is mostly indifferent to I/O devices, most of these efforts are using or planning on using the USRP as the interface between the RF world and the PC.

Resources for this article: /article/7650.

Eric Blossom is the founder of the GNU Radio Project. Prior to his involvement with software radio, he spent many years in the secure phone business. When he's not hacking software radio, you're likely to find him practicing yoga or jujutsu. He can be reached at eb@comsec.com.

Archive Index Issue Table of Contents

Chat on the Air with LinPsk

Volker Schroer

Issue #125, September 2004

The original international electronic hobbyist community is ham radio. Here are the basics of the chat room without the room.

Long before the Internet was established, an international group of people was chatting on the air—the radio amateurs. In the beginning, they used Morse code, a special form of digital communication. Over the years, more digital modes have been introduced to ham radio. Today, you can use satellites for communication or the moon to reflect radio waves. In the past, you could communicate with the Mir Space Station; now, it's with the International Space Station. To learn more about the fascinating world of ham radio, take a look at the on-line Resources section for this article.

One of the most popular new communication modes in use today is PSK31. Peter Martinez (G3PLX) developed PSK31 based on an idea by Pawel Jalocha (SP9VRC). PSK31 was designed for enabling keyboard-to-keyboard chats. PSK31 is a phase shift keying modulation technique, requiring a bandwidth of about 31Hz. By comparison, a ham FM signal is almost 500 times as large, about 10kHz.

In the early days, hams built their equipment themselves and tried to improve it. Today, it is hard to compete against industrial transceivers, but under certain conditions, digital modes offer a new chance to compete. All you need is your transceiver, a computer with a sound card and a program that supports the desired mode. If the program is open source, you also have an opportunity to do your own experiments.

Radio operators have developed a kind of language that they use for communication. Some ham-related abbreviations used in this article are listed in Table 1.

Table 1. Abbreviations

ALC	Automatic level control
bcnu	Be seeing you
CQ	General call
DL1KSV	This is my callsign. A callsign is a worldwide, unambiguous identifier (like an Ethernet MAC address). The prefix DL stands for Germany.
Dummy load	Simulated antenna
de	From
es	And
k	Go
pse	Please
RX	Receive
TX	Transmit
QSO	Connection
73	Best regards

My first contact with PSK31 was in 1999. Even though I was a fan of Linux, I had to use a Microsoft Windows program for my first PSK31 QSOs. Later, I tried to develop a PSK31 program for Linux. Luckily, I found WinPsk 1.0, by Moe Wheatley (AE4JY), which had been released in source code form. I used this code as the basis for my initial release of LinPsk 0.2. The version 0.7 release of LinPsk included the expanded functionality of RTTY, and around this time, LinPsk also was made available for Mac OS X as DarwinPsk.

Getting LinPsk

The primary source for getting LinPsk is the LinPsk home page, where you can find the current prerequisites. At the time of this writing, they are GCC 3.3, Qt 3.3.x, fftw 3.0.1 and portaudio v18. See the on-line Resources section for LinPsk sources. A Debian package of the LinPsk is available. Currently, it requires either OSS sound drivers or the OSS emulation under ALSA.

For those of you who are new to Linux, an ISO image is available that contains a complete Linux system with many ham-related programs. This image contains LinPsk. You can download it and burn it onto a CD, then boot your PC from this CD without installing any programs.

If you decide to install LinPsk yourself, the installation is straightforward:

```
tar xzf linpsk-0.8.0.3.tar.gz
cd linpsk-0.8.0.3
./configure
```

Or, you can type:

```
./configure --prefix="installation directory"
```

for an installation directory other than the standard /usr/local/bin.

Finally, do:

```
make
su
make install
```

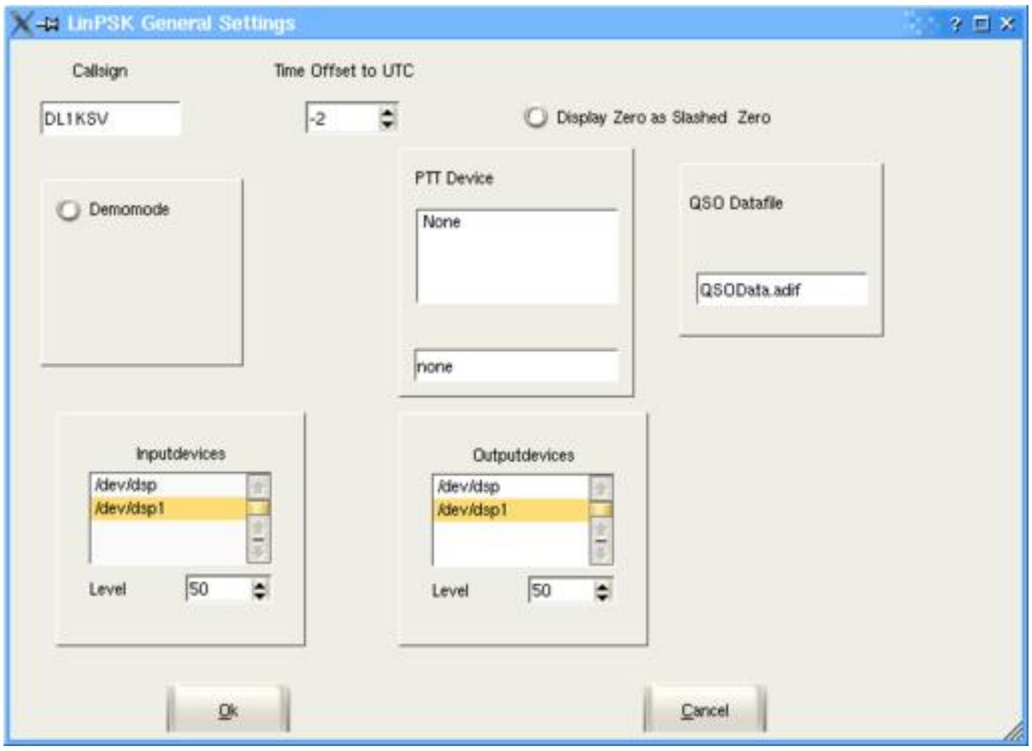
That's all. You should find a Linpsk executable in /usr/local/bin.

Now, it's time to connect the sound card to the transceiver. There are different ways of doing so, and a few proposals can be found at WM2U's PSK31 page.

No other program should use the sound card while Linpsk is starting. KDE's artsd, for example, should be stopped before running LinPsk, because it allocates the sound device.

Running LinPsk for the First Time

The first time you run the program, a warning appears to alert you that no LinPsk.config file exists. This is a reminder that LinPsk should be configured first. The configuration can be tweaked or changed at a later stage, but an initial configuration must be set up. The configure dialog can be found under the menu title Settings→General Settings. Once selected, a window as shown in Figure 1 should appear.


LinPSK General Settings

Callsign: DL1KSV

Time Offset to UTC: -2

Display Zero as Slashed Zero

Demomode

PTT Device: None

none

QSO Datafile: QSOData.adif

Inputdevices: /dev/dsp, /dev/dsp1 (selected)

Level: 50

Outputdevices: /dev/dsp, /dev/dsp1 (selected)

Level: 50

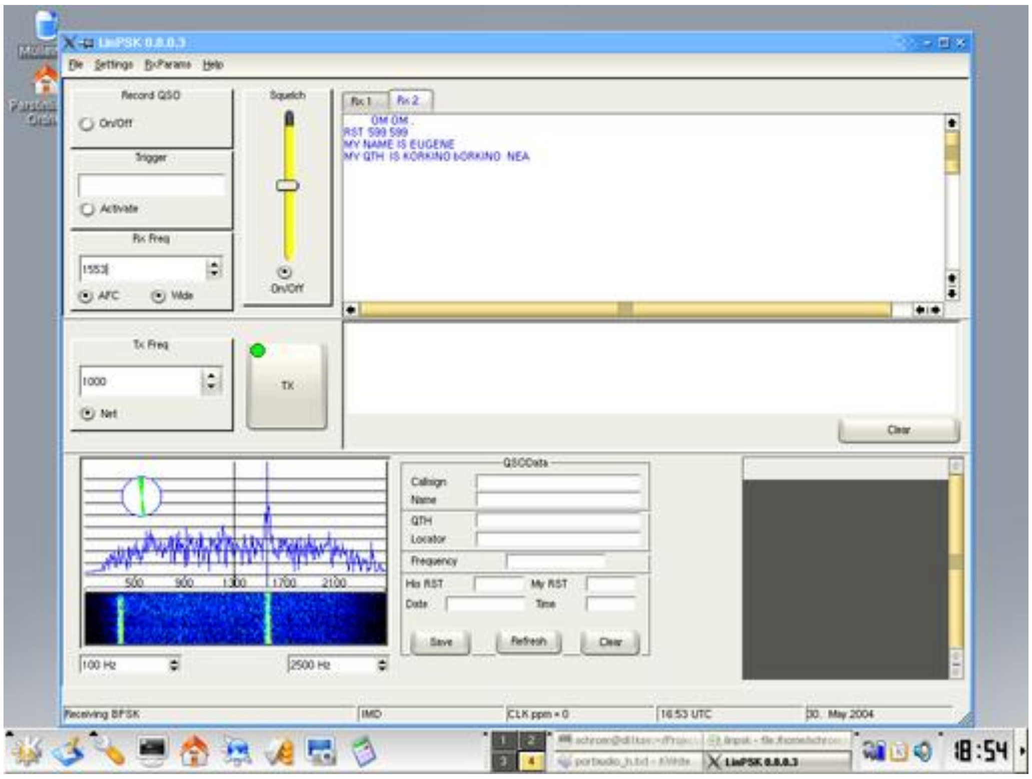
Figure 1. From the LinPsk General Settings menu you can configure your callsign, time zone and the devices to use.

The first field contains the callsign, which is used in the macros. The offset between the local and utc time should be set as well, and it is defined as:

```
   offset = utc - localtime .
```

To get the configuration dialog for the sound card, demo mode has to be deselected. The available input and output devices appear, and the one that is going to be used should be selected. The configuration dialog can be closed by clicking OK. The settings are not saved automatically for later use, so it is a good idea to save these settings with Save Settings in the Settings menu.

Tune your transceiver to 14.070.15MHz, upper sideband. If the 20-meter band is open, you should hear a variety of warbling tones from the radio speaker, each one representing a ham using PSK31. If everything is connected correctly, the screen should look like Figure 2 after clicking the RX button.



```
     OM OM .
RST 599 599
MY NAME IS EUGENE
MY QTH  IS KORKINO bORKINO  NEA
```

Rx Freq: 1553
Tx Freq: 1000
100 Hz
2500 Hz
Receiving BPSK | IMD | CLK ppm = 0 | 16:53 UTC | 30. May 2004

Figure 2. The main LinPsk window shows the available PSK31 signals in the lower left.

In the bottom left corner, the spectrum of the received signal is shown, and beneath that a translation of the spectrum values into colour values appears.

Due to the bandwidth of your transceiver, there may be a black gap on the left and right side of the spectrum. If a PSK signal is received, you should see one or more sharp peaks in the spectrum. If you click near the top of one of such a peak, and the options AFC and Wide under Rx Freq are selected, LinPsk tunes to the signal. In the circle inside the spectrum display, the phase of the received signal is represented.

This phase display can be used as indicator for quality of the received signal and as a tuning indicator at the same time. For a well-tuned BPSK signal, one sees only a vertical line. If the signal is slightly mistuned, the line is a bit buckled and rotated against the vertical.

As the AFC pulls the RX frequency to the center frequency of the PSK signal, the squelch display changes. The bar inside the squelch display increases and exceeds the threshold, and it changes colour from cyan to yellow. If this happens, letters appear in the receive window—you're now seeing the ongoing chat between two or more PSK31 users.

It is possible to listen to more than one QSO at the same time. To do this, one has to open another RX window from the file menu. As LinPsk supports other modes of operation, not only PSK, you are asked to select a mode for the new window. After that, you can click into the spectrum display at the desired frequency and watch.

The First QSO

After watching for a while, you might get curious and decide to answer a call. Be warned, though: you need a license to transmit on the ham bands.

Before transmitting, you should adjust the output volume carefully. PSK31 consumes little bandwidth, but if the output level is too high it can overdrive your transceiver, resulting in a lot of side loops that do not improve the coverage of the transmission. It's a good idea to use a dummy load when adjusting the output volume.

You should set the output volume in the General Settings menu to a low value. Pushing the TX button changes the labeling from TX to RX. The next time you push this button, the state changes to RX again.

Re-open the settings and watch the ALC of your transceiver. You should raise the output volume slowly until the ALC reacts, and then lower the volume some units again until the ALC settles down. Now the settings can be closed. Switch back to RX by pressing the TX button, and you are ready to answer calls.

You might tune to a running QSO and wait until it's over and one station calls CQ. Sending CQ means that the ham operating the station is interested in starting a new QSO with any other ham out there. Frequently, many QSOs are running at the same time. Which station will call CQ again first?

In such a case, open more RX windows from the File menu, tune each window to a QSO and activate the trigger in the upper-left corner. As the text to be triggered, enter CQ CQ. The trigger corresponds to the active RX window each time. Switch between the different RX windows and make another one active by clicking a tab in the right part of the window.

As soon as the trigger text is detected, LinPsk beeps and switches to the window in which the trigger text was detected. Whichever way you prefer, it now is time to answer the call.

Push TX and type the text you want to send into the TX window. Each character of the text transmitted is displayed in the RX window. When all text is transmitted, push RX again and wait for the reply. If the other station returns to your response, your first PSK31 QSO is going to start.

Tips

Make sure that the net option is selected in the Tx Freq window, which is the default. This option causes the transmit frequency to be identical to the receive frequency.

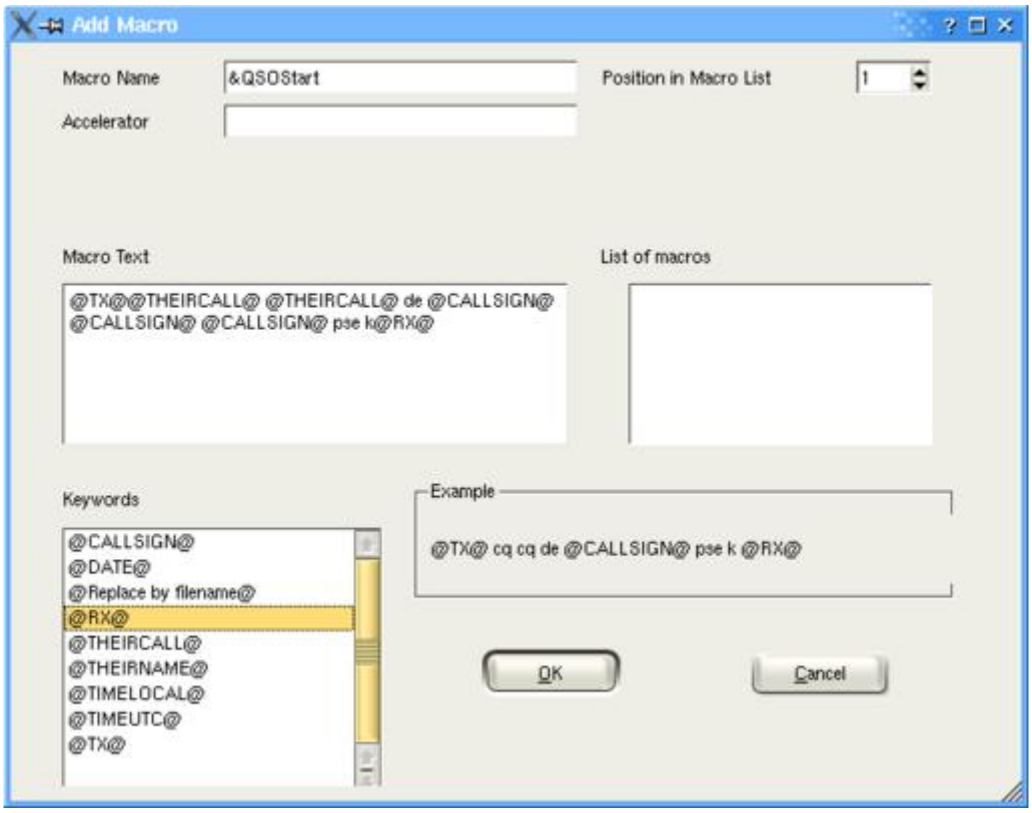
You can enter text into the TX window while listening. The prescribe buffer spans up to 1,000 characters. This text is transmitted as soon as you push TX. While the text is being transmitted you can add more text. You can use copy and paste for adding text. When all is transmitted, switch back to RX.

Further Steps

After following or even doing a few QSOs, you should notice that most of the text you type is the same. So instead of typing callsigns and other things over and over again, you can use macros, which are intended to make repeating text as easy as possible. The macro functionality is explained by answering a CQ call as an example.

Answering a call usually follows a few standard steps. Switch to TX, send the remote callsign twice, then “de”, then your callsign three times, then “pse k” and then switch to RX. You can implement all of these steps in a macro. To create a new macro, open Add Macro in the Settings menu. The first thing to enter is the name of the new macro. This name serves as a label for the button that appears in the lower right-hand black box.

We use QSOStart as the macro name and mark the letter Q with a preceeding &, which enables us to execute this macro by pressing Ctrl-Q. Of course, one is able to execute this macro by clicking its button as well. The complete macro definition is shown in Figure 3.



Macro Name: &QSOStart

Position in Macro List: 1

Macro Text:
```
@TX@@THEIRCALL@ @THEIRCALL@ de @CALLSIGN@
@CALLSIGN@ @CALLSIGN@ pse k@RX@
```

Keywords:
```
@CALLSIGN@
@DATE@
@Replace by filename@
@RX@
@THEIRCALL@
@THEIRNAME@
@TIMELOCAL@
@TIMEUTC@
@TX@
```

Example: @TX@ cq cq de @CALLSIGN@ pse k @RX@

Figure 3. Available keywords for macros include your callsign, the other station's callsign and the time.

In the lower-left side of the window, you can see a list of available keywords. Double-clicking a keyword copies it from the keyword box into the macro definition.

Using the @TX@ and @RX@ keywords allows you to switch between different modes. The @CALLSIGN@ keyword references the callsign entered in the General Settings menu. @THEIRCALL@ and @THEIRNAME@ reference the callsign and name field in the QSOData window. You can fill in these fields by entering the corresponding values directly or by copying and pasting them from the RX window. If one of the referenced fields is empty, an empty string is printed when using the macro.

Even inserting the contents of a complete file into a macro is possible. To do so, one has only to enter the name of file in the user's home directory quoted by @, or click the @Replace by filename@ keyword and replace the text. No replacement of keywords takes place inside the files. I used such a text file for the description of my station and to introduce myself. Because I can use different files, I am able to send this description in different languages. The other keywords do exactly what their names suggest.

You can implement the macros depending on your own needs. The number of macros possible is limited only by memory and storage. If the macros are set up properly, one can handle complete QSOs with only a few mouse clicks.

Final tip: you should always save settings if you are interested in keeping your macros.

Additional Features

Some settings are available to tailor LinPsk to your personal needs. On the Font Settings submenu, for example, you can select the font and its size. In the colour submenu, you can choose the colour used to display text in the RX window. Then, when using many RX windows you can distinguish among them by colour. The colour of the text is calculated automatically, but you might prefer some other colours. Displaying the text is done in the same colour used to paint the center frequencies in the spectrum display.

The colours settings menu sets the colour only for the current active RX window. By saving the settings, all colour settings are saved and used again on restart. If you open more RX windows than there are colours set, the additional colours again are chosen by calculation.

If you want to save the text of a QSO, simply click Record qso. Clicking the Record button again stops the recording. You activate this feature for each RX window individually.

The main data of a QSO can be saved to a file. The data is written as plain ASCII text in adif format. The default name of this file is QSOData.adif and can be reset in the General Settings menu. If no file by this name exists in the home directory, the file is created. Each record is appended to the end of this file.

Future Plans

When beginning work on LinPsk, my objective was to learn C++ and to develop a PSK31 program for Linux that was easy to use. KDevelop was a great help for me during this development. Meanwhile, I implemented RTTY as an additional mode. This mode works, but the decoder part should be improved. At that point, I tried to modify the code to get a framework for implementing different digital modes. I tried to implement MFSK16, but that mode is not functional as of yet. So this is another point for future developments.

I was asked to port LinPsk to Mac OS X. That was an interesting challenge and thus DarwinPsk was born. At the moment, my development platform is a dual-boot iBook with Mac OS X and Gentoo Linux. All examples were taken from the Gentoo Linux part and KDE 3.2. Now I maintain two distributions and receive many proposals for improving the program. Of course, I receive bug reports as well. The latest documentation was version 0.6, dated January 2002, so something new is needed. Help is welcome in the form of writing user-oriented documentation, testing the program or making proposals for improvements or reporting bugs. Even implementing new modes or writing is possible. Improving existing programs or designing new modes fits well with the experimental nature of amateur radio. And, believe me, the ideas of open source fit the spirit of ham radio.

73 es bcnu on PSK31, de Volker, DL1KSV

Resources for this article: /article/7642.

Dr Volker Schroer came into contact with Linux in 1995 when setting up a firewall. Becoming a ham in 1996, he started developing LinPsk in 1999. Comments and proposals are welcome to dl1ksv@users.sourceforge.net.

Archive Index Issue Table of Contents

Advanced search

Driving the Mars Rovers

Frank Hartman

Scott Maxwell

Issue #125, September 2004

Linux is the desktop platform for the team driving NASA's Mars rovers, *Spirit* and *Opportunity*.

At 8:30 PM PST on January 3, 2004, a thousand pounds of metal sent from Earth blazed through the sky of Mars. Six minutes later, after sprouting a parachute and airbags, this chunk of metal slammed into the red planet, bounced 28 times over 300 meters and slowly rolled to a stop. The airbags deflated, revealing the metal pyramid they protected, which in turn unfolded to reveal a six-wheeled robotic geologist named *Spirit*.

So began a three-month exploration of Mars. For a team of hundreds of scientists and engineers at NASA's Jet Propulsion Laboratory, *Spirit* is serving as our eyes—and our toolbox, using the tools at the tip of its foldaway arm—during a new chapter in the exploration of our neighbor planet. As is *Spirit*'s twin, *Opportunity*, which duplicated *Spirit*'s performance three weeks later on the opposite side of Mars. And what are all those scientists and engineers using to drive the rovers? They are using Linux.

The Mars Exploration Rover (MER) mission marks a turning point for use of Linux in the space program. Linux has been used on space missions before—a Debian laptop rode the Space Shuttle on STS-83, for instance, as long ago as 1997. But the Mars Exploration Rover Project is the first JPL mission to use Linux systems for critical mission operations. On MER, Linux is being used for high-level science planning and for low-level command sequencing, visualization and simulation.

How We Got Here

As odd as this may seem, Linux originally was not intended to be our primary development platform. Our software initially was intended for use on the Mars '01 mission, which essentially would have been a repeat of JPL's 1997 Mars Pathfinder mission. The plan was to do all rover commanding on a single Silicon Graphics workstation, as had been done on Mars Pathfinder. Our Linux support was, at that time, mainly a hedge against SGI's fortunes.

After the failure of JPL's Mars '98 missions, however, JPL re-planned its Mars strategy. The Mars '01 plans were scrapped in favor of a later launch, the mission that eventually became MER. The upshot was we got two more years of development time than we had planned, albeit for a different spacecraft than we had planned. The MER rovers are larger, smarter and more complex than Pathfinder's *Sojourner* rover, and each one sports a robotic arm.

In those two years, Linux and the generic x86 hardware it runs on made enormous strides in both CPU and graphics speeds, thanks in no small part to graphics chip-maker NVIDIA and its strong Linux support. As a result, we increasingly came to favor the faster, better, cheaper solution that was Linux. We eventually purchased two high-end SGIs to add to our Linux boxes during flight operations, but it is unclear whether we will end up using them; the Linux boxes now give us equal, if not better, performance. As a result, the MER rovers are being commanded by multiple teams working in parallel on the dozens of Linux systems we've purchased for the mission.

The RSVP System

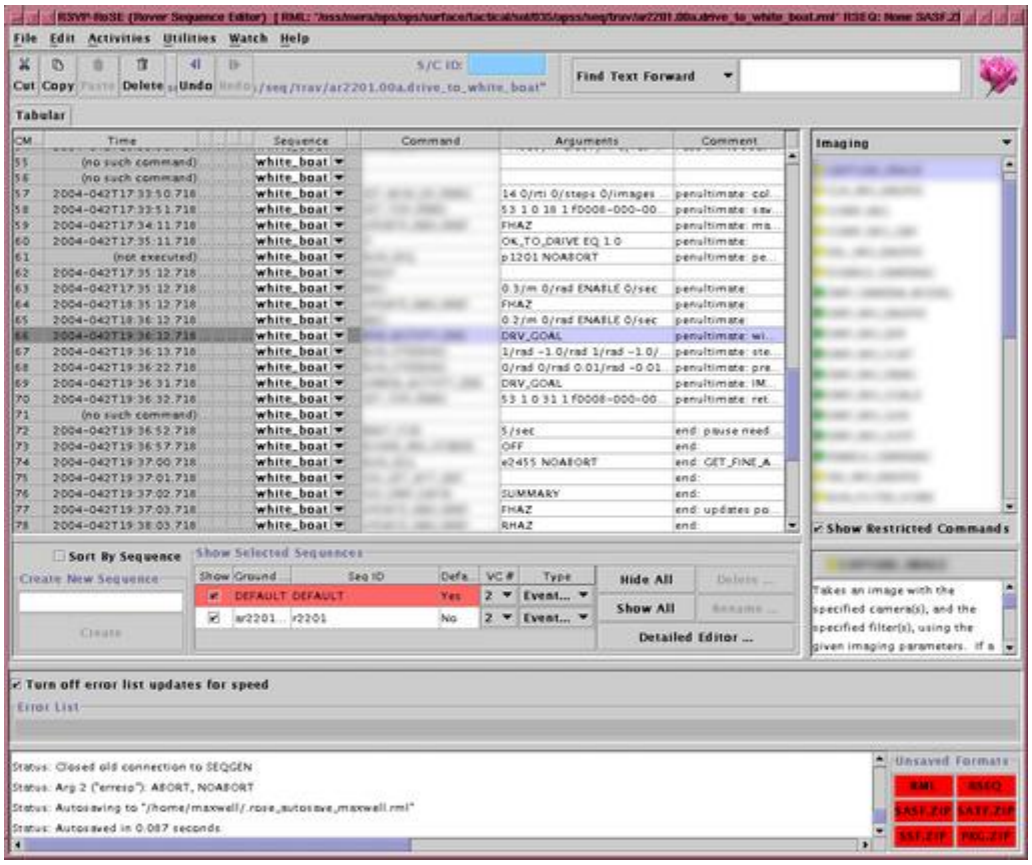
Our software, a suite of applications called the Rover Sequencing and Visualization Program (RSVP), was developed, tested and deployed on Linux. RSVP gives MER's engineers and scientists sophisticated tools for commanding Mars rovers. Because of the lengthy light-time delays, which amounted to nearly a 20-minute round-trip on *Spirit*'s landing day, it's not possible to drive the rovers interactively. Instead, RSVP provides an immersive environment that accurately displays the rover's environment and simulates its behavior. Throughout the Martian night, we Earthlings use RSVP to plan a day's activities at the command level and then uplink the resulting commands to the rover. The rover executes those commands during the following Martian day.

The two main components of RSVP are the Rover Sequence Editor (RoSE), which provides text-oriented commanding for all spacecraft commands, and HyperDrive, which provides advanced three-dimensional graphics for driving, arm motion and imaging commands. Each of the individual applications can run in a standalone mode, but they are more powerful when used together. In combined mode, the applications are run under the control of the parallel virtual machine (PVM) software developed by Oak Ridge National Laboratories. PVM provides a data bus, a way for the RSVP component applications to synchronize their work by passing messages to one another.

The content of most of our messages is based on RML, an XML-based specification designed specifically for representing rover command sequences. This message-passing approach has a number of benefits, not the least of which is that we can implement individual tools in different languages. As long as the tool can send PVM messages, it can be part of the suite. RoSE, our message-passing hub, and our message logger are written in Java; HyperDrive, our image viewer, and our sequence flow browser are written in C++ and C. For the next mission, we may try adding tools written in scripting languages such as Perl and Python.

RoSE

RoSE is a Java application running on the Sun Java Virtual Machine version 1.3.1; however, it's compiled not with Sun's compiler but with IBM's much faster Java compiler, Jikes. RoSE benefits from Java's well-known portability: we got the benefit of compiling and testing on a nice, fast Linux box and periodically installed and tested it on the SGI with little or no trouble. We even did an experimental port to Mac OS X, but abandoned it because we lacked time to support a third platform. There were no evident problems on Mac OS X, but if any had arisen, we wouldn't have had time to deal with them. We also had no requirement to support Mac OS X, so we dropped it altogether.


RSVP-RoSE (Rover Sequence Editor) | RML: "/oss/mera/ops/ops/surface/tactical/sol/030/apss/seq/trav/ar2201.00a.drive_to_white_boat.rml"

.../seq/trav/ar2201.00a.drive_to_white_boat"

CM	Time	Sequence	Command	Arguments	Comment
55	(no such command)	white_boat			
56	(no such command)	white_boat			
57	2004-042T17:33:50.718	white_boat	[redacted]	14 0/rti 0/steps 0/images ...	penultimate: col...
58	2004-042T17:33:51.718	white_boat	[redacted]	53 1 0 18 1 f0008-000-00...	penultimate: sw...
59	2004-042T17:34:11.718	white_boat	[redacted]	FHAZ	penultimate: ma...
60	2004-042T17:35:11.718	white_boat	[redacted]	OK_TO_DRIVE EQ 1.0	penultimate:
61	(not executed)	white_boat	[redacted]	p1201 NOABORT	penultimate: pe...
62	2004-042T17:35:12.718	white_boat	[redacted]		
63	2004-042T17:35:12.718	white_boat	[redacted]	0.3/m 0/rad ENABLE 0/sec	penultimate:
64	2004-042T18:35:12.718	white_boat	[redacted]	FHAZ	penultimate:
65	2004-042T18:36:12.718	white_boat	[redacted]	0.2/m 0/rad ENABLE 0/sec	penultimate:
66	2004-042T19:36:12.718	white_boat	[redacted]	DRV_GOAL	penultimate: wi...
67	2004-042T19:36:13.718	white_boat	[redacted]	1/rad -1.0/rad 1/rad -1.0/...	penultimate: ste...
68	2004-042T19:36:22.718	white_boat	[redacted]	0/rad 0/rad 0.01/rad -0.01...	penultimate: pre...
69	2004-042T19:36:31.718	white_boat	[redacted]	DRV_GOAL	penultimate: IM...
70	2004-042T19:36:32.718	white_boat	[redacted]	53 1 0 31 1 f0008-000-00...	penultimate: ret...
71	(no such command)	white_boat			
72	2004-042T19:36:52.718	white_boat	[redacted]	5/sec	end: pause need...
73	2004-042T19:36:57.718	white_boat	[redacted]	OFF	end:
74	2004-042T19:37:00.718	white_boat	[redacted]	e2455 NOABORT	end: GET_FINE_A...
75	2004-042T19:37:01.718	white_boat	[redacted]		end:
76	2004-042T19:37:02.718	white_boat	[redacted]	SUMMARY	end:
77	2004-042T19:37:03.718	white_boat	[redacted]	FHAZ	end: updates po...
78	2004-042T19:38:03.718	white_boat	[redacted]	RHAZ	end:

Show	Ground	Seq ID	Defa...	VC #	Type
✓	DEFAULT	DEFAULT	Yes	2	Event...
✓	ar2201...	r2201	No	2	Event...

Takes an image with the specified camera(s), and the specified filter(s), using the given imaging parameters. If a

✓ Turn off error list updates for speed

```
Status: Closed old connection to SEQGEN
Status: Arg 2 ("erresp"): ABORT, NOABORT
Status: Autosaving to "/home/maxwell/.rose_autosave_maxwell.rml"
Status: Autosaved in 0.087 seconds
```

Figure 1. RoSE Command Editor (Blurring due to ITAR Restrictions)

RoSE is aggressively data-driven, learning everything it knows about the rover from XML configuration files read at startup time. The IDE used for developing RoSE is good old tried-and-true GNU Emacs. Much of RoSE's GUI is built with hand-crafted Java code using Java's standard GUI toolkit, Swing. As nice as Swing is, hand-crafting GUI code sometimes is tedious; if a Java-supporting version of Glade had been available at the time, we likely would have used it. Still, important parts of the user interface are built dynamically. One of the XML configuration files RoSE reads at startup time is a description of the rover's command dictionary, sort of like an application programming interface (API) for the rover. RoSE uses this command dictionary to generate dialog boxes dynamically for editing each command, an approach that saved untold amounts of work as the command dictionary changed dramatically over its years of development.

The automation-friendly Linux environment contributed greatly to RoSE's reliability. Early on, we wrote extensive self-test code for RoSE and then set up a simple cron job to run the tests every night. If any self-tests failed, they were reported in an e-mail message to the developer the next morning. This helped us catch bugs early, while the changes that had caused them still were fresh in our minds. Our aggressive self-testing for RoSE was something of an experiment, and it is one that paid off handsomely.

As a side note, Java has a reputation for being slow, but we have to say that this reputation appears to be unfair. Although there are places where RoSE could be faster, this is true of any software. If we had more time (the constant refrain in software development), we could eliminate all of the bottlenecks. As it is, Java is fast enough for our purposes, and Linux's strong Java support has been a great benefit to us.

HyperDrive

HyperDrive is the interactive visualization component of the RSVP suite; it's the part they'll show on CNN. HyperDrive's main purpose is to enable the safe and efficient planning of rover driving and arm motions by giving the operator a good understanding of the rover's state and its current environment. This is accomplished by fusing multiple data sources into a three-dimensional scene and then providing multiple ways of viewing this scene.

One of the main sources of data for HyperDrive is terrain models generated from stereo imagery. Most of the cameras on the two rovers are stereo cameras, which means that they can be used to see with depth perception, much like human eyes. By using a technique known as stereo correlation, we can go from the flat, two-dimensional world of a picture into the three-dimensional world of rendered computer graphics. By combining these terrain models with CAD data representing the rover, we have the nucleus of a system that is able to render views of the rover interactively driving over the terrain or placing its arm on the Martian surface (Figure 2).

We further may combine these renderings with the raw imagery from the rover cameras to provide an augmented reality view, where the scene as viewed from the rover is overlaid with synthetic imagery and possibly viewed three-dimensionally using LCD shutter glasses (Figure 3). We also can render simulated views from any of the rover's cameras to help in image targeting.

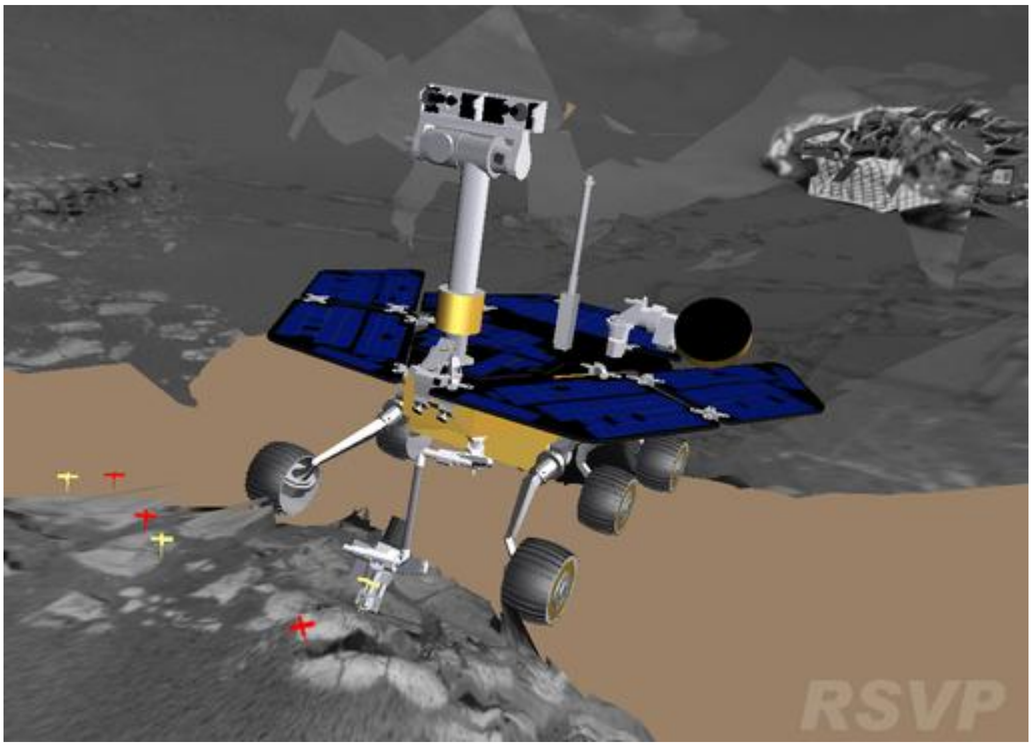


Figure 2. Drilling a Rock with the RAT (Rock Abrasion Tool)

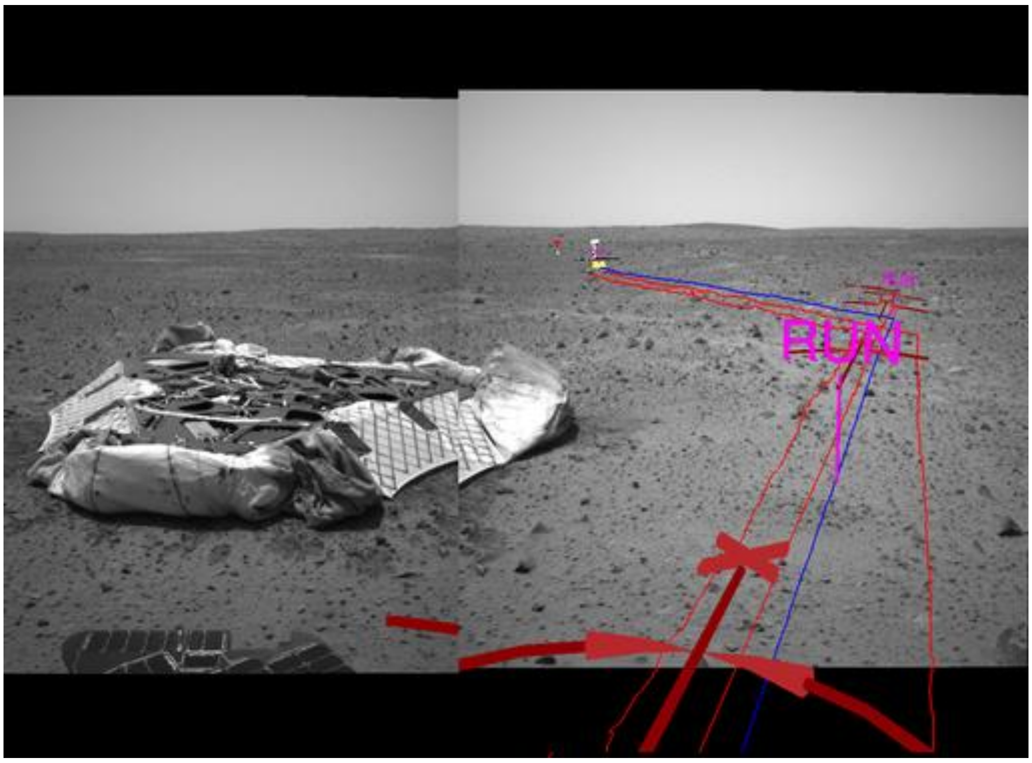


Figure 3. *Spirit* Leaving the Nest in Augmented Reality View

Into this virtual world we place icons and annotations, software-created objects that illustrate the command sequence and describe features of the world. The most important group of these objects is the sequence of icons that represents commands to be executed by the rover. While the text-oriented RoSE editor shows the entire command sequence for editing, HyperDrive shows only the subset of commands that have a sensible visual representation. In HyperDrive, mobility, robotic arm and imaging commands are shown as a string of icons placed across the terrain, each one located at the position where it will be executed. This allows us to program the rover's traverses across the terrain visually and to place its arm with a high degree of precision. Our first placement on the Martian surface was estimated to be within 5mm of its intended target. When we placed a second instrument on the same target, we did even better, placing it within 0.3mm of the first position. Not bad for a target a hundred million miles away.

Three-dimensional graphics programming under Linux started as quite a bleeding-edge adventure and matured into a capable platform with robust drivers and libraries. HyperDrive is based on the OpenGL Performer rendering API from Silicon Graphics. We use GTK+ and libglade for HyperDrive's user interface. The rapid prototyping enabled by Glade and libglade made GUI development painless and efficient, and we highly recommend this toolset to others in need of a robust interface toolkit. As our primary target evolved from IRIX to Linux, we were able to use more and more open-source tools and libraries and then find or build ports back to IRIX to maintain compatibility.

HyperDrive also is able to animate the rover's predicted actions, building an animation in real time and allowing the user to fly around and examine the rover's behavior from multiple perspectives. The same feature can be used to play back the rover's actual performance, based on the rover's daily report. And we were pleasantly surprised, to say the least, when these playback animations became a regular feature of JPL's daily press conferences.

Conclusion

Overall, Linux has evolved and surpassed our expectations substantially for it as a programming platform. The background of our team initially varied from experienced Linux advocates to complete Linux newbies, but we all gained new experience with Linux and the many open-source tools available to its users. Particularly with regard to accelerated OpenGL-based graphics, what started as a low-end alternative to high-priced graphics servers continues to outperform the big iron for our applications.

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference herein to any specific commercial product, process or service by trade name, trademark, manufacturer or otherwise does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

Frank Hartman currently is a software engineer specializing in computer graphics and a Mars rover driver at the Jet Propulsion Laboratory in Pasadena, California. He holds a Bachelor of Fine Arts Degree in Sculpture from the University of the Arts in Philadelphia and a Master of Science in Aeronautical and Astronautical Engineering from Stanford University. Frank resides in Altadena, California with his wife, Leah.

Scott Maxwell is a software developer and Mars rover driver at the Jet Propulsion Laboratory in Pasadena, California. He enjoys practicing aikido, reading classic literature and summarizing his own life in a few words. He is the author of *Linux Core Kernel Commentary*.

Archive Index Issue Table of Contents

Advanced search

The GPS Toolkit

Brian W. Tolman

Ben Harris

Issue #125, September 2004

Learn how GPS works and get a sharper fix on your position with this freely licensed library.

Explosive is perhaps the best term to describe the growth of the Global Positioning System (GPS) market in recent years. Contributing factors are numerous, and perhaps the most dramatic are economic: access to GPS is absolutely free and the cost of hardware continues to plummet. As a result, the GPS user can choose from a variety of devices that provide a position estimate. GPS has long been used, however, to explore topics beyond positioning; space weather, precise timing and continental drift are but three examples.

In order to use GPS for advanced topics or simply for improved positioning, the raw observations collected by the GPS receiver must be processed. In the past, the nuts and bolts of such processing have been left up to proprietary software. Now, a project called the GPS Toolkit, or GPSTk, is available under the LGPL to the Open Source and research communities. GPSTk is the by-product of GPS research conducted at the Applied Research Laboratories of the University of Texas at Austin (ARL:UT) since before the first satellite launched in 1978. It is the combined effort of many software engineers and scientists. Recently, the research staff at ARL:UT has decided to open source much of their basic GPS processing software as the GPSTk.

The Global Positioning System

The Global Positioning System actually is a US government satellite navigation system that provides a civilian signal. As of this writing, the signal is broadcast simultaneously by a constellation of 29 satellites, each with a 12-hour orbit. From any given position on the Earth, 8–12 satellites usually are visible at a time.

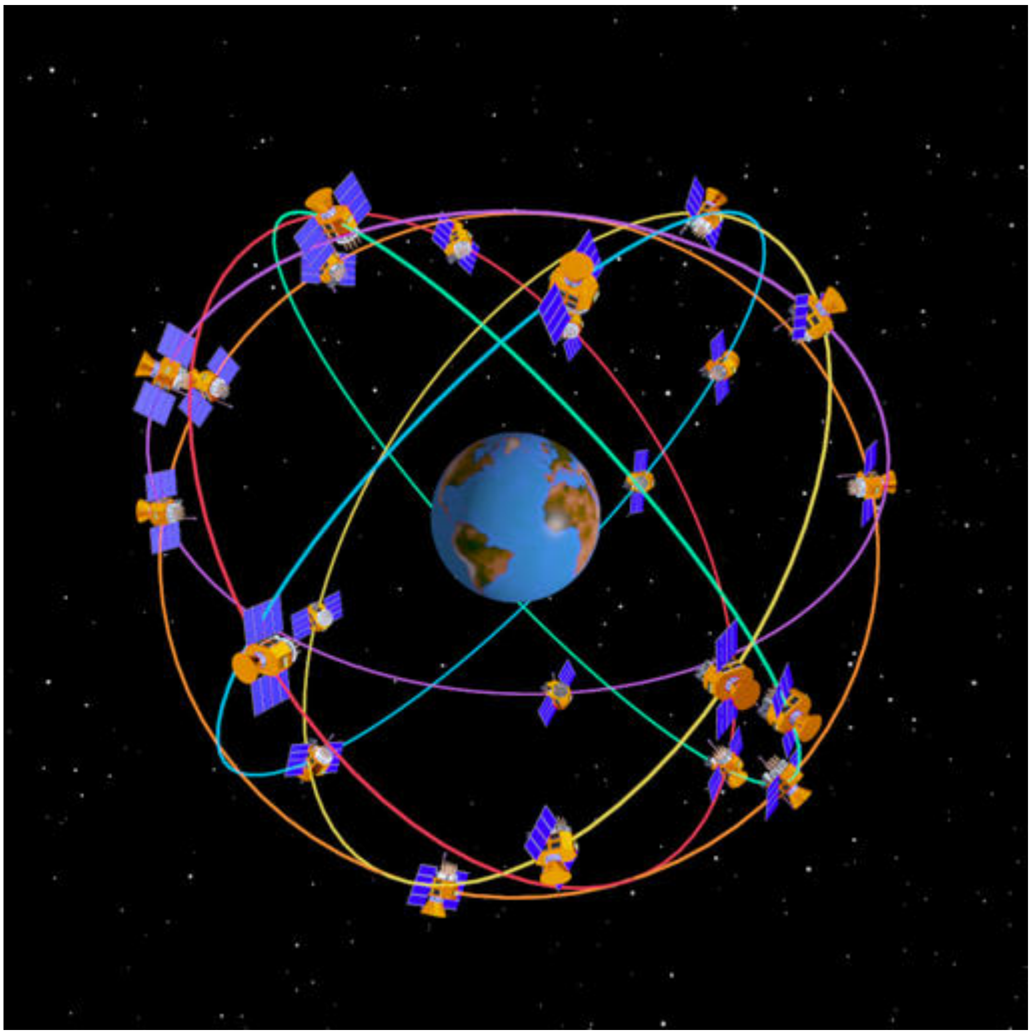


Figure 1. GPS Satellite Constellation Image from the Aerospace Corporation Web Site (www.aero.org/news/current/gps-orbit.html)

GPS in a Nutshell

Each satellite broadcasts spread spectrum signals at 1,575.42 and 1,227.6MHz, also known as L1 and L2, respectively. Currently, the civil signal is broadcast only on L1. The signal contains two components, a time code and a navigation message. By differencing the received time code with an internal time code, the receiver can determine the distance, or range, that the signal has traveled. This range observation is offset by errors in the (imperfect) receiver clock; therefore, it is called a pseudorange. The navigation message contains the satellite ephemeris, which is a numerical model of the satellite's orbit.

GPS receivers record, besides the pseudorange, a measurement called the carrier phase, or phase. The phase also is a range observation like the pseudorange is, except it has an unknown constant added to it, the phase ambiguity. It also is much smoother, having about 100 times less measurement noise than the pseudorange, which makes it useful for precise positioning. Because of the way it is measured, the phase is subject to random, sudden jumps. These discrete changes always come in multiples of the wavelength of the GPS signal and are called cycle slips.

The Position Solution

The standard solution for the user location requires a pseudorange measurement and an ephemeris for each satellite in view. At least four measurements are required, as there are four unknowns: three coordinates of position plus the receiver clock offset. The basic algorithm for the solution is described in the official GPS Interface Control Document, ICD-GPS-200. The position solution is corrupted due to two sources of error, errors in the observations and errors in the ephemeris.

Reducing Measurement Errors

The GPS signal travels through every layer of the Earth's atmosphere. Each layer affects the signal differently. The ionosphere, which is the high-altitude, electrically charged part of the atmosphere, introduces a delay, and therefore a range error, into the signal. The delay is frequency-dependent, so it can be computed directly if you have data on both the GPS frequencies. There also is a delay due to the troposphere, the lower part of the atmosphere. This delay also can be modeled and removed. Many other errors are associated with the GPS signal. Multipath reflections and relativistic effects are two examples.

More precise applications reduce the effect of error sources by a technique referred to as differential GPS (DGPS). By differencing measurements simultaneously collected by the user and a nearby reference receiver, the errors common to both receivers (most of them) are removed. The result of DGPS positioning is a position relative to the reference receiver; adding the reference position to the DGPS solution results in the absolute user position.

The alternative to DGPS is to model and remove errors explicitly. Creating new and robust models of phenomena that affect the GPS signal is an area of active research at ARL:UT and other laboratories. The positioning algorithm can be used to explore such models. Essentially, the basic approach is to turn the positioning algorithm inside out to look at the corrections themselves. For example, observations from a network of receivers can create a global map or model of the ionosphere.

Improved Ephemerides

The GPS position solution can be improved by using a better satellite ephemeris. The US National Geospatial-Intelligence Agency (NGA) generates and makes publicly available a number of precise ephemerides, which are more accurate satellite orbits. Satellite orbits described by the broadcast navigation message have an error on the order of meters, and the precise ephemeris has decimeter accuracy. The International GPS Service (IGS) is a global civil cooperative effort that also provides free precise ephemeris products. Global networks of tracking stations produce the observations that make generation of the precise ephemerides possible.

GPS Data Sources

GPS observation data from many tracking stations are available freely on the Internet. Many such stations contribute their data to the IGS. In addition, many networks of stations also post their data to the Internet, such as the Australian Regional GPS Network (ARGN) and global cooperatives including NASA's Crust Dynamics Data Information System (CDIS).

GPS File Formats

Typically, GPS observations are recorded in a standardized format developed by and for researchers. Fundamental to this format is the idea that the data should be independent of the type of receiver that collected it. For this reason, the format is called receiver independent exchange, or RINEX. Another format associated with GPS is SP-3, which records the precise ephemeris. The GPSTk supports both RINEX and SP-3 formats.

GPS Receivers and Open Source

GPS receivers have become less expensive and more capable over the years, in particular handheld and mobile GPS receivers. The receivers have many features in common. All of the receivers output a position solution every few seconds. All receivers store a list of positions, called waypoints. Many can display maps that can be uploaded. Many can communicate with a PC or handheld to store information or provide position estimates to plotting software.

Typically, communication with a PC and other system follows a standard provided by the National Marine Electronics Association, called NMEA-0183. NMEA-0183 defines an ASCII-based format for communication of position solutions, waypoints and a variety of receiver diagnostics. Here is an example of a line of NMEA data, or sentence:

```
$GPGLL,5133.81,N,00042.25,W*75
```

The data here is a latitude, longitude fix at 51° 33.81 min North, 0° 42.25 min West. The last part is a checksum.

As a public standard, the NMEA-0183 format has given the user of GPS freedom of choice. NMEA-0183 is the format most typically used by open-source applications that use receiver-generated positions.

Closed standards also are common. SiRF is a proprietary protocol licensed to receiver manufacturers. Many receiver manufacturers implement their own binary protocols. Although some of these protocols have been opened to the public, some have been reverse engineered. GPSBabel is an open-source project to communicate with consumer-grade receivers. The Sharc Project is a similar project to provide communication with survey-grade receivers.

A number of interesting open-source applications are available that utilize consumer-grade receivers. With one, you can use open-source applications to navigate in your car. The GPS Drive Project helps you do that, using a graphical map. GPS Drive also can be linked to the Festival application to get driving directions in the form of speech output. Internet sites such as WiGLE.net have lists of the geographic coordinates of open wireless LANs; you can use your GPS unit to find these.

Traditionally, DGPS is accomplished with two or more receivers that communicate position information with radio waves. You can do DGPS over IP now, using open-source applications. The open-source project called gpsd essentially broadcasts NMEA-0183 sentences over TCP/IP. The gps3d Project, which allows you to visualize your position and the configuration of GPS in 3-D, also can utilize a gps3d server.

All of these applications are based on standard positioning. To move your positioning capability to the next level, you have to work directly with the observations made by the receiver. Only a few open-source or freely available programs exist that give the user this freedom. OpenSourceGPS is a project to create a GPS receiver based on the Zarlink chipset. teqc from UNAVCO performs quality assurance and processes raw data from receivers to generate RINEX, but it is closed source. In contrast, the purpose of the GPSTk is to give the user the ability to manipulate not only GPS observations but also to improve the processing algorithms.

The GPS Toolkit

The GPS Toolkit (GPSTk) is coded entirely in ANSI C++. It is platform-independent and has been built and tested on Linux, Solaris and Microsoft Windows. Everything needed to write standalone, console-based programs is included, along with several complete applications.

The design is highly object-oriented. Everything is contained in the namespace gpstk::. For example, reading and writing a RINEX observation file is as simple as this:

```
// open, read and re-write a RINEX file
using namespace gpstk;
// input file stream
RinexObsStream rin(inputfile);
// output file stream
RinexObsStream rout(outputfile,
ios::out|ios::trunc);
DayTime nextTime; //Date/time object
RinexObsHeader head; //RINEX header object
RinexObsData data; //RINEX data object

// read the RINEX header
rin >> head;
rout.header = rin.header;
rout << rout.header;

// loop over all data epochs
while (rin >> data) {
nextTime = data.time;
// change obs data&
rout << data;
}
```

The core capability of the library is built around RINEX file I/O. It also includes a complete date and time class to manipulate time tags in GPS and many other formats.

In addition to the RINEX I/O, GPSTk includes classes for handling geodetic coordinates (latitude and longitude) and GPS ephemeris computations. There also is a complete template-based Matrix and Vector package. And, of course, there are GPS positioning and navigation algorithms, including several tropospheric models.

Finally, several standalone programs are included in the distribution. Included are utilities to validate or modify RINEX files, a summary program, a utility to remove or modify observations, a phase discontinuity corrector and a program to compute standard errors and corrections, such as the total electron content (TEC) of the ionosphere along the signal path.

Getting Started with the GPS Toolkit

The GPS Toolkit is available for download as a tarball (see the on-line Resources section). To build the toolkit you need to use jam, a replacement for make, and Doxygen, a source code documentation generator. The entire build sequence looks like the following:

```
tar xvzf gpstk-1.0.tar.gz
cd gpstk
jam
doxygen
```

```
su
jam -sPREFIX=/usr install
```

This sequence builds and installs the GPSTk dynamic and shared libraries, as well as the header files, in the /usr tree. In addition, a doc subdirectory is created, containing HTML-based documentation of the GPSTk library.

Below are three example applications of the GPSTk created at ARL:UT. The second example actually is distributed as an application with the GPSTk.

Enhanced Positioning

Position solutions generated by the GPSTk provide improved precision and robustness compared to those generated by a GPS receiver. Figure 2 illustrates the benefits; each axis extends from –10 to 10 meters.

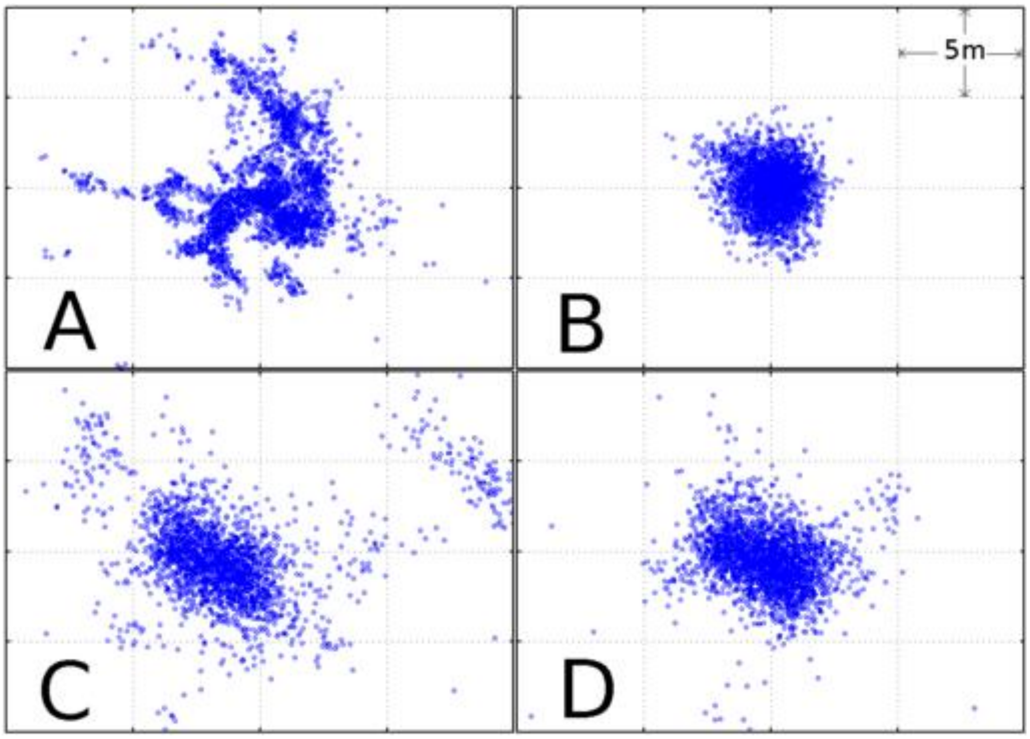


Figure 2. Left: positions from a GPS receiver. Right: positions generated using GPSTk algorithms.

Plot A shows position computations and how they vary along the East and North directions. Such results are representative of solutions created with a consumer-grade GPS receiver. Plot B shows how the position estimate improves when atmospheric delays are accounted for. Direct processing not only improves precision, but it also increases robustness. Plot C shows the effect of a faulty satellite. The faulty satellite is detected and removed using the GPSTk in Plot D.

Carrier-Phase Discontinuity Correction

An important problem in GPS data processing involves discontinuities in the carrier phase. Before phase data can be used, cycle slips must be found and fixed. The GPSTk distribution includes an application called a discontinuity corrector that does just that. This feature is available in the library as well.

The GPSTk discontinuity corrector works by forming two useful linear combinations of the dual-frequency phase data, called the wide-lane phase bias and the geometry-free phase. An example of these for normal data is shown in Figure 3. The wide-lane bias (red) is noisy but has a constant average. The geometry-free phase does not depend on the receiver-satellite geometry, but it depends strongly on the ionospheric delay. In fact, it is proportional to that delay. Normally, the ionosphere is quiet and smooth, but at times it can be active and rough; then this quantity can vary wildly. The geometry-free phase and the wide-lane noise increase at both ends of the dataset, because the satellite is rising or setting there. Consequently, the signal must travel through more atmosphere.

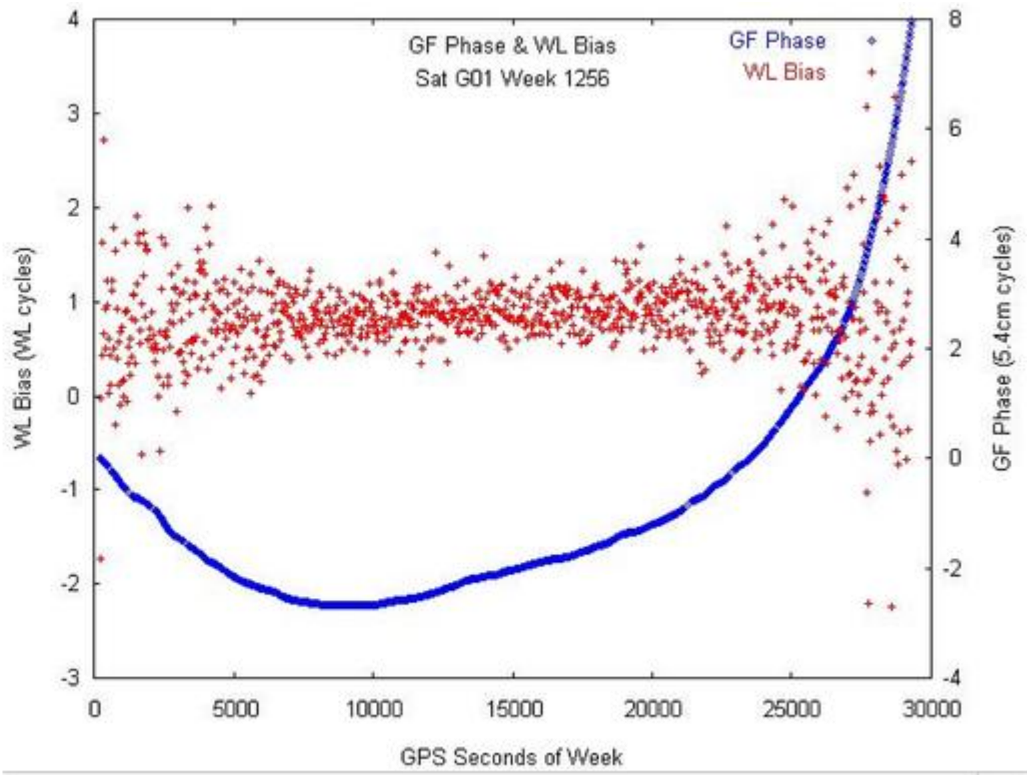


Figure 3. Normal wide-lane (red) and geometry-free (blue) phases for one satellite.

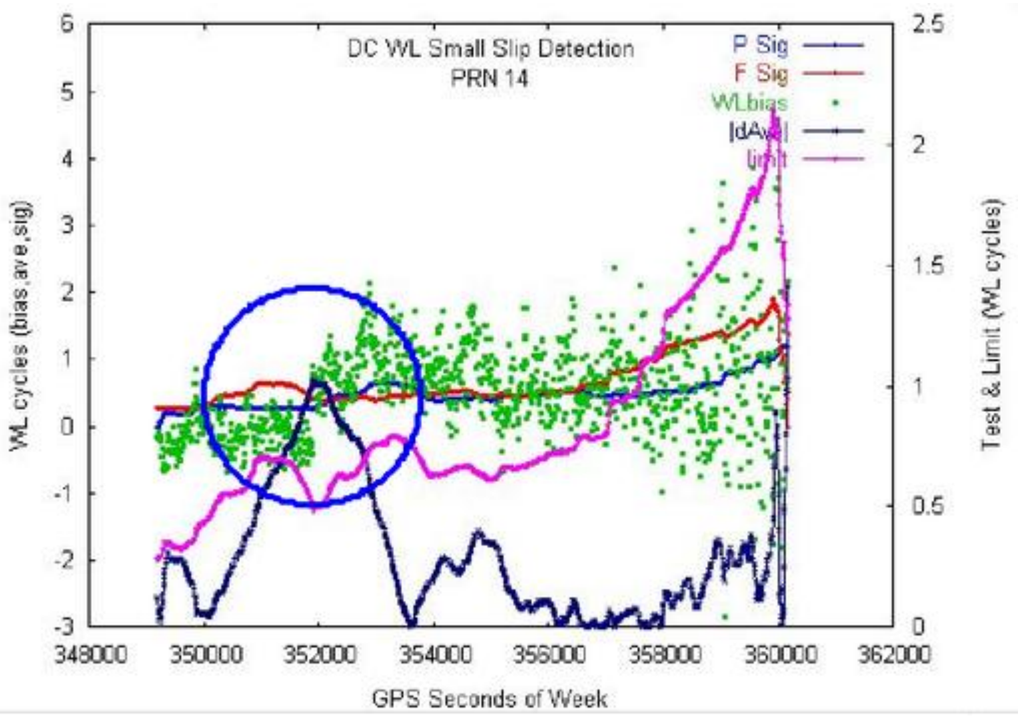


Figure 4. Slip detected (blue circle) in the wide-lane data (green) where test quantity (dark blue) is larger than limit (magenta).

The discontinuity corrector works by first looking for slips in the wide-lane phase bias; Figure 4 illustrates a case in which it found one. When a slip in the wide-lane slip is found, the code turns to the geometry-free phase and looks for the slip there. To estimate the size of the slip, low-order polynomials are fit to the data on each side of the slip, extrapolated to the point where the slip occurred and then differenced.

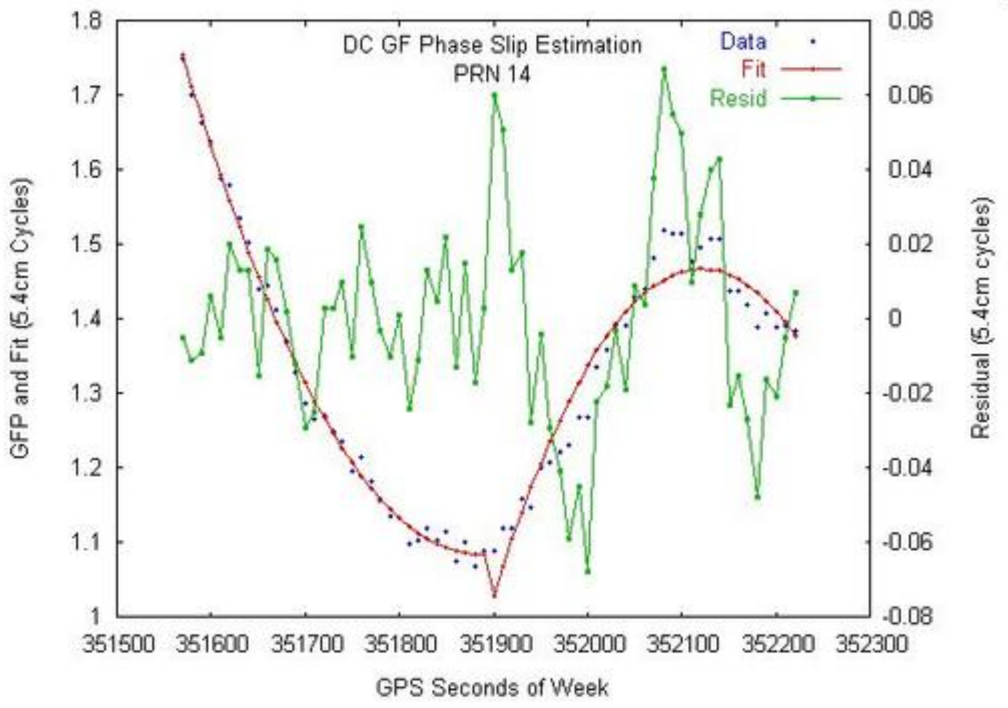


Figure 5. Estimation of the Cycle Slip Size

Satellite Position Interpolation

Another GPSTk application at ARL:UT involves a satellite in low-earth orbit that carries a GPS receiver. This satellite collects GPS data both for the satellites above it, referred to as top-side data, and those visible below it, or bottom-side data. The GPS signal for the bottom-side data has a long path length through the atmosphere, which is ideal for remote sensing of the atmosphere. The top-side data is used for computing the LEO satellite's rapidly changing position as it orbits Earth. A problem arises here as the top-side data is collected with a lower data rate (10 seconds) than on the bottom-side (1 second), yet the position of the LEO satellite is needed for processing the bottom-side data at the higher data rate. To solve the problem, a program was written, using only GPSTk, which reads the GPS data, computes the LEO position with the top-side data and then interpolates those positions to 1-second epochs. The result is shown in Figure 6, a plot of the position of the LEO satellite as it orbits Earth.

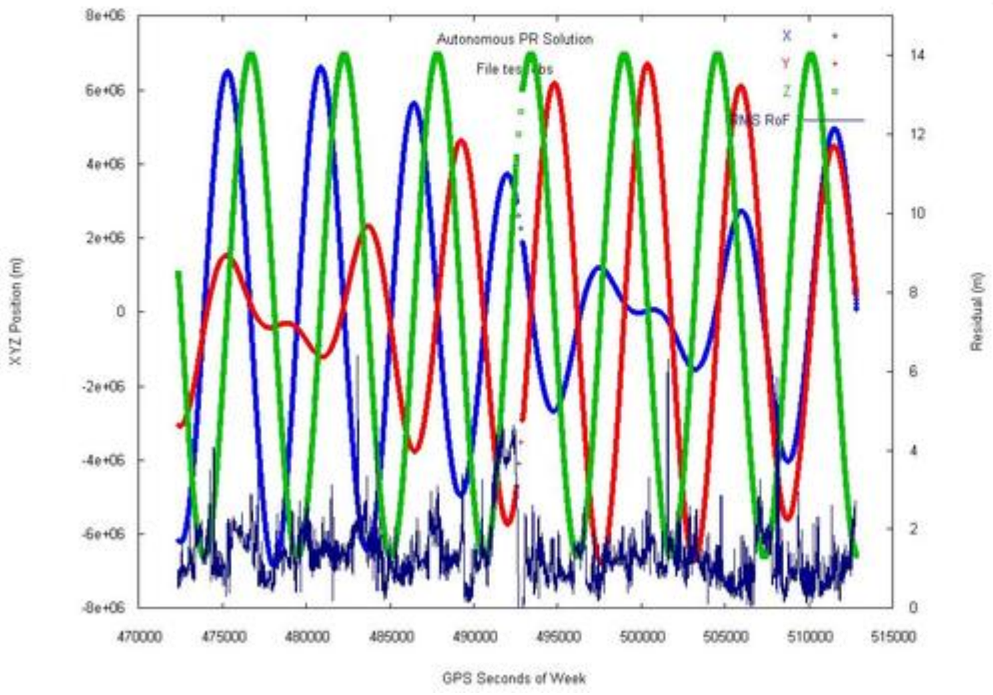


Figure 6. Position Interpolation

The Future of GPSTk

Open-source GPS processing, on the scale anticipated for the GPS Toolkit, is unprecedented; we are excited by the prospect of what could develop. GPSTk potentially has a broad range of audiences. Universities can use the GPSTk to process GPS data with open-source code. Embedded developers can develop software to perform GPS positioning and to read, write and edit RINEX data files. Researchers may find that this code is an excellent foundation for GPS receivers implemented entirely in software, called software receivers.

Although the growth of the GPSTk will depend strongly on user feedback and participation, changes also will be driven by shifts in the satellite navigation arena. In the near term, the first satellite to provide dual-frequency pseudoranges to civilians is scheduled for launch in 2005. Furthermore, the European community is creating Galileo, which will provide a public-regulated service compatible with GPS, essentially augmenting the current constellation with a new one. In the long term, GPS will have new signals in the L5 and M code. GPSTk, with its emphasis on fundamental observations, can provide the basis to explore and exploit these changes.

It is our hope that university students, laboratory researchers, system engineers and software developers will contribute to, as well as benefit from, the GPS Toolkit. We already have seen many benefits to using this code within our lab and believe that the GPSTk will inspire a number of innovative GPS applications.

Resources for this article: /article/7651.

Dr Brian W. Tolman is a research scientist at Applied Research Laboratories, The University of Texas, with 18 years experience in GPS-related research, data analysis and software development. He holds a PhD in theoretical physics from The University of Texas at Austin.

Ben Harris is an Engineering Scientist at UT Austin. When he is not researching GPS, studying for a PhD or spending time with his beautiful family, he reprograms animatronic fishes to perform scenes from *Pulp Fiction*.

Ximba Radio: Developing a GTK+/Glade GUI to XM Satellite Radio

Michael J. Hammel

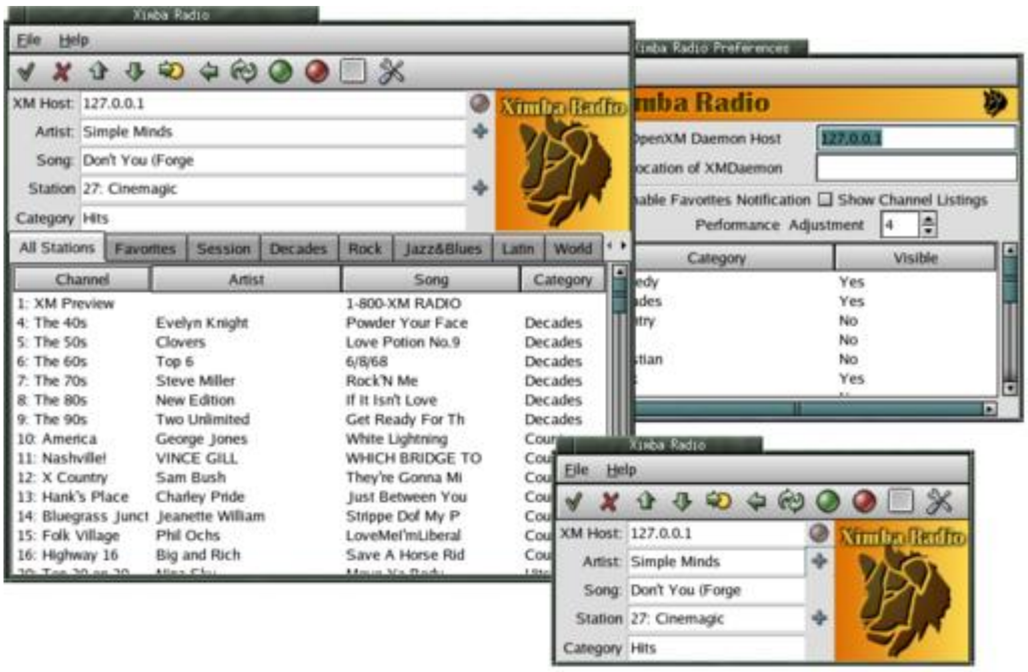
Issue #125, September 2004

They say Glade can make desktop application prototyping a quick and painless process. To scratch my own itch with XM Satellite Radio on my PC, I decided to see exactly how fast and painless.

As American TV slowly sinks into the abyss of fictional reality, you may find yourself, like me, returning to the roots of electronic entertainment—radio. Satellite radio is the latest incarnation of this medium, offering a wide range of stations accessible from nearly anywhere you can drive your car.

Because I spend more of my time in front of a monitor than behind a steering wheel, I was fortunate to find a PC-based solution for satellite radio. XMPCR is a USB-connected receiver for the XM Radio satellite system that is sold primarily for Microsoft Windows systems. The device is supported under Linux by the OpenXM Project, a set of Perl scripts that acts as a network dæmon for controlling the device. Unfortunately, the only user interface for the dæmon is a limited text-based tool.

A Preview of the Project Results



XM Host: 127.0.0.1
Artist: Simple Minds
Song: Don't You (Forge
Station 27: Cinemagic
Category Hits

Channel	Artist	Song	Category
1: XM Preview		1-800-XM RADIO	
4: The 40s	Evelyn Knight	Powder Your Face	Decades
5: The 50s	Clovers	Love Potion No.9	Decades
6: The 60s	Top 6	6/8/68	Decades
7: The 70s	Steve Miller	Rock'N Me	Decades
8: The 80s	New Edition	If It Isn't Love	Decades
9: The 90s	Two Unlimited	Get Ready For Th	Decades
10: America	George Jones	White Lightning	Coun
11: Nashville!	VINCE GILL	WHICH BRIDGE TO	Cou
12: X Country	Sam Bush	They're Gonna Mi	Cou
13: Hank's Place	Charley Pride	Just Between You	Cou
14: Bluegrass Junct	Jeanette William	Strippe Dof My P	Cou
15: Folk Village	Phil Ochs	LoveMel'mLiberal	Cou
16: Highway 16	Big and Rich	Save A Horse Rid	Cou

penXM Daemon Host 127.0.0.1
ocation of XMDaemon
able Favorites Notification ☐ Show Channel Listings
Performance Adjustment 4

Category	Visible
edy	Yes
ades	Yes
try	No
	No
tian	No
	Yes

Figure 1. The main window, channels listings and preferences. The second version shows the minimalist form, with the channel listings hidden.

Ximba Radio was born as a graphical front end to OpenXM. The application provides a minimalist main window with current channel information that can be expanded to show channel listings and user favorites. A button bar across the top offers easy management of the radio and station navigation as well as quick access to configuration options. The menu bar gives users the comfort of a traditional desktop application.

Channel listings are shown in multiple formats. A main channel listing window shows all channels, and category-specific tabs show related channels. Separate tabs provide access to user-selectable artist and channel favorites along with a current session history. Category tabs can be hidden using the Preferences dialog, which also allows a user to set performance settings and favorites notices.

On the back side of Ximba Radio is the OpenXM dæmon. This Perl script and associated Perl module drive the connection to the USB port. The dæmon reads from a configuration file or can take command-line arguments. Communication with the dæmon occurs on a configurable TCP port with a list of acceptable clients. The dæmon also can run on Windows systems.

Design Goals

My goals for this application were to match the functionality of the Windows version, provide a minimalist interface and be simple to configure. Also, implementation would need to be completed in less than one man-week (40 hours).

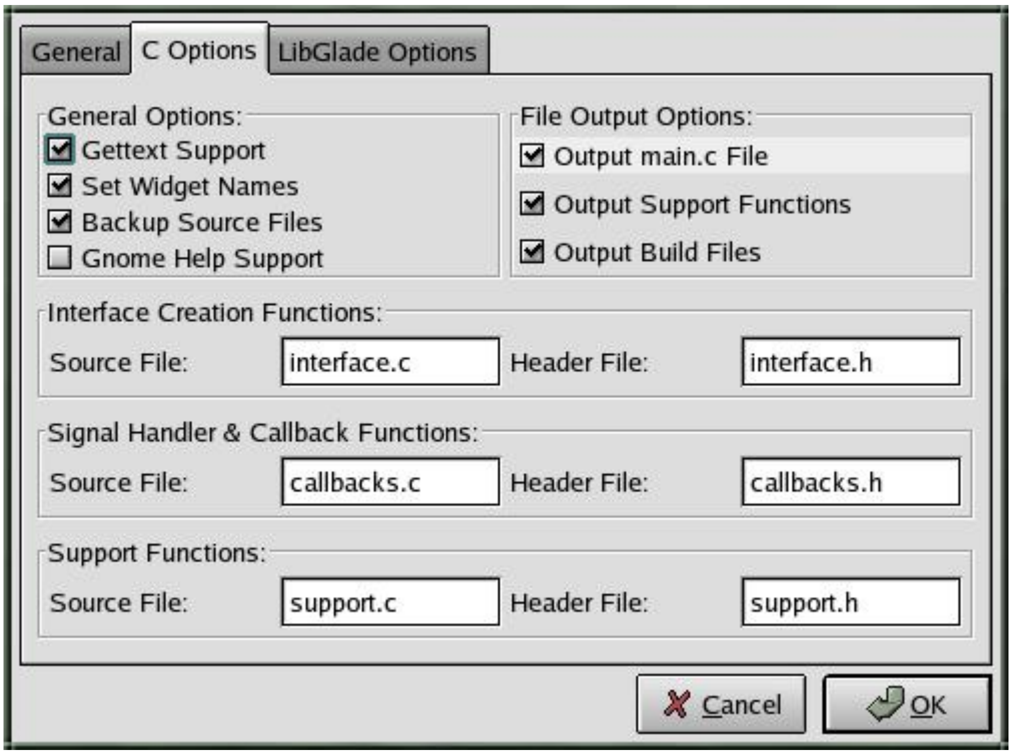
Another goal was to keep the user-interface code as independent of the application code as possible. Application code should be able to be used with any suitable user interface, so a good design could have a curses or Web interface dropped on top with little additional work. This goal is in line with GNOME development guidelines as well as future plans for Glade.

To meet these goals, I planned to use a single header file and a single C module. To speed the implementation and solve some time-consuming problems, I opted to use global variables. Remember, this is a prototype of a simple desktop application, not an enterprise-ready 24/7 fault-tolerant behemoth. If managed correctly, the use of globals can be removed with future updates.

Getting Familiar with Glade

With goals in hand, I started working with Glade to sculpt the user interface; I cover specific code details in the next section. I configured Glade to generate C code and took the default settings for everything else in the Options dialog. Most important here are the source and header files that hold the user-interface definitions (interface.c) and the widget callbacks (callbacks.c), along with the option to generate a main.c file.

Glade generates a complete build environment for building an application. This includes the source directory (src) and a directory for image files (pixmaps). Image files used in GtkImage widgets need to be in .xpm format. Other generated files include autoconf and automake templates and pot files for internationalized strings.


General | C Options | LibGlade Options

General Options:
- ☑ Gettext Support
- ☑ Set Widget Names
- ☑ Backup Source Files
- ☐ Gnome Help Support

File Output Options:
- ☑ Output main.c File
- ☑ Output Support Functions
- ☑ Output Build Files

Interface Creation Functions:

Source File: interface.c Header File: interface.h

Signal Handler & Callback Functions:

Source File: callbacks.c Header File: callbacks.h

Support Functions:

Source File: support.c Header File: support.h

Cancel OK

Figure 2. The Options dialog for Glade allows you to generate code, specify filenames and choose support for internationalized text.

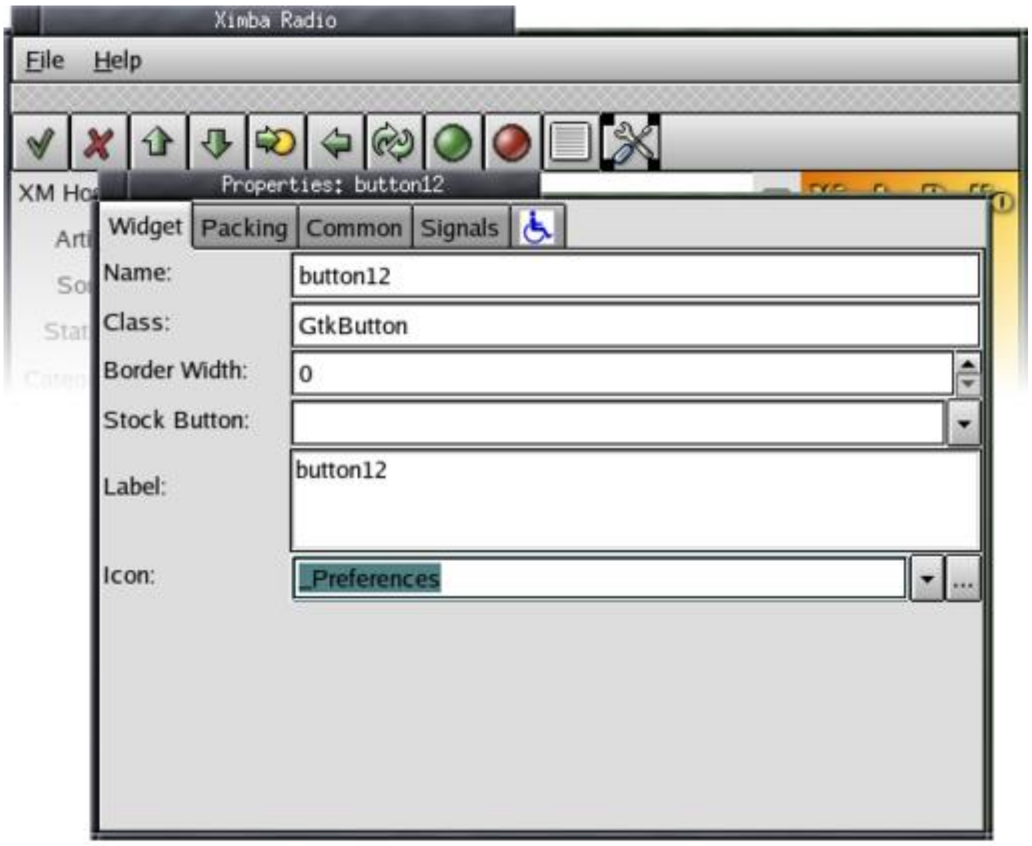
Internationalization is optional and handled with GNU gettext support. The interface.c file, for example, has gettext-enabled strings. Even with this option enabled, you don't have to create pot files for any language; Glade simply uses any text strings you provide as the default language.

I found that prototyping Ximba Radio with Glade required hand-editing only two of the generated files, main.c and callbacks.c. The former required only minor additions for initialization options related to configuration handling for the application. The callbacks.c file was modified mostly to pass calls to my C module, utils.c.

As I edited my project in Glade and regenerated the C code, the callbacks.c file was appended with new functions. Fortunately, Glade does a good job of knowing when a callback already exists and didn't destroy any of my changes. Unfortunately, it sometimes re-adds an existing function. It was necessary to pull out those extra functions manually from time to time. When using libGlade, which processes the Glade XML file directly instead of using generated C code, this problem does not exist. Discussion of libGlade is beyond the scope of this article.

UI Design with Glade

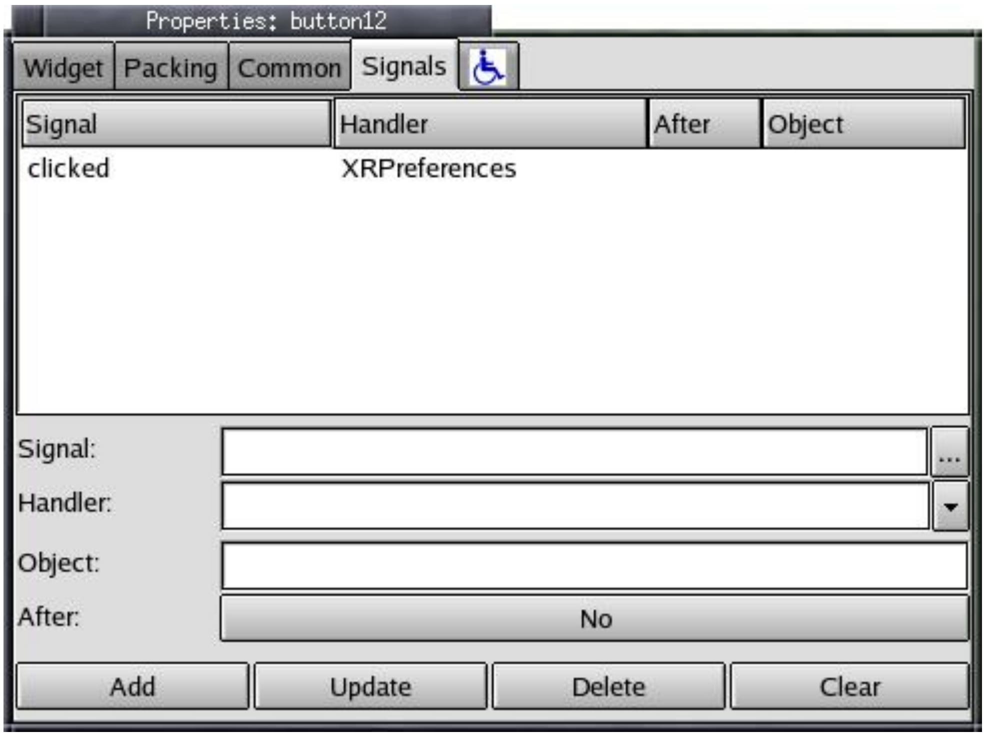
Ximba Radio required two primary windows, the Main Window and the Preferences Dialog, and a number of secondary pop-up windows. The Main Window's button bar was created with Glade's toolbar widget, and the buttons were added to that manually. GTK+ buttons can have text or images. Glade allows a choice of application images, stock buttons or stock icons. Stock buttons use the same icons as stock icons except that tooltips are not available. Because of this, I suggest using stock icons and leaving the stock button field blank.



Name: button12
Class: GtkButton
Border Width: 0
Stock Button:
Label: button12
Icon: _Preferences

Figure 3. Glade's toolbar widget and the properties used to set icons in buttons inside a toolbar.

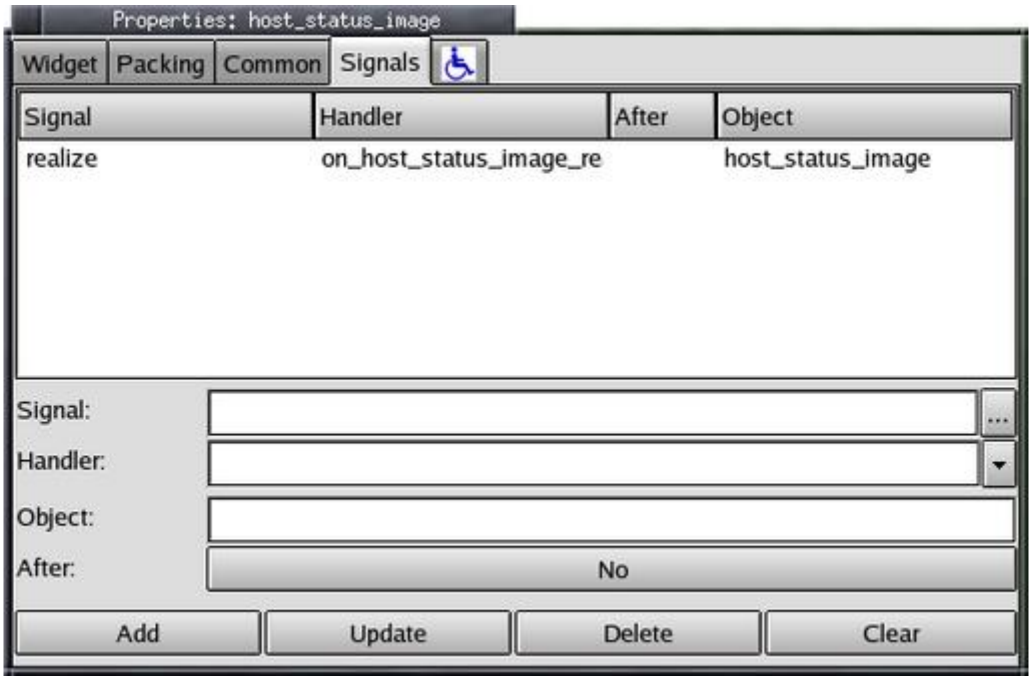
Each button in the toolbar has a single callback function attached to the click signal. The callback function can have any name and, if desired, be passed the name of the widget itself as an argument. For callbacks attached to click signals, the latter is not necessary. In callbacks attached to realize signals, which I discuss in a moment, the widget name is passed to the function.


Properties: button12

Signal	Handler	After	Object
clicked	XRPreferences		

Figure 4. Buttons in the toolbar have a single callback attached to the clicked signal.

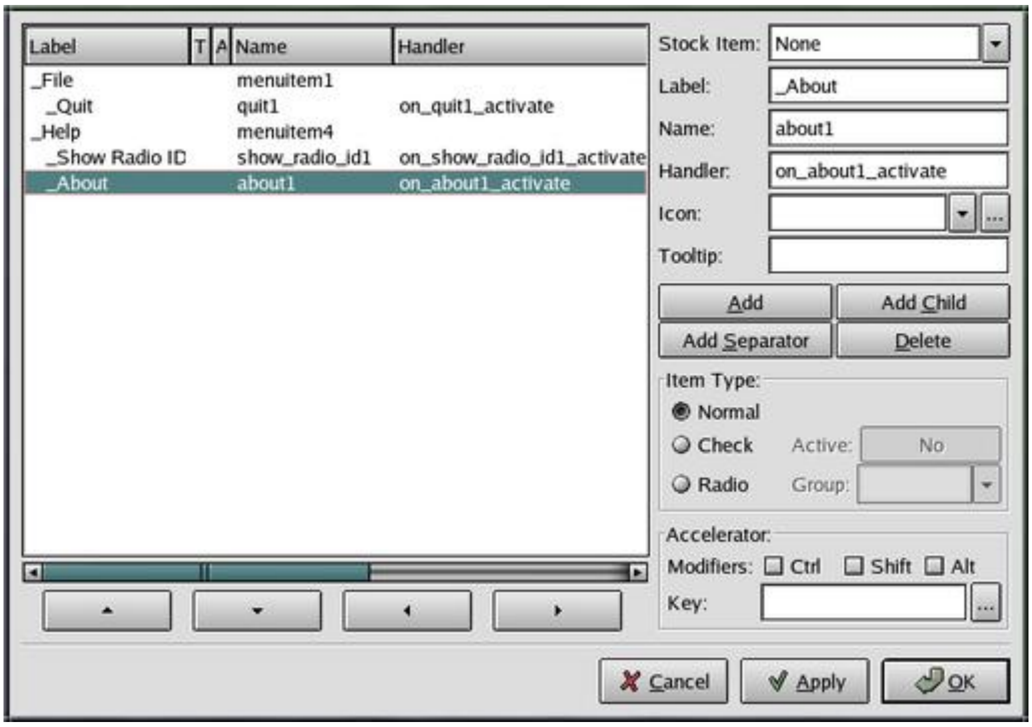
I added three GtkImage widgets in the Main Window. The first is a State icon positioned to the right of the Host name field. I set this to the Remove icon—Glade offers many stock icons—to show no connection to the dæmon. To show a connected state I used the Apply icon. In order to change the icon at runtime, I saved the widget ID of this GtkImage in a realize callback. During normal use, this icon also can be changed to the Off stock icon in order to show a connected but muted state. I examine the code for handling these changes in the next section.


Properties: host_status_image

Signal	Handler	After	Object
realize	on_host_status_image_re		host_status_image

Figure 5. The State icon needs a realize callback to get the name of the widget so we can update it at runtime.

The Favorites buttons are plus signs. Glade and GTK+ call these Add icons. These buttons have a single callback attached to their click signal. The callback adds the current artist or channel to the appropriate list of favorites. The menu bar at the top of the main window was created using Glade's built-in Menu Editor. The editor has many options, but for this project I used only the Label, Name and Handler options, the latter to define the function to be called when the menu item was selected.



Label	T	A	Name	Handler
_File			menuitem1	
_Quit			quit1	on_quit1_activate
_Help			menuitem4	
_Show Radio ID			show_radio_id1	on_show_radio_id1_activate
_About			about1	on_about1_activate

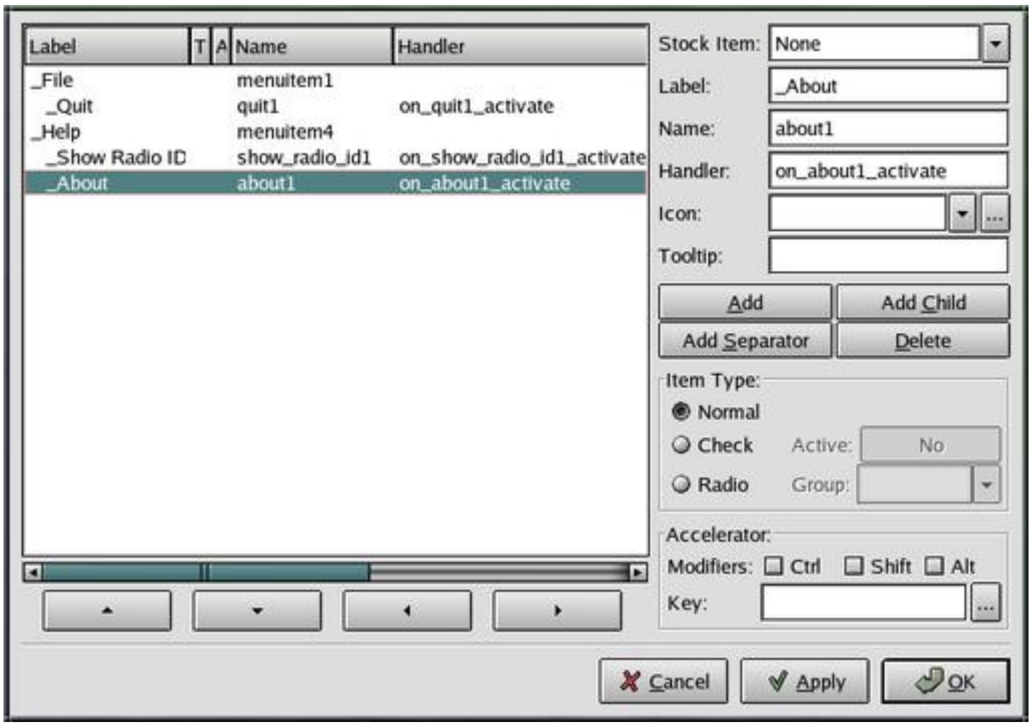
Stock Item: None
Label: _About
Name: about1
Handler: on_about1_activate
Icon:
Tooltip:

Item Type: [✓ Normal] Check Radio
Active: No
Group:

Accelerator:
Modifiers: ☐ Ctrl ☐ Shift ☐ Alt
Key:

Figure 6. Glade's Menu Editor offers many options, but only Label, Name and Handler are necessary for our prototype.

A notebook widget provides access to the complete channel listings, as well as category, favorite and session-specific listings. All of these are provided through the CList widget. Glade fully supports this widget even though GTK+ prefers that new code use the newer and more complex Tree and List widget. I cover this controversial decision in more detail a little later.



Label	T	A	Name	Handler
_File			menuitem1	
_Quit			quit1	on_quit1_activate
_Help			menuitem4	
_Show Radio ID			show_radio_id1	on_show_radio_id1_activate
_About			about1	on_about1_activate

Stock Item: None
Label: _About
Name: about1
Handler: on_about1_activate
Icon:
Tooltip:

Item Type:
● Normal
○ Check Active: No
○ Radio Group:

Accelerator:
Modifiers: ☐ Ctrl ☐ Shift ☐ Alt
Key:

Figure 7. Glade fully supports the CList widget, which is preferred over the Tree and List widget for simple list management.

Coding Process

Glade generates empty functions for callbacks, often referred to as stub functions. The stub functions make it possible to follow a simple process in prototype development: design the UI, generate code, write callbacks, test and repeat. I left most of the callback coding—aside from menu quit functions—until after the UI was complete. Later, I went back and filled in the callbacks. This methodology allowed me to experiment with the layout of the application before having to get involved too deeply with what that layout actually would do. Again, this is part of the whole goal of separating user-interface code from application code. By keeping these two pieces separate, I allow future changes to the UI to happen without serious impact on the core code. Callbacks are the glue between the UI and the application code because they map UI events to code that performs some action.

Callbacks have varying interfaces. Button click signals need callbacks that take the button widget ID and user data as input arguments. CList callbacks for the select-row signal, sent when a row is clicked on, get five arguments. Letting Glade generate them makes it possible to learn these varying interfaces quickly. In fact, because the API for callbacks is not well documented—at least documentation is not easy to find—letting Glade create these is the best way to learn callback syntax.

Filling in the callback code can be done directly in callbacks.c, but this C module will be dropped in the future when I move to libGlade. Instead, I usually pass parameters straight through to a similar function in utils.c that does the actual work. Despite this general rule, one important bit of code was put in callbacks.c: assigning widget IDs to global variables. Listing 1 shows how a global variable is used in a callback to save the ID of the Preferences dialog.

Listing 1. The Preferences dialog is created the first time it is requested, and the widget ID is saved in a global variable.

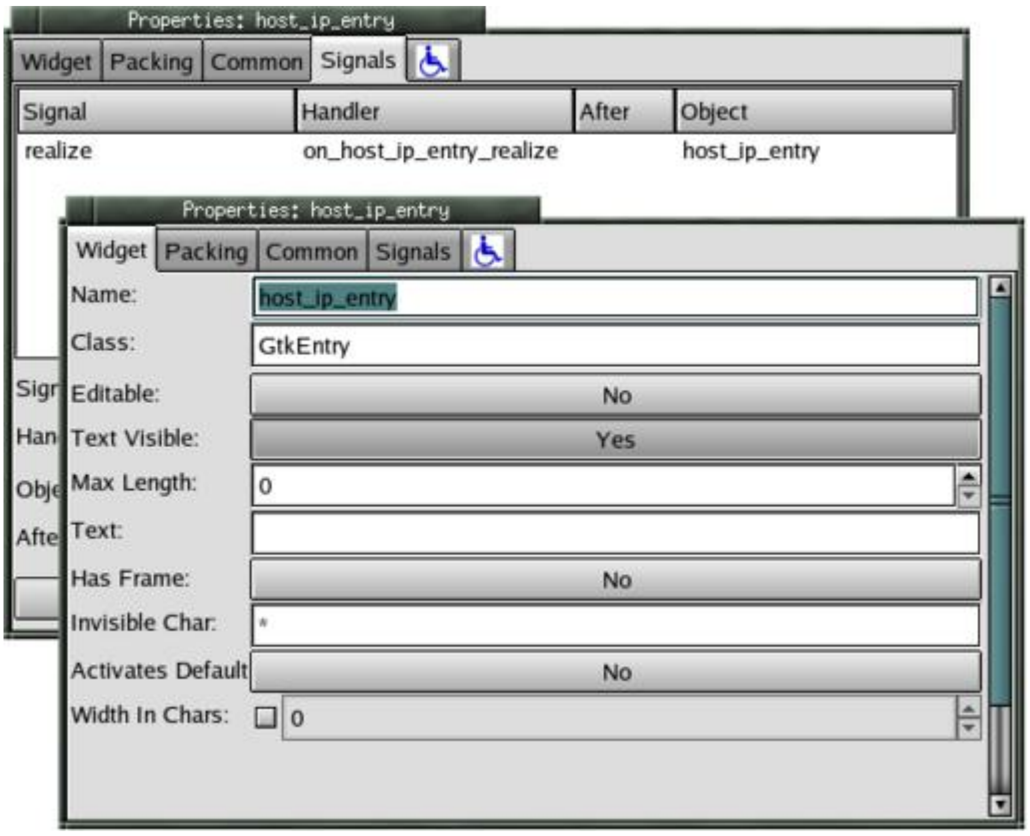
```
void
XRPreferences    (GtkButton       *button,
                  gpointer         user_data)
{

   /* If it hasn't been opened before, create
    * the dialog.
    */

   if ( XR_Preferences_Window == NULL )
   {
      XR_Preferences_Window = create_preferences();
      gtk_widget_realize(XR_Preferences_Window);
   }
   ...
}
```

Keeping track of individual widgets became necessary for multiple reasons. First, some icons change dynamically depending on varying states of the program. Second, many windows are displayed only temporarily and creating and destroying them is overkill. It's far easier to create them once and then simply hide and display them as needed. Finally, Glade-generated CLists need to be updated at runtime. The variables that hold the widget IDs are scoped only within the interface.c file, which means functions outside this Glade-generated file can't make changes easily to those widgets.

To solve this problem I set a realize signal for every widget I need access to at runtime. Glade lets you specify the name of the variable to be defined in interface.c. The callback associated with the realize signal is passed that variable value as the object parameter. In the callback, the value is saved in a global variable defined in xr.h, the single header file I created for this project. All globals are scoped using #ifdefs, with #defines specified at the top of the C module, as shown in the two code snippets of Listings 2 and 3.


Properties: host_ip_entry

Signal	Handler	After	Object
realize	on_host_ip_entry_realize		host_ip_entry

Properties: host_ip_entry

Name:	host_ip_entry
Class:	GtkEntry
Editable:	No
Text Visible:	Yes
Max Length:	0
Text:	
Has Frame:	No
Invisible Char:	*
Activates Default	No
Width In Chars:	0

Figure 8. Realize signals are used to pass the widget ID to a callback that saves the ID in a global variable.

Listing 2. Global variables and functions are declared in xr.h.

```
code:
#ifdef XR_CB_C
GtkWidget *XR_Msg_Window = NULL;
GtkWidget *XR_Msg = NULL;
void XRUMsg();
void XRUInit();
#else
extern GtkWidget *XR_Msg_Window;
extern GtkWidget *XR_Msg;
extern void XRUMsg();
extern void XRUInit();
#endif
```

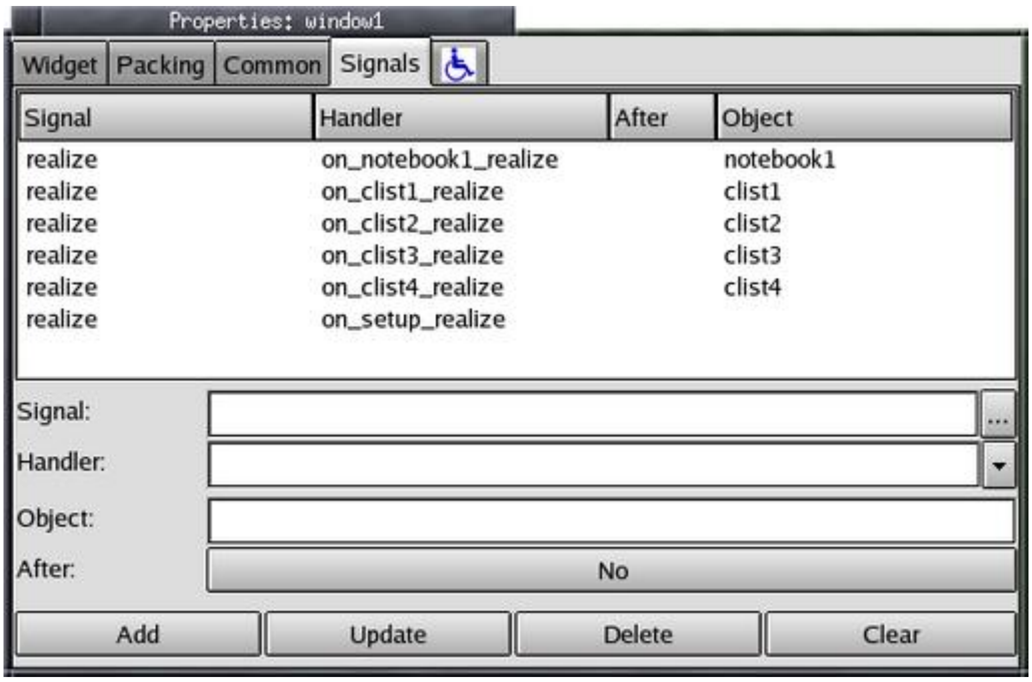
Listing 3. #defines make variables and functions properly accessible for C modules.

```
code:
#define XR_UTIL_C
#include <stdio.h>
#include <stdlib.h>
#include "xr.h"
...
```

One problem with this methodology is defining at what point a widget ID becomes available. The callback for the realize signal is called only right before the widget actually becomes visible. Sometimes you need access to that widget

ID before this happens. Fortunately, this is solved easily. The widget is created in interface.c before the signal handlers are set up. A signal handler is a function that associates a callback with some event.

Because of this, the locally named variables all have valid values by the time a realize signal is configured. It therefore becomes possible to set multiple callbacks for a single widget, all of which are set to the realize signal for that widget, which saves the widget IDs of other widgets. For example, the main window widget for Ximba Radio has realize callbacks set for it that save the widget IDs of all the predefined CList widgets in the Channel Listing window; there are four such widgets. This is required because, initially, those CLists are not visible even after the main window is made visible, and I need to start updating the lists right away. If I didn't use the main window to save the widget IDs of the CLists, I wouldn't be able to start updating them with channel information until those lists were displayed at least once.



Signal	Handler	After	Object
realize	on_notebook1_realize		notebook1
realize	on_clist1_realize		clist1
realize	on_clist2_realize		clist2
realize	on_clist3_realize		clist3
realize	on_clist4_realize		clist4
realize	on_setup_realize		

Figure 9. Right before the main window is displayed, the widget IDs of its CList subwindows are saved in global variables by multiple callbacks.

Dynamic changes to widgets also require saving the widget ID. One example of this is the state icon for Ximba Radio. To change the state icon I needed to use only GTK+ stock icons and save the widget ID of the Glade-generated GtkImage widget. When the user changes the program state—either by disconnecting from the dæmon or enabling or disabling the mute—the state icon is changed easily with a single GTK+ function call, as shown in Listing 4. The complete set of GTK+ stock icons is listed in the on-line GTK+ documentation.

Listing 4. This function changes the state icon to the connected state using what GTK+ calls the Apply icon.

```
code:
gtk_image_set_from_stock(GTK_IMAGE(XR_Status_Image),
   GTK_STOCK_APPLY,
   GTK_ICON_SIZE_BUTTON);
```

The use of global variables is not the only issue experienced developers might take with my methodology. Another is my choice of using a deprecated GTK+ widget: the CList, also known as the columned list widget. Deprecated implies that the widget, while still part of the current distribution, is no longer being developed actively and may be removed from the GTK+ distribution in the future.

The current replacement for the CList widget is the Tree and List widget. Ximba Radio requires list entries to be added and removed dynamically on a frequent basis. Neither the CList nor the Tree and List widget were helpful with this requirement. However, because the CList was designed specifically for handling lists and not expandable trees, I found it the less complex of the two choices. By definition, prototypes require getting an application up and running quickly with a standard or typical interface. CList support in Glade makes this possible without having to learn the complexities of the Tree and List widget. The trade-off will come later when I have to deal with the eventual disposition of the CList.

Ximba Radio makes heavy use of lists. All channel, category and favorites information is kept in columned lists. While the complete channel listings and the favorites are static lists that never go completely away, the category lists are dynamic. Users can enable or disable them from the display, making it easier to find stations based on their own preferences. To manage the dynamic nature of the category lists, I made use of GLib's doubly linked list, the GList. Few applications of any complexity can avoid the use of linked lists, and GLists are extremely easy to use. A one-way link list option exists, the GSList, but it costs little to have the two-way links in a GList, and reserving the option to travel in both directions in a list is worth any extra weight a GList might add.

Listing 5. These two functions write preference data to a file, traversing a GList to get category information.

```
code:
void
XRUSavePrefs()
{
   ...

   /* Write the preferences to it. */
   fprintf(fd, "hostname:%s\n", prefs.hostname);
   if ( prefs.daemondir )
      fprintf(fd, "daemondir:%s\n",prefs.daemondir);
   else
      fprintf(fd, "daemondir:\n");
   fprintf(fd, "favorites:%d\n",
      (int)prefs.enable_favorites);
   fprintf(fd, "channels:%d\n",
      (int)prefs.channel_windows);
   fprintf(fd, "performance:%d\n",
      prefs.performance);

   /* Run the list of categories and save them
    * and their states
    */

   g_list_foreach(prefs.categories,
      SavePrefsCategory, fd);

   ...
}

static void
SavePrefsCategory(
   CatEntryT   *catentry,
   FILE        *fd
)
{
   fprintf(fd, "category:%s:%d\n", catentry->name,
      catentry->state);
}
```

One other pitfall to avoid comes with testing your application if you use pixmaps. Ximba Radio uses a logo pixmap in several places. The pixmap is not found if you run the application from the default src directory unless you first do a complete build:

```
./configure --prefix=<install directory>
make
make install
```

This process copies over pixmaps to a pixmap directory under the install directory prefix. For example, if <install directory> is set to /usr/local/ximbaradio, the pixmaps are installed in /usr/local/ximbaradio/share/ximbaradio/pixmaps. Once installed, the compiled program finds the pixmaps correctly. If the pixmaps are updated, the make install step needs to be rerun. Switching to compiled logos—that is, making them part of the binary so relocation issues are not relevant—is possible with modifications to the support.c file. I've done this for previous applications, but the technique is beyond the scope of this article.

Final Analysis

I managed to get the complete application written in approximately 30 hours of total work. Most of that was spent on core code. UI code was completed in perhaps ten hours of total work. The application matches the Windows UI in all major features and the Preferences are easy to manage. The UI code is independent of the core code, although some dependency on the callbacks.c file still is present.

Development of Ximba Radio continues. Plans include adding audio control options and a GStreamer-based reflector. The reflector, Ximba Radio and OpenXM will combine to allow remote access to PC-based XM Radio satellite service.

Glade 3 is under development, and it's expected that the code generation feature will be removed. Because this upcoming release has been long in development and its release does not appear to be close to the horizon, working with generated code remains a viable option for Glade-built prototypes for the near future. That said, prototype development does remain fast and painless with GTK+ and Glade.

Michael J. Hammel is a software engineer and author living in Houston, Texas, with his wife, Brinda, and daughter, Ryann. An avid runner and tennis player, who loves his dogs Reba and Bailey as much as he loves his computer, Michael spends his spare time nursing aging knees, cleaning up torn sofa cushions and asking himself why he doesn't have any spare time. His Web site is graphics-muse.com, and he can be reached at mjhammel@graphics-muse.org.

Ten Commands Every Linux Developer Should Know

John Fusco

Issue #125, September 2004

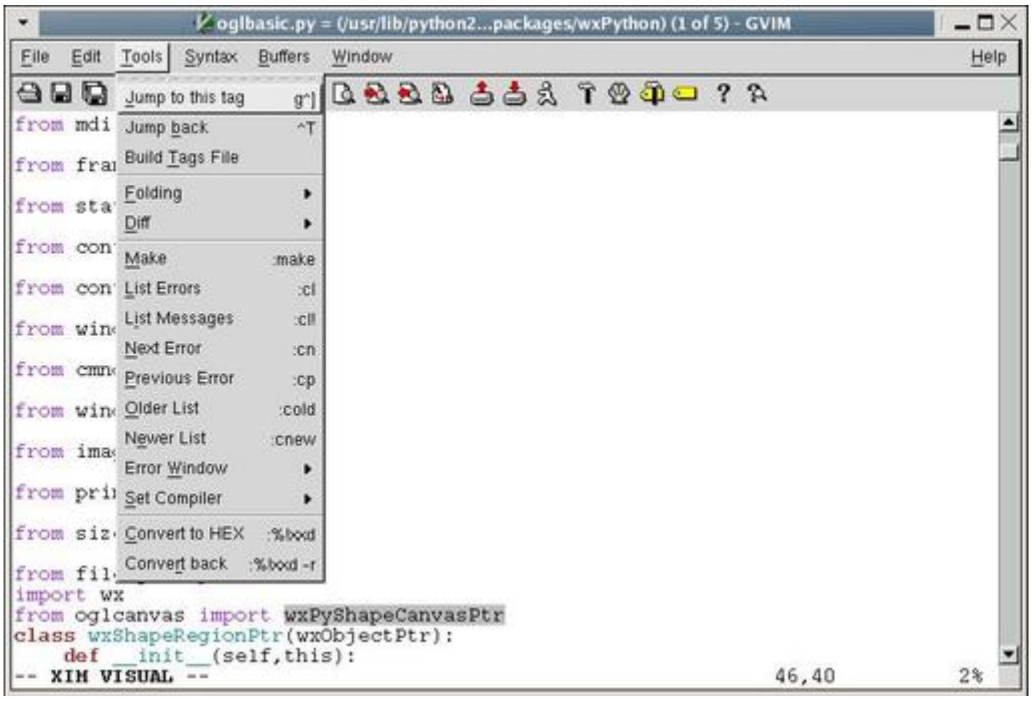
A few simple utilities can make it easier to figure out and maintain other people's code.

This article presents a list of commands you should be able to find on any Linux installation. These are tools to help you improve your code and be more productive. The list comes from my own experience as a programmer and includes tools I've come to rely on repeatedly. Some tools help create code, some help debug code and some help reverse engineer code that's been dumped in your lap.

1. ctags

Those of you addicted to integrated development environments (IDEs) probably never heard of this tool, or if you did you probably think it's obsolete. But a tags-aware editor is a productive programming tool.

Tagging your code allows editors like vi and Emacs to treat your code like hypertext (Figure 1). Each object in your code becomes hyperlinked to its definition. For example, if you are browsing code in vi and want to know where the variable foo was defined, type `:ta foo`. If your cursor is pointing to the variable, simply use Ctrl-right bracket.



```
from mdi
from fra
from sta
from con
from con
from win
from cmn
from win
from ima
from pri
from siz
from fil
import wx
from oglcanvas import wxPyShapeCanvasPtr
class wxShapeRegionPtr(wxObjectPtr):
    def __init__(self,this):
-- XIM VISUAL --                                      46,40          2%
```

Figure 1. gvim at Work with Tags

The good news for the vi-impaired is ctags is not only for C and vi anymore. The GNU version of ctags produces tags that can be used with Emacs and many other editors that recognize tag files. In addition, ctags recognizes many languages other than C and C++, including Perl and Python, and even hardware design languages, such as Verilog. It even can produce a human-readable cross-reference that can be useful for understanding code and performing metrics. Even if you're not interested in using ctags in your editor, you might want to check out the human-readable cross-reference by typing `ctags -x *.c*`.

What I like about this tool is that you get useful information whether you input one file or one hundred files, unlike many IDEs that aren't useful unless they can see your entire application. It's not a program checker, so garbage in, garbage out (GIGO) rules apply.

2. strace

strace lets you decipher what's going on when you have no debugger nor the source code. One of my pet peeves is a program that doesn't start and doesn't tell you why. Perhaps a required file is missing or has the wrong permissions. strace can tell you what the program is doing right up to the point where it exits. It can tell you what system calls the program is using and whether they pass or fail. It even can follow forks.

strace often gives me answers much more quickly than a debugger, especially if the code is unfamiliar. On occasion, I have to debug code on a live system with no debugger. A quick run with strace sometimes can avoid patching the system or littering my code with printfs. Here is a trivial example of me as an unprivileged user trying to delete a protected file:

```
strace -o strace.out rm -f /etc/yp.conf
```

The output shows where things went wrong:

```
lstat64("/etc/yp.conf", {st_mode=S_IFREG|0644,
st_size=361, ...}) = 0
access("/etc/yp.conf", W_OK) = -1 EACCES
(Permission denied)
unlink("/etc/yp.conf") = -1 EACCES (Permission
denied)
```

strace also lets you attach to processes for just-in-time debugging. Suppose a process seems to be spending a lot of time doing nothing. A quick way to find out what is going on is to type `strace -c -p mypid`. After a second or two, press Ctrl-C and you might see a dump something like this:

```
% time     seconds  usecs/call     calls    errors  syscall
------ ----------- ----------- --------- --------- ----------------
 91.31    0.480456        3457       139           poll
  6.66    0.035025         361        97           write
  0.91    0.004794          16       304           futex
  0.52    0.002741          14       203           read
  0.31    0.001652           3       533           gettimeofday
  0.26    0.001361           4       374           ioctl
  0.01    0.000075           8        10           brk
  0.01    0.000064          64         1           clone
  0.00    0.000026          26         1           stat64
  0.00    0.000007           7         1           uname
  0.00    0.000005           5         1           sched_get_priority_max
  0.00    0.000002           2         1           sched_get_priority_min
------ ----------- ----------- --------- --------- ----------------
100.00    0.526208                  1665           total
```

In this case, it's spending most of its time in the poll system call—probably waiting on a socket.

3. fuser

The name is a mnemonic for file user and tells what processes have opened a given file. It also can send a signal to all those processes for you. Suppose you want to delete a file but can't because some program has it open and won't close it. Instead of rebooting, type `fuser -k` *`myfile`*. This sends a SIGTERM to every process that has myfile opened.

Perhaps you need to kill a process that forked itself all over the place, intentionally or otherwise. An unenlightened programmer might type something like `ps | grep` *`myprogram`*. This inevitably would be followed by several cut-and-paste operations with the mouse. An easier way is to type `fuser -k` *`./myprogram`*, where *`myprogram`* is the pathname of the executable. fuser typically is located in /sbin, which generally is reserved for system administrative tools. You can add /usr/sbin and /sbin to the end of your $PATH.

4. ps

ps is used to find process status, but many people don't realize it also can be a powerful debugging tool. To get at these features, use the -o option, which lets you access many details of your processes, including CPU usage, virtual memory usage, current state and much more. Many of these options are defined in the POSIX standard, so they work across platforms.

To look at your running commands by pid and process state, type `ps -e -o pid,state,cmd`. The output looks like this:

```
4576 S /opt/OpenOffice.org1.1.0/program/soffice.bin -writer
4618 D dd if /dev/cdrom of /dev/null
4619 S bash
4645 R ps -e -o pid,state,cmd
```

Here you can see my dd command is in an uninterruptible sleep (state D). Basically, it is blocking while waiting for /dev/cdrom. My OpenOffice.org writer is sleeping (state S) while I type my example, and my ps command is running (state R).

For an idea of how a running program is performing, type:

```
ps -o start,time,etime -p mypid
```

This shows the basic output from the time command, discussed later, except you don't have to wait until your program is finished.

Most of the information that ps produces is available from the /proc filesystem, but if you are writing a script, using ps is more portable. You never know when a minor kernel rev will break all of your scripts that are mining the /proc filesystem. Use ps instead.

5. time

The time command is useful for understanding your code's performance. The most basic output consists of real, user and system time. Intuitively, real time is the amount of time between when the code started and when it exited. User time and system time are the amount of time spent executing application code versus kernel code, respectively.

Two flavors of the time command are available. The shell has a built-in version that tells you only scheduler information. A version in /usr/bin includes more information and allows you to format the output. You easily can override the built-in time command by preceding it with a backslash, as in the examples that follow.

A basic knowledge of the Linux scheduler is helpful in interpreting the output, but this tool also is helpful for learning how the scheduler works. For example, the real time of a process typically is larger than the sum of the user and system time. Time spent blocking in a system call does not count against the process, because the scheduler is free to schedule other processes during this time. The following sleep command takes one second to execute but takes no measurable system or user time:

```
\time -p sleep 1
real 1.03
user 0.00
sys 0.00
```

The next example shows how a task can spend all of its time in user space. Here, Perl calls the log() function in a loop, which requires nothing from the kernel:

```
\time perl -e 'log(2.0) foreach(0..0x100000)'
real 0.40
user 0.20
sys 0.00
```

This example shows a process using a lot of memory:

```
\time perl -e '$x = 'a' x 0x1000000'

0.06user 0.12system 0:00.22elapsed 81%CPU
(0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (309major+8235minor)pagefaults
0swaps
```

The useful information here is listed as pagefaults. Although the GNU time command advertises a lot of information, the 2.4 series of the Linux kernel stores only major and minor page-fault information. A major page fault is one that requires I/O; a minor page fault does not.

6. nm

This command allows you to retrieve information on symbol names inside an object file or executable file. By default, the output gives you a symbol name and its virtual address. What good is that? Suppose you are compiling code and the compiler complains that you have an unresolved symbol _foo. You search all of your source code and cannot find anywhere where you use this symbol. Perhaps it got pulled in from some template or a macro buried in one of the dozens of include files that compiled along with your code. The command:

```
nm -guA *.o | grep foo
```

shows all the modules that refer to foo. If you want to find out what library defines foo, simply use:

```
nm -gA /usr/lib/* | grep foo
```

The nm command also understands how to demangle C++ names, which can be handy when mixing C and C++. For example, forgetting to declare a C function with `extern"C"` produces a link time error something like this:

```
undefined reference to `cfunc(char*)'
```

In a large project with poorly defined headers, you might have a hard time tracking down the offending module. In this case, you can look for all the unresolved symbols in each object file with demangling turned on as follows:

```
nm -guC *.o
extern-c.o:cfunc
no-extern-c.o:cfunc(char*)
```

The first module is correct; the second is not.

7. strings

This command looks for ASCII strings embedded in binary files. It can be used for good or for evil. The good uses include trying to figure out what library is producing that cryptic string on stdout every once in a while, for example:

```
strings -f /usr/lib/lib* | grep "cryptic message"
```

On the evil side, the character strings can be used to probe your format strings looking for clues and vulnerabilities. This is why you should never put passwords and logins in your programs. It might be wise to examine your own programs with this tool and see what a clever programmer can see. The version of strings that comes with the GNU binutils has many useful options.

8. od, xxd

These two commands do basically the same thing, but each offers slightly different features. od is used to convert a binary file to whatever format you like. When dealing with programs that generate raw binary files, od can be indispensable. Although the name stands for octal dump, it can dump data in decimal and hexadecimal as well. od dumps integers, IEEE floats or plain bytes. When looking at multibyte integers or floats, the host byte order affects the output.

xxd also dumps binary files but does not try to interpret them as integers or floats, so the host byte order does not affect the output, which can be confusing or helpful depending on the file. Let's create a four-byte file on an Intel machine:

```
$ echo -n abcd > foo.bin
$ od -tx4 foo.bin
0000000 64636261
0000004

$ xxd -g4 foo.bin
0000000: 61626364         abcd
```

The output of od is a byte-swapped 32-bit integer, and the output of xxd is a group of four bytes in the same byte order as they appear in the file. If you're looking for the string abcd, xxd is the command for you. But, if you're looking for the 32-bit number 0x64636261, od is the right command.

xxd also knows a few cool tricks that od doesn't, including the ability to format the output in binary and to translate a binary file into a C array. Suppose you have a binary file that you want to encode inside an array in your C program. One way to do this is by creating a text file as follows:

```
$ xxd -i foo.bin

unsigned char foo_bin[] = {
  0x61, 0x62, 0x63, 0x64
};

unsigned int foo_bin_len = 4;
```

9. file

UNIX and Linux have never enforced any policy of filename extensions. Naming conventions have evolved, but they are guidelines, not policies. If you want to name your digital picture image00.exe, go ahead. Your Linux photo application gladly accepts the file no matter what the name is, although it may be hard to remember.

The file command can help when you have to retrieve a file from a brain-dead Web browser, which mangles the name—say a file that should have been named foo.bar.hello.world.tar.gz comes out as foo.bar. The file command can help like this:

```
$ file foo.bar

foo.bar: gzip compressed data,
was "foo.bar.hello.world.tar", from Unix
```

Perhaps you received a distribution with a bin directory full of dozens of files, some of which are executables and some are scripts. Suppose you want to pick out all the shell scripts. Try this:

```
$ file /usr/sbin/*  | grep script

/usr/sbin/makewhatis:  a /bin/bash script text
executable

/usr/sbin/xconv.pl:    a /usr/bin/perl script
text executable
```

The file command identifies all the files in the bin directory, and the grep command filters out everything not a script. Here are some more examples:

```
file core.4867

core.4867: ELF 32-bit LSB core file Intel 80386,
version 1 (SYSV), SVR4-style, from 'abort'

file /boot/initrd-2.4.20-6.img

/boot/initrd-2.4.20-6.img: gzip compressed data,
from Unix, max compression

file -z /boot/initrd-2.4.20-6.img

/boot/initrd-2.4.20-6.img: Linux rev 1.0 ext2
filesystem data (gzip compressed data, from Unix,
max compression)
```

Just as you shouldn't judge a book by its cover, you shouldn't assume the contents of a file based on its name.

10. objdump

This is a more advanced tool and is not for the faint of heart. It's sort of a data-mining tool for object files. A treasure trove of information is encoded inside your object code, and this tool lets you see it. One useful thing this tool can do is dump assembly code mixed with source lines, something `gcc -S` doesn't do for some reason. Your object code must be compiled with debug (-g) for this to work:

```
objdump --demangle --source myobject.o
```

objdump also can help extract binary data from a core file for postmortem debug when you don't have access to a debugger. A complete example is too long for this article, but you need the virtual address from `nm` or `obdump -t`. Then, you can dump the file offsets for each virtual address with `objdump -x`. Finally, objdump is able to read from non-ELF file formats that gdb and other tools can't touch.

This article is not intended as a definitive reference but as a starting point to help you become more productive. Each one of these commands is well documented in the Linux man and info pages. Consult them for more information and more ideas.

Resources for this article: /article/7658.

John Fusco is a software developer with General Electric Healthcare (formerly GE Medical Systems), where he designs Linux software and device drivers for GE's Lightspeed series of Computed Tomography scanners (www.gemedicalsystems.com/rad/ct/products/light_series/index.html).

Archive Index Issue Table of Contents

Advanced search

LDAP Account Manager

John H. Terpstra

Issue #125, September 2004

If you want to give your company's Microsoft admins the ability to do routine user management on the Samba server, try this well designed Web-based tool.

The LDAP Account Manager (LAM) is an application suite for managing POSIX accounts as well as Samba SAM accounts for users, groups and Microsoft Windows machines. LAM can be used with any Web server that has PHP4 support. It connects to the LDAP server using either unencrypted connections or SSL.

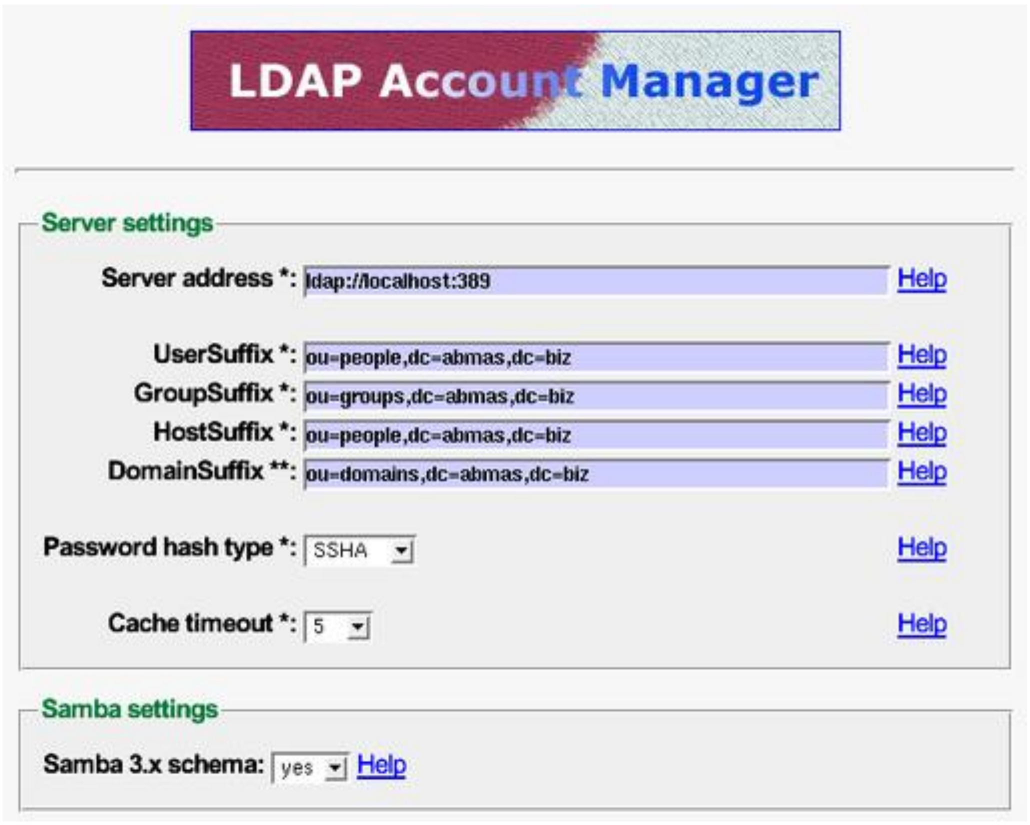
LAM is written in PHP and is available from the LAM home page, sourceforge.net/projects/lam, under the GNU GPL. The default password is lam. You should use only an SSL connection to your Web server for all remote operations involving LAM. If you want secure connections, you must configure your Apache Web server to permit connections to LAM using only SSL.

LAM requirements are as follows:

- A Web server that works with PHP4.
- PHP4 (available from the PHP home page, www.php.net).
- OpenLDAP 2.0 or later.
- A Web browser that supports CSS.
- Perl.
- The gettext package.
- mcrypt+mhash.
- SSL support—not necessary, but good to have.

Installation instructions are provided in the distribution tarball and are easy to follow. When you have installed LAM, start your Web server, and then, using your Web browser, connect to the LAM URL. Click the Configuration Login link and then the Configuration Wizard link to begin executing the default profile.

Your LDAP server needs to be running at the time LAM is configured. This permits you to validate correct operations.



Server settings

Server address *:	ldap://localhost:389	Help
UserSuffix *:	ou=people,dc=abmas,dc=biz	Help
GroupSuffix *:	ou=groups,dc=abmas,dc=biz	Help
HostSuffix *:	ou=people,dc=abmas,dc=biz	Help
DomainSuffix **:	ou=domains,dc=abmas,dc=biz	Help
Password hash type *:	SSHA	Help
Cache timeout *:	5	Help

Samba settings

Samba 3.x schema: yes Help

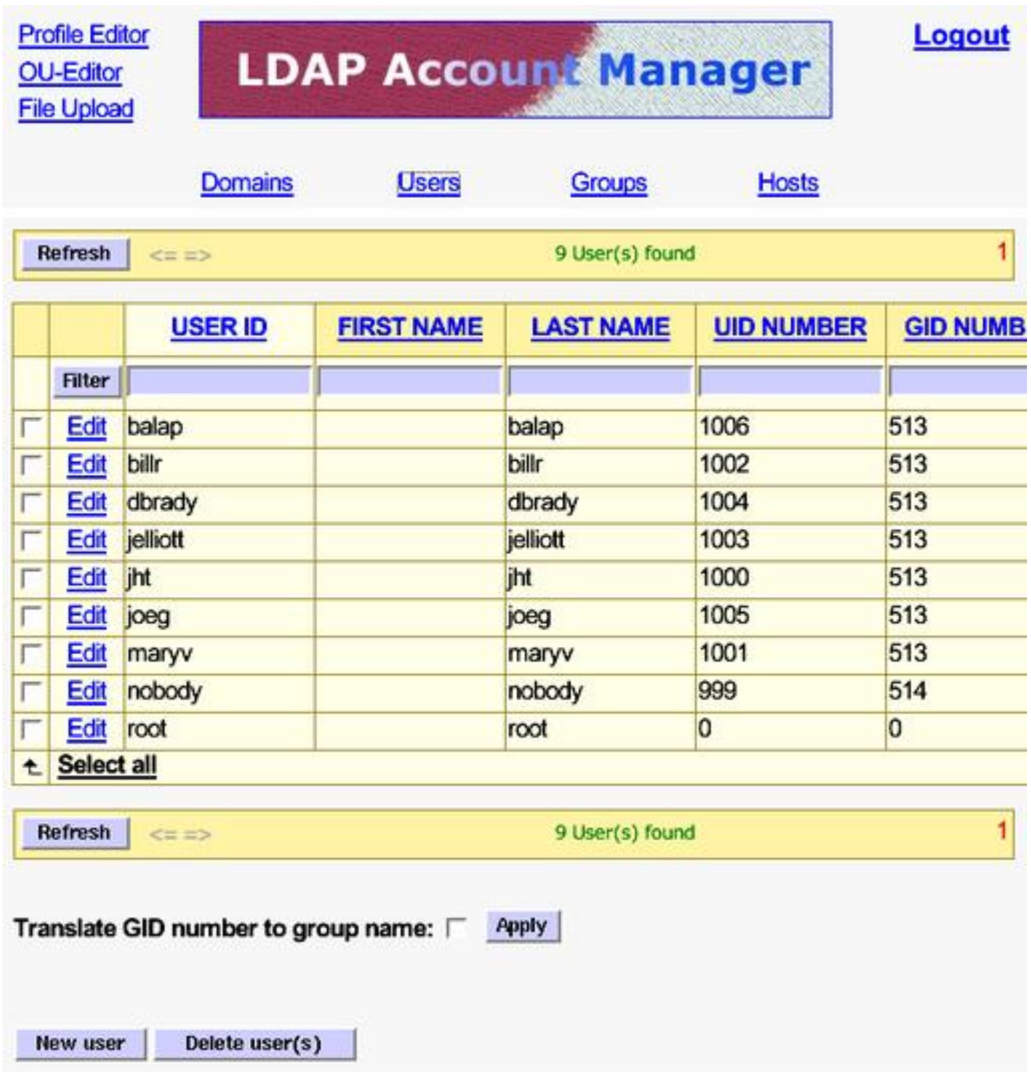
Figure 1. Part of the LAM Configuration Page

Alternately, copy the lam.conf_sample file in the config directory to lam.conf. Then, using your favorite editor, change the settings to match local site needs. The comments and help information provided in the profile file the wizard creates are useful and can help many administrators avoid pitfalls.

The LAM configuration editor has a number of options that must be managed correctly (Figure 1), such as setting the minimum and maximum UID/GID values permitted for use on your site. The default values may not be compatible with a need to modify initial default account values for well-known Windows network users and groups. The best work-around is to set the minimum values to zero (0) temporarily to permit the initial settings to be made. Do not forget to reset these to sensible values before using LAM to add additional users and groups.

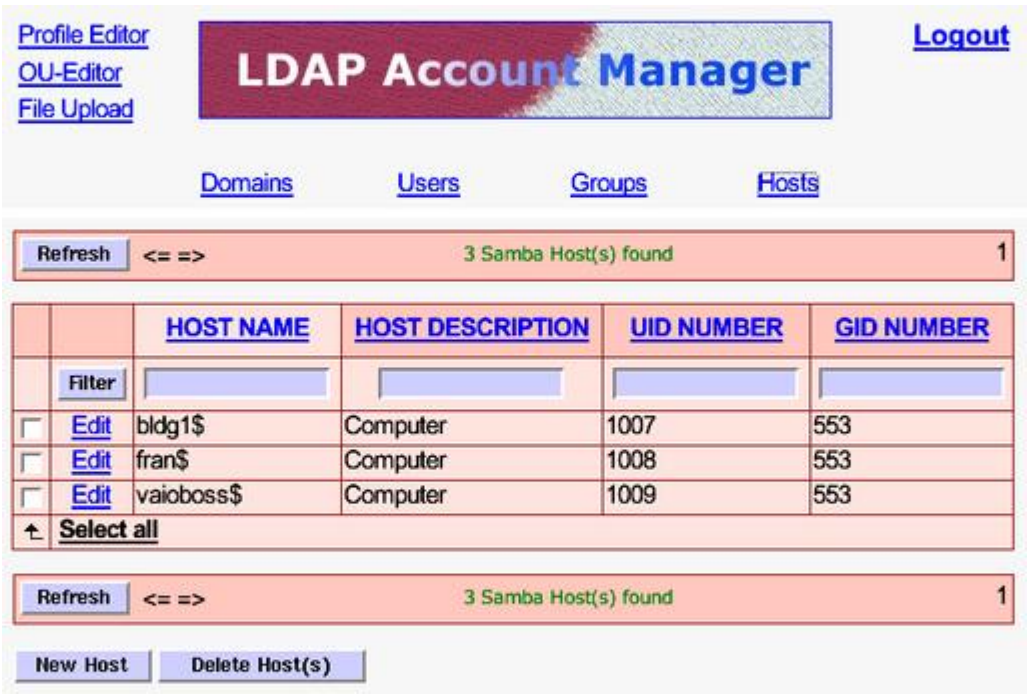
LAM is not without its oddities. For example, one unexpected feature present on most application screens permits the generation of a PDF file that summarizes configuration information. This is a well-thought-out facility.

When you log in to LAM, the opening screen drops you into the user manager (Figure 2), a logical action that permits the most common facility to be used immediately. The process of editing an existing user, as well as adding a new user, is easy to follow and clearly expressed in both layout and intent. It is a simple matter to edit generic settings, UNIX standard parameters and then Samba account requirements. Each step involves clicking a button that drives you through the process. When you have finished editing, simply click the Final button.



		USER ID	FIRST NAME	LAST NAME	UID NUMBER	GID NUMB
	Filter					
☐	Edit	balap		balap	1006	513
☐	Edit	billr		billr	1002	513
☐	Edit	dbrady		dbrady	1004	513
☐	Edit	jelliott		jelliott	1003	513
☐	Edit	jht		jht	1000	513
☐	Edit	joeg		joeg	1005	513
☐	Edit	maryv		maryv	1001	513
☐	Edit	nobody		nobody	999	514
☐	Edit	root		root	0	0

Figure 2. LAM opens at the most commonly used task, the user manager.



		HOST NAME	HOST DESCRIPTION	UID NUMBER	GID NUMBER
	Filter				
	Edit	bldg1$	Computer	1007	553
	Edit	fran$	Computer	1008	553
	Edit	vaioboss$	Computer	1009	553

Figure 3. LAM makes it easy to manage Windows domain members.

As with the edit screen for user accounts, group accounts can be dealt with rapidly. Host accounts are managed automatically using the smbldap-tools scripts. This means the Hosts edit screen (Figure 3) is not used in most cases.

One aspect of LAM that might annoy users is the way it forces conventions on the administrator. For example, LAM does not permit the creation of Windows user and group accounts that contain uppercase characters or spaces, even though the underlying operating system may have no problems with them. Given the propensity for using uppercase characters and spaces (particularly in the default Windows account names), this lack may cause some annoyance. For the rest, LAM is a useful administrative tool.

John H. Terpstra is CTO of PrimaStasys, Inc., a company that mentors organizations in alternative information technology choice evaluation and facilitates profitable change in practices. He is a long-term member of the Samba-Team, a member of the Open Source Software Institute Advisory Board and author of *The Official Samba-3 HOWTO and Reference Guide* and *Samba-3 by Example.*

Archive Index Issue Table of Contents

Advanced search

Space-Time Processing—Linux Style

Ian McLoughlin

Tom Scott

Issue #125, September 2004

Developing a new generation of wireless communications means you need FPGA development tools, a cluster for simulation and an embedded OS for prototype devices. Linux fits the need.

Here at Tait Electronics Group Research in Christchurch, New Zealand, we quietly have been building an advanced wireless networking concept called space-time (ST) processing. ST is predicted by many to drive the next big-step improvement in wireless technology after 3G. The nature of space-time is that the mathematics behind it and, consequently, its implementation, are extremely complex. Many academic researchers are working on this, but we have completed two world-first practical implementations on our ST array research (STAR) platform. These are time-reversal space-time block coding (TR-STBC) and the tongue-twisting single carrier, adaptive multivariate decision feedback equalised multiple-in, multiple-out (SC-AMV-DFE-MIMO) coding scheme. Without a doubt, these achievements were possible largely thanks to the performance and adaptability of Linux.

This article describes how Linux was key in enabling our mathematical simulations and was adopted for our embedded processing and runtime control. We cover such diverse aspects as message passing interface (MPI), cluster computing, embedded Linux, PHP, shell scripts, network filesystem (NFS), SMB (server message block) and kernel modules on ARM, Alpha and Intel architecture computers. We skip development details, but describe the system as it operates today, highlighting some of the real advantages we gained by using Linux.

In its basic configuration, each STAR platform consists of one multichannel transmitter unit and one multichannel receiver unit, both connected to a shared LAN and transmitting anything up to around 200Mb/s at microwave frequencies. There actually are 23 printed circuit boards in each unit, which took 15 people a year to design and build.

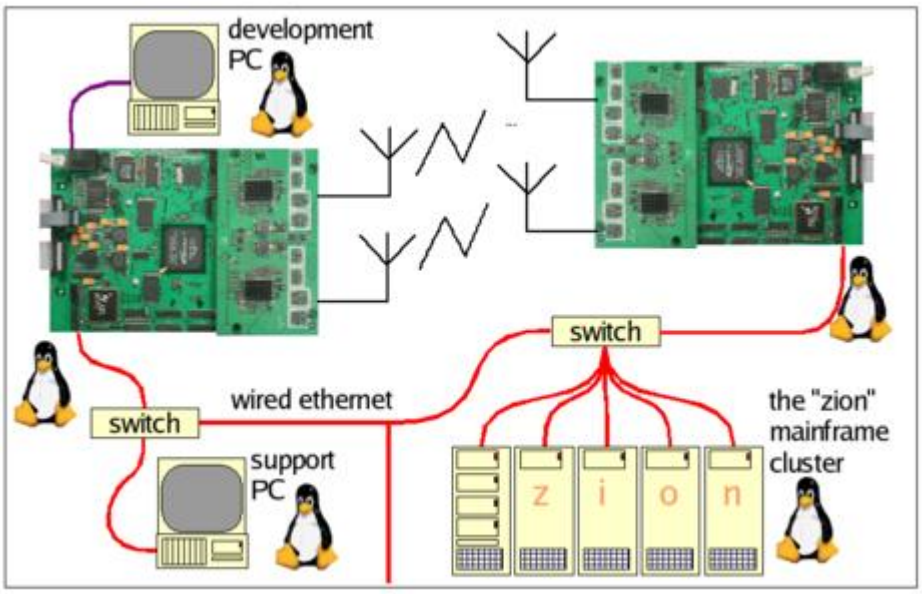


Figure 1. A bidirectional RF link uses one multichannel transmitter and one multichannel receiver on each STAR platform.

Figure 1 shows the system wired to a shared Ethernet as a bidirectional RF link. The setup shown is for experimental purposes; an actual product would not have a shared Ethernet between transmitter and receiver.

All of the clever radio processing is done on the digital board, not by the ARM processor but by a dedicated field programmable gate array (FPGA) and digital signal processor (DSP). We call FPGA code firmware because we can't decide whether it's software or hardware; this way we don't have to comply with either the software coding standards or the hardware design standards of the company. In the middle of last year, when we designed the system, we used the biggest, fastest and most expensive FPGA and DSP we could obtain, and we since have added two more big FPGAs. The ARM is not used for low-level processing because we require over 50,000MIPS for ST processing. With this complexity, even the fastest components alone are not enough, so we decided early on to build a multiprocessor-capable system.

STAR units are extensible through high-speed low-voltage differential signaling (LVDS) connections that run at up to 1Gb/s. Each board has two sets of five-channel bidirectional LVDS interconnects to link two neighbouring boards. At the same time, each board is wired to Ethernet. High-speed data travels over LVDS, and control data uses Ethernet.

Development

Most processing occurs in the FPGA, coded in VHDL. A growing number of VHDL tools are available for Linux (see sidebar), and we used Quartus from Altera. In our system, we developed algorithms first with GNU-Octave and then ported them to VHDL. Octave and the mostly compatible MATLAB are available for Linux and Microsoft Windows, but Octave has an MPI-enabled version that runs on clusters. We compiled this for a cluster of old DEC Alphas we call zion. MPI-Octave really screams on zion despite the fact that the fastest CPU is only 500MHz.

For our digital board, we used an ARM Linux toolchain from handhelds.org to compile freshly patched 2.4.18 kernel sources. Russell King began ARM Linux when working on a distribution for Acorn computers in the early 1990s. Now, ARM is one of the best-supported processors under Linux and is a real joy to us. Three days was enough to port Linux to our custom hardware, although the Ethernet driver took a couple more days to complete. ARM is easily the world's number one processor by sales, with a huge amount of support for running Linux. This led Tait Electronics to commit to ARM processors in our future processor road map.

With ARM Linux booting, it was time for a RAM disk. Having 16MB of SDRAM and 2MB of Flash memory on the ARM, we opted for a maximum compressed kernel and RAM disk size of 1,024K each, although the uncompressed RAM disk is 4MB in size. Both are stored in Flash and allow booting without external interaction.

We opted to use BusyBox for common tools, including ls, cd, mount, insmod and ping, and TinyLogin for login and password support. Other onboard utilities handle memory-mapped peripherals and Flash, and netkit-base provides a telnet dæmon that uses TinyLogin functions. Tait Electronics bought a range of MAC addresses from the IEEE to support its Linux-on-ARM development programme, and the MAC address for each board is held in Flash memory.

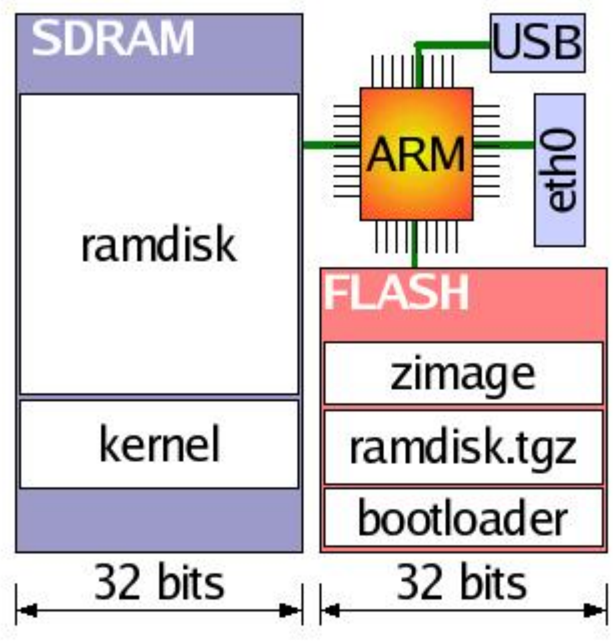


Figure 2. Each unit has both a Flash-based and a RAM-based filesystem.

We developed many small applications and even got the Abyss Web server running, but these all can't fit into Flash. We therefore NFS-mount a directory from the zion mainframe to each ARM, accessing many gigabytes of extra space. Figure 2 shows the filesystem arrangement.

We also set up SMB mounts to users' shared drives from zion, accessible to the ARM boards over NFS. If a Web server is run on the ARM boards, we end up with a highly interlinked system. Windows users can access part of their own local drivespace through the Web served by way of an embedded ARM board linked over NFS to a remote cluster server.

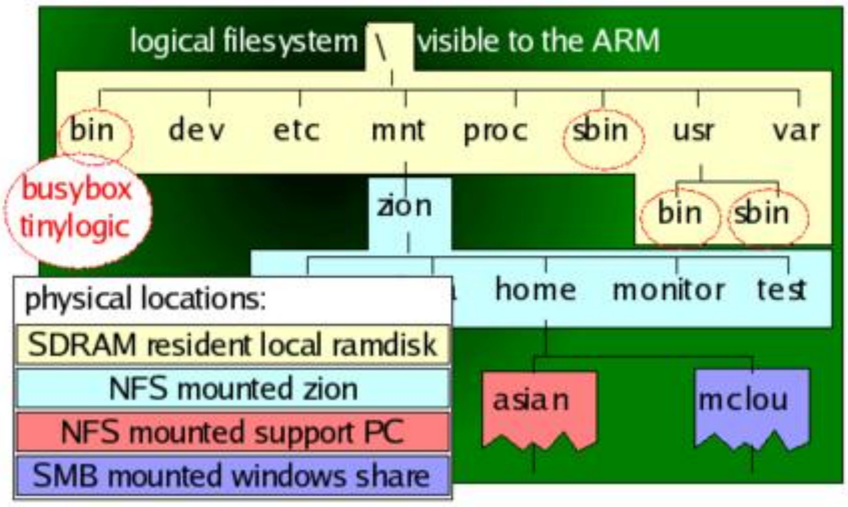


Figure 3. The Filesystem Tree as Seen from the Unit

Zion

In late 2001, we obtained the old DEC Alphas (described as boat anchors) and decided to see what we could do with them. First, we modified the built-in bootloader to get Red Hat 7.1 running. We made two major changes. First, we chose one master machine and loaded up its SCSI interface with six hard disks and a DVD-ROM. Five disks became a level-5 RAID array (using raidtools) to store simulation or experimental data, and the remaining disk is for booting and recovery.

The second change was to install the MPI on all machines. Although installation was fairly easy, the need to derive all IP addresses through DHCP caused problems. In the end, we negotiated a fixed IP address for the master machine, now called zion. We run startup scripts on other machines that log their IP addresses onto an SMB mount using RPC (Listing 1).

Listing 1. Script to store a cluster node's IP address on an SMB share.

```
#!/bin/sh
#
# Write my IP to an SMB share
#
#Mount central SMB share
smbmount //foo/bar /mnt/bar \
  -o username=me,password=mine >& \
  /dev/null#Write IP
#Grab IP address from ifconfig
address/sbin/ifconfig | grep Bcast \
  | sed 's/^.*addr://;s/Bcast.*//' > \
  /mnt/bar/$HOSTNAME.ip
```

Once MPI was working, we downloaded the latest Octave source and patched it with Octave-MPI. Since then, Octave-MPI seems to have been taken over by

Transient Research (see the on-line Resources section). We set up our MPI system with a script that gathers the IP addresses stored by the script above, pings them and builds an rhosts file. Once dynamic generation of rhosts is complete, we simply run Octave-MPI over 4+1 machines with:

```
recon -v
lamboot
mpirun -v -c4 octave-mpi
```

With the system running, the Octave processing load can be shared across the cluster. We find that Alphas have significantly faster floating-point performance than do Pentiums, but using Ethernet to pass MPI messages slows the cluster down. We haven't benchmarked the system, but a simulation that takes a couple of hours to complete on a 2GHz Compaq PC runs about 10% faster on our first cluster of four Alphas (300MHz–500MHz).

We found GNU-Octave to be an excellent tool for numerical simulations. When executed with the -traditional option, also known as -braindead, it runs most MATLAB scripts. In some cases, Octave provides better features than MATLAB does, although MATLAB has a better plotting capability than the default gnuplot engine Octave uses.

Some engineers prefer to develop on Windows, so we provide a Web interface to MPI-Octave. It uses a JavaScript telnet client served from Apache on zion and some back-end scripting. Scripts were adapted from an on-line MUD game engine. For Windows users, the telnet client script automatically mounts their shared Windows drives on the zion filesystem, runs Octave over telnet and sets up plotting capabilities so that plots are written to a directory on the Web server in PNG format for displaying with a browser. Another option lets users save plots directly to their shared drives.

For debugging, a large debug buffer in the FPGA is accessible to the ARM. We memory-mapped the FPGA with 32-bit high-speed asynchronous access to the ARM, with access in userland or kernel space. Unsurprisingly, in kernel space we use a character driver module accessed as a file. This bursts up to 1,024 data words at high speed and handles all signaling, although the sustained speed is not so good. Userland access is accomplished through the neat method of mmapping to the /dev/mem interface, as long as you remember to create /dev/mem on your embedded filesystem first (Listing 2).

Listing 2. writeport.c: a simple program that writes a 32-bit integer to a physical memory location.

```
#include <stdio.h>
#include <fcntl.h>      //needed for O_RDWR and O_SYNC
#include <sys/mman.h>   //needed for PROT_READ etc.
```

```
#define GRAB_SIZE 1024UL
#define GRAB_MASK (GRAB_SIZE - 1)

int main(int argc, char **argv)
{
   void *grab_base, *virt_addr;
   unsigned int md, read_result, writeval;
   off_t phys_addr = strtoul(argv[1], 0, 0);
   /*open memory interface*/
   if((md = open("/dev/mem", O_RDWR | O_SYNC)) == -1)
   {
      printf("ERR - /dev/mem open failed\n");
      exit(1);
   }
   /* Map one page to the physical address given*/
   if(grab_base = mmap(0, GRAB_SIZE, PROT_READ |
     PROT_WRITE, MAP_SHARED, md, phys_addr &
     ~GRAB_MASK), grab_base == (void *) -1)
   {
      printf("ERROR: failed to map\n");
      exit(1);
   }
   /*write to virtual memory that is now mapped to
   the requested physical address*/
   *((unsigned long *)grab_base + (phys_addr &
     GRAB_MASK)) = strtoul(argv[2], 0, 0);
   /*close memory interface*/
   if(munmap(grab_base, GRAB_SIZE) == -1)
   {
      printf("ERR - unmap failed\n");
      exit(1);
   }
   close(md);
}
```

These tools allow us to upload known test vectors, saved from Octave on zion, to the debug buffer. Under ARM control, we route the debug buffer output to the input of a block under test, run the system for a few clock cycles and capture the output back in the debug buffer. Analysing the result in Octave tells us if the block works.

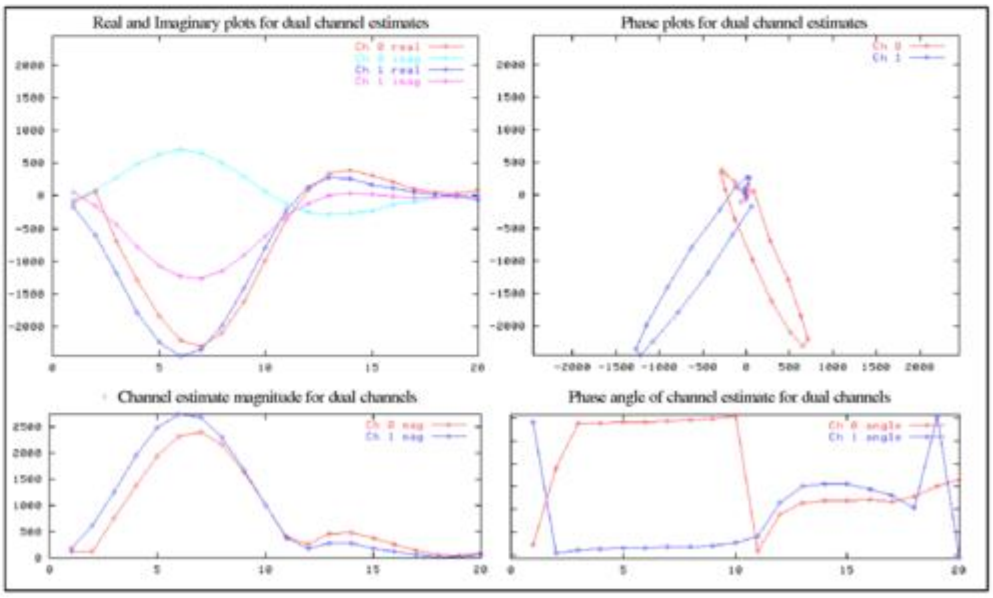


Figure 4. Visualising Channel Modes—The Complex Paths from Transmitter to Receiver

We found visualisation to be an important factor. After our first milestone, we invited some people to see the system. They saw boxes humming away with a couple of green LEDs to indicate everything was working. We noted a distinct lack of enthusiasm at what we believe was a world-first demonstration of the technology, so we realised something more was required. To this end, we chose to display channel models as they adapted in near real time. A channel is the complex path from a transmitter to a receiver, including reflections, multiple paths, dispersion and so on. The system we built sent training symbols over the air to sound the channel before sending data. Sounding gives us a picture of the channel, which we decided to display. We used the FPGA debug buffer on the receiver, with an ARM script running periodically to execute a program to extract the channel data from the buffer, format it and save in an Octave-compatible .mat file on zion. Octave was run non-interactively on zion to read the channel data periodically, analyse it and generate four plots as PNG image files, which a zion Web server PHP page displayed as a visualisation updated every four seconds (Figure 4).

Support

Our Linux-powered digital boards normally run complex processing algorithms and communicate over air and Ethernet to other systems. Even under laboratory conditions, it is difficult to know whether all components are operating correctly, so we decided to utilise the power of Linux to implement self-test and monitoring solutions. Like our visualisations, this ties many components together in a flexible way.

Nontechnical users demanded a graphical interface, which we could have created using GTK, Tk, Qt or something similar, but we decided instead on PHP-powered Web scripting. This allows users of all platforms to access the system. The C-like language syntax makes it easy and quick to use for C programmers, and the familiarity of modern users with Web interfaces helps. The source code for most applications is less than 50KB in size, which is a major advantage in debugging and testing and consequently improves our confidence in that code.

Unfortunately, such a user interface has a couple of disadvantages: first, when a background event must be brought to the attention of a user and, second, when the system must interact directly with hardware. The first problem can be solved through the use of ticker-type messages in a separate frame. In our system, we solved the second problem through the use of small low-level C programs with a file-based interface to PHP.

In truth, our system is quite convoluted. The controller in each unit is the ARM processor, but the controller for all controllers in a multi-unit system is the Web server serving the PHP scripts. For self-tests, the system implements a repetitive monitor script running on each ARM every five seconds and interrogates the system for the Web server. Details are written to a file named after the IP address of the Ethernet on each ARM. The monitor script also is responsible for ensuring that each operational system is synchronised in time to zion. We didn't use NTP due to code size and because we require sufficient time resolution only to prevent shared filesystem time consistency errors. We script zion to write the current time and date, using the date command, to the file every five seconds, and each ARM reads that file every five seconds to set its time. This gets around time synchronisation problems and fixes timestamps on files. We also use it as a watchdog to reset boards more than a minute out of date.

Apart from self-tests, the script also queries version numbers of RAM disk and kernel on startup and writes this to a status file. A final use of the script is to execute board-specific instructions. These are written by the PHP control Web page to tell each ARM board what it should be doing. Instructions from PHP are written as a shell script to be executed by a single board, identified by IP address.

All units run identical kernels and RAM disks stored locally in Flash memory and verified against master copies on zion at startup. Any inconsistency causes the correct kernel or RAM disk to be burnt into the local Flash memory. We built custom Flash memory tools to do this. In practice, we have had no storage errors, but we use this to roll out new versions of kernel and RAM disk without user intervention.

The PHP Web pages refresh every five seconds using the autoload HTML tag. Of course, this works only if at least one user has a browser currently viewing the page. If not, the information probably isn't required anyway. A manager's-eye-view Web page hides almost all useful information but looks great. Real users can click down through the levels to get to individual scripts that customise each board.

Debugging and Monitoring

Linux software on the ARM sets up FPGA firmware to stream small packets of data from the wireless system to a debug buffer in the FPGA. In slower time, the ARM extracts this and stores it on zion. These files can be analysed automatically to look for faults (like all 0s) and if found, the user is alerted through the Web pages.

Conclusion

We recently have started to think about products rather than pure research. It is likely that we will communicate IP packets and some form of embedded Linux will power the product. This will happen not only because we relied upon it during development, but there are few viable options for handling IP packets. WinCE is slow and bloated, and VxWorks is lean but costly and lacks in protocol support.

Tait Electronics is a nonprofit electronics trust run to benefit the employees and society of Christchurch, New Zealand. It was founded by Sir Angus Tait more than 30 years ago. It exports 97% of its mobile radio products to more than 200 countries worldwide.

Linux Tools for VHDL

Simulation

Simulation tools let you write VHDL, simulate and debug it, but you need a synthesis tool if you want to run your VHDL in hardware. Many simulation tools have graphical waveform displays. Most professional VHDL tools will run on Linux. Free tools include FreeHDL and GHDL, a GCC front end for VHDL.

Synthesis

Synthesis takes working VHDL and maps it to a hardware device, such as an FPGA or MPGA (mask programmed gate array). This is device-specific, so free tools are rare. Big manufacturers support Linux with proprietary tools. Altera Quartus is available for Linux but not in the Web edition. Xilinx ISE also is available for Linux but not in the Web pack edition. Pages on-line can tell you how to get Quartus and ISE running on Linux (see Resources).

Resources for this article: /article/7648.

Ian McLoughlin has used Linux for about 12 years and has a signal processing background. Before emigrating to New Zealand, he was a university lecturer in Singapore and still travels there frequently as a Visiting Scientist to the XSat Satellite Programme (due for launch in 2006). He is married with two young Linux-using kids.

Tom Scott is a director of Mission Technologies Ltd., (www.missiontech.co.nz) known for his terse, straightforward, no-nonsense, hands-on approach. Father of two, husband of one and a wanna-be missionary.

At the Forge

WordPress

Reuven M. Lerner

Issue #125, September 2004

If you want a Weblog package with minimal command-line setup and a full-featured Web-based administration system, WordPress may be for you.

A fitting end to this series on Weblog software is a brief introduction to WordPress, which offers a wide array of features and continues to grow in popularity and sophistication. WordPress is Weblog software with an extremely clean, easy-to-use interface written in PHP with a MySQL back end.

Installing WordPress

Installing WordPress is amazingly straightforward if you have a bit of experience with Apache and MySQL. WordPress stores all of its information in a set of MySQL tables, which means you need to create at least one database before installing WordPress. WordPress allows you to have more than one Weblog inside of a single database, which is useful if your site is hosted by a company that gives you only a single MySQL database.

To create a new MySQL database, you need to use the mysqladmin program, which is in /usr/local/mysql/bin on my system:

```
# /usr/local/mysql/bin/mysqladmin -p create wordpress
```

The above command assumes you are running as root and that the root user has administrative privileges. Once you have created the database, you need to grant permissions for the WordPress user on these tables; we do this by logging in to the database:

```
# /usr/local/mysql/bin/mysql -p -u root
```

Once you have logged in, you can grant permissions to the WordPress user, which I called wpuser, with:

```
GRANT ALL PRIVILEGES ON wordpress.*
    TO wpuser@localhost IDENTIFIED BY 'wppass';
GRANT ALL PRIVILEGES ON wordpress.*
    TO wpuser IDENTIFIED BY 'wppass';
```

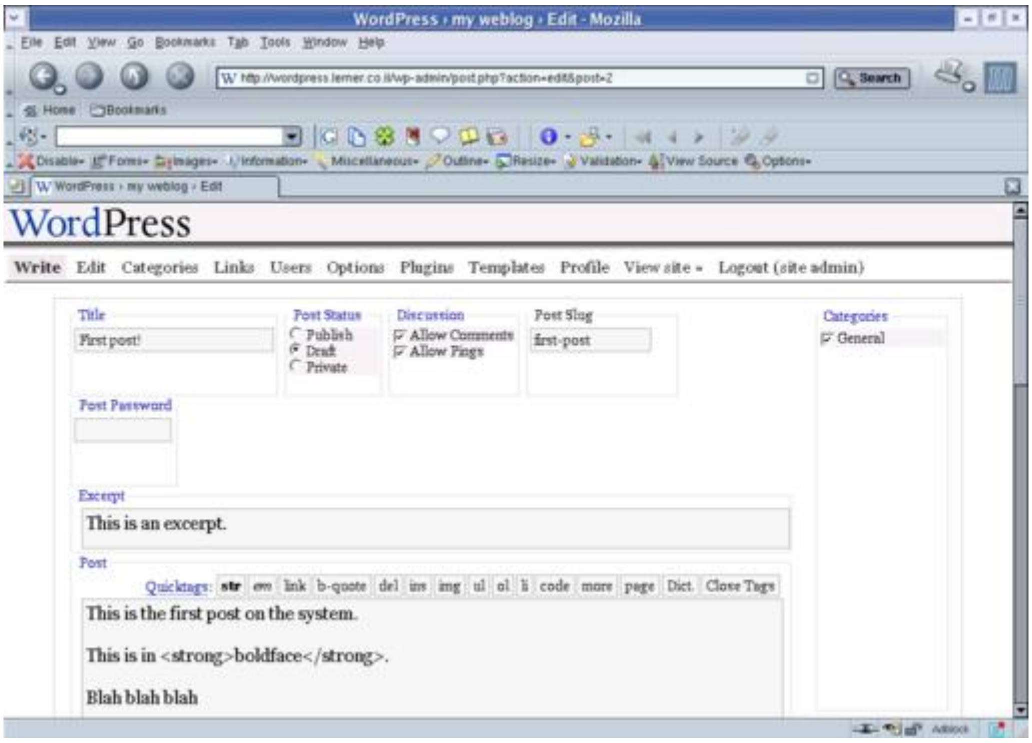
Next, download the source code to WordPress (see the on-line Resources section) and open that .tar.gz file inside of your Apache document root directory. You probably want to put the WordPress files inside of their own directory or perhaps under a virtual host, but all of the files must be within the document root if they are to be of any use.

Now point your browser to wp-admin/install, and you're on your way. The installation screens check that everything is installed correctly and ask you to answer several questions. Typically, you need to click on a link at the bottom of each page to continue with and finish the installation.

If you fail to create the database first, WordPress tells you to do this, reminding you that the database needs to be created before it can install the tables. You can use the WordPress installer only a single time. If you try to run the installer on an already-installed system, you are told to remove the old installation first. Attention to these sorts of details is nice to see in a program aimed clearly at a relatively nontechnical audience.

Using WordPress

Once you have installed WordPress, you can log in as the admin user with a randomly generated password. Logging in as the administrator allows you to add, edit and delete postings, as well as configure the system for other users. WordPress allows you to create any number of users, each of whom is assigned a privilege level between 1 and 10. The administrator, with a privilege level of 10, is allowed to do anything to the system; other users similarly can be given free rein by being assigned 10s.


WordPress › my weblog › Edit - Mozilla

http://wordpress.lerner.co.il/wp-admin/post.php?action=edit&post=2

WordPress

Write Edit Categories Links Users Options Plugins Templates Profile View site » Logout (site admin)

Title: First post!

Post Status: ( ) Publish (•) Draft ( ) Private

Discussion: [✓] Allow Comments [✓] Allow Pings

Post Slug: first-post

Categories: [✓] General

Post Password:

Excerpt: This is an excerpt.

Post

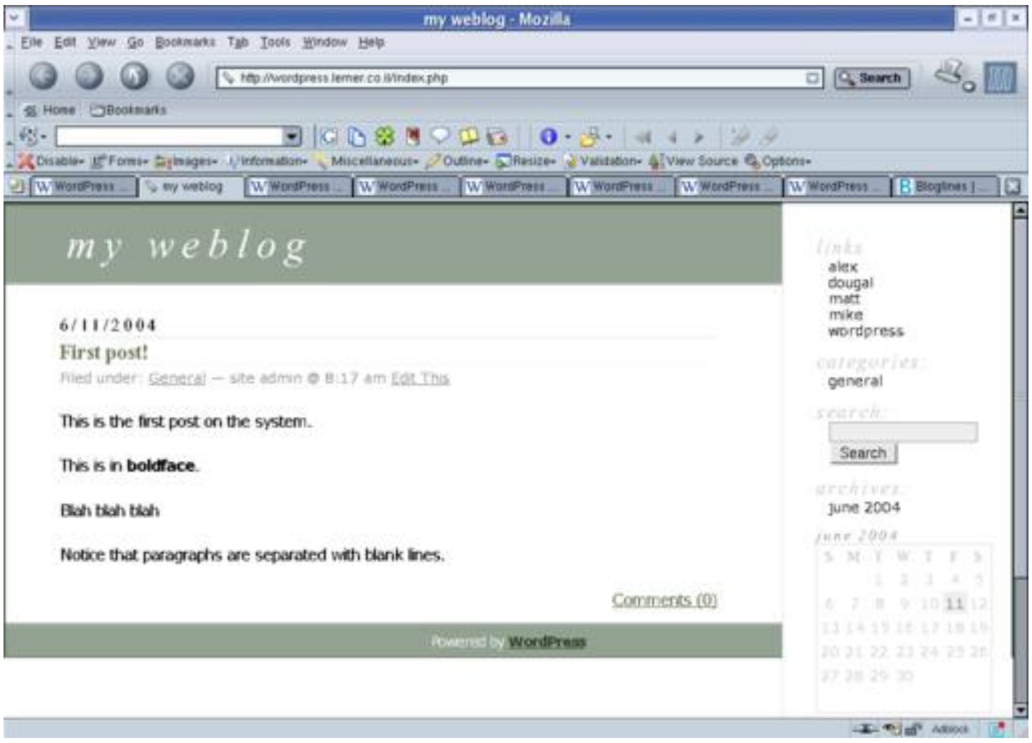
Quicktags: str em link b-quote del ins img ul ol li code more page Dict. Close Tags

This is the first post on the system.

This is in <strong>boldface</strong>.

Blah blah blah

Figure 1. Entering a Weblog Posting with WordPress


my weblog - Mozilla

http://wordpress.lerner.co.il/index.php

my weblog

6/11/2004

First post!

Filed under: General — site admin @ 8:17 am Edit This

This is the first post on the system.

This is in **boldface**.

Blah blah blah

Notice that paragraphs are separated with blank lines.

Comments (0)

Powered by WordPress

links: alex, dougal, matt, mike, wordpress

categories: general

search: Search

archives: june 2004

june 2004

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

Figure 2. Display of That Posting with the Default WordPress Template

But, of course, you don't want to give every user the equivalent of root access. The default WordPress installation raises the bar on a number of features, so you can assign privilege levels to different users. Thus, users with a level of 5 and above can manage the list of links that appears on the right side of the page, and they also can upload images. These levels can be changed by using the administrative interface.

WordPress administration is entirely Web-based; once you have created the database tables, you can manipulate the entire system by using a Web browser. A number of menus are available when you are inside the administrative interface, and some of these menus have submenus as well. Although I originally was confused by the placement of certain features, I soon began to understand the layout of the system and was able to locate and modify many of the different options.

For example, I decided that it would be nice to include my blogroll, the list of Weblogs I read on a regular basis. I use Bloglines.com, a Web-based aggregator, and was able to generate a list of Weblogs in OPML (Outline Processor Markup Language), an XML application that is the standard for such data. Importing the list into WordPress was a snap by using the links menu and choosing blogroll. One of the listed options was to import an OPML file; as soon as I did that, my list of blogs was visible to the world.

User Interface

Open-source applications often are criticized for their lack of a friendly user interface. This is largely because open-source programmers are writing the software for themselves and their colleagues, which means that anything other than core functionality is cast aside.

The authors of WordPress, to their credit, have spent a great deal of time on the user interface, trying to ensure that it is straightforward and unsurprising for nontechnical users. For example, many Weblogs assume that the author knows basic HTML tags and understands how to add and remove them. The WordPress editing window lets advanced users place such tags manually, but it also provides a set of JavaScript buttons for less-experienced users. These buttons not only indicate what possibilities exist, but change shape to indicate that a particular format currently is in use. The Close button at the end of the row was a particularly clever idea; it closes all of the currently open tags.

Like most software with good user interfaces, WordPress includes many small features that add up to a pleasant experience. For example, it automatically creates en and em dashes when you use two or three hyphens. It allows you to classify postings as drafts (the default), private or public, which means that you can start working on a Weblog entry, go to lunch and then return to work on it.

WordPress is sensitive to the fact that deleting data is a dangerous act that the user always should have to confirm. Thus, whenever the mouse moves over a delete button, the background of that button turns red. As an additional reminder, users are asked to confirm the action with a JavaScript dialog box.

With the exception of deleting, it is easy to undo any mistake that you might make in WordPress by returning to the menu in question and changing the value. All entries, including drafts and comments, can be edited repeatedly until they are ready for publication.

Finally, the look and feel of a WordPress Weblog can be changed by modifying the CSS, which handles the fonts, colors, sizes and placement; the templates, which largely are standard PHP; and even the plugins, which can change almost anything. It is possible to change the template within WordPress itself, although I expect that most readers of this magazine would prefer to use Emacs or vi to change the file directly on disk.

Installing a plugin requires downloading it and placing it within the appropriate plugins directory, but activating it is completely Web-driven. This means that system administrators can install a number of plugins for their users and let the individuals choose which plugins they would like to activate. Several sample plugins are included in the WordPress distribution, and others are available from the WordPress Web site.

Conclusion

Over the past few months, we have looked at a number of different types of Weblog software. With the exception of COREBlog, a Zope product that installed easily and quickly into my Zope server, WordPress was by far the easiest and fastest to install. It has a full list of features, many of which have to do with the clean, easy-to-use user interface. Even novice computer users and Webloggers can publish regularly with this software. Although the underlying code and technologies used—PHP and MySQL—are not my favorites, the set of features, growth of the platform and the large community all make WordPress a winning choice.

Resources for this article: /article/7641.

Reuven M. Lerner, a longtime Web/database consultant and developer, now is a first-year graduate student in the Learning Sciences program at Northwestern University. His Weblog is at altneuland.lerner.co.il, and you can reach him at reuven@lerner.co.il.

Kernel Korner

Extending Battery Life with Laptop Mode

Bart Samwel

Issue #125, September 2004

Economize on hard drive usage and extend your laptop's battery life.

Laptops give you the freedom to do whatever you want, wherever you want to do it. But when your battery runs out, the fun is over. Fortunately, there are a lot of ways to save power and make your battery last longer. For instance, you can lower the processor speed, dim the display's backlight and spin down the hard drive. The first two tricks work well on Linux, but until recently, spinning down the hard drive could be quite a struggle. Even if you could get the drive to spin down, it never would stay down long enough to save any power. This article explains how you can use Laptop Mode, a feature recently added to the Linux kernel, to spin down the drive for real. I talk only about Linux 2.6 here; a Laptop Mode is available in 2.4, but it is a bit different.

Hard Drives and Battery Life

Let's do a little math to find out how much extra battery life you can get by spinning down the hard drive. A typical laptop on the market today has a lithium-ion battery with a capacity of 50–100 Watt-hours of power, which is good for two to four hours. Say we have a laptop with a battery of 50 Watt-hours. If the battery lasts for 3.5 hours with the hard drive on, then we can calculate the average power usage as 50/3.5 = 14.3 Watts. Say the laptop uses a typical laptop hard drive, which uses about 0.9 Watts in idle mode and about 0.3 Watts in standby. In theory we can reduce the power usage by 0.6 Watts, to about 13.7 Watts. This increases battery life to 50/13.7 = 3 hours and 39 minutes. The gain always depends on how much power you save relative to the total power usage. In our example, the spindown saves about 4% of the total power, so the maximum gain in battery life is about 4% as well.

So much for theory, I want to show you some real data. I borrowed a friend's Apple PowerBook G4, installed Debian GNU/Linux and the Linux 2.6.6 kernel and then did some experiments. I wanted to estimate the maximum gain in battery life and the time we needed between spinups to come close to that maximum. I expected a pretty large gain, because the laptop was equipped with a power-hungry 5,400 rpm drive and because I stripped the system of the X server and all dæmons. I wrote a benchmark program that always performs the same amount of disk I/O per hour, but with a configurable inactive period between I/O bursts. During inactive periods, the benchmark program spins down the hard drive. I ran the benchmark with a number of inactive period lengths, and I used the APM battery information to calculate the expected battery life.

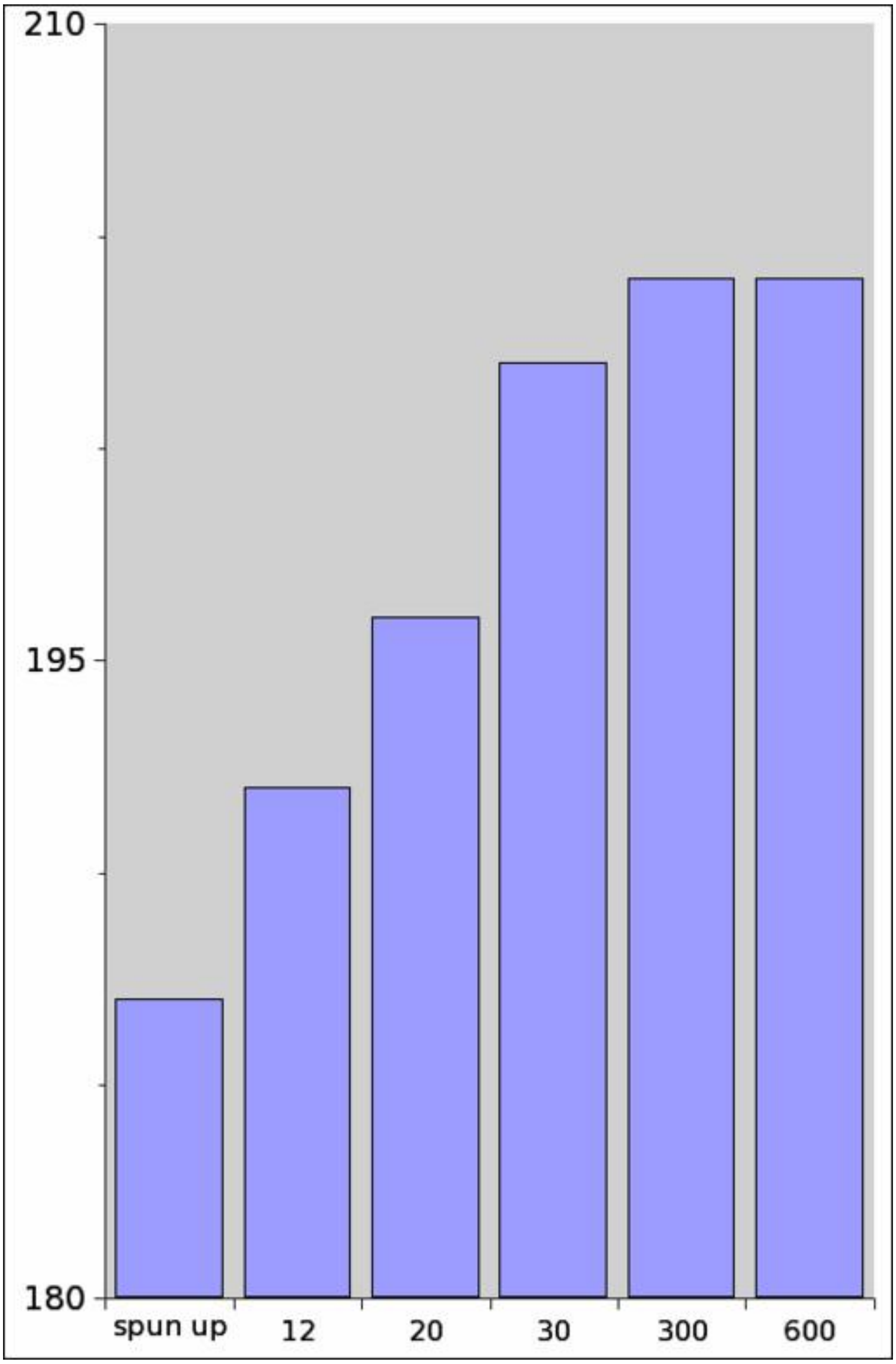


Figure 1. Results of the Battery Life Experiment

I've run this experiment with the disk spun up all the time and with burst intervals ranging from 12 seconds to ten minutes. The results are illustrated in Figure 1. As you can see, as soon as you do I/O less than once every 30 seconds, you have pretty much saved all you can. This seems strange, because spinning up the disk costs a lot of power, right? No, actually, it doesn't. If the drive can spin up in two seconds, that takes about as much power as keeping it running idle for, say, eight seconds. So if you can spin that drive down for nine seconds, you've already saved some power. The 30-second burst interval already leaves room for a pretty long spindown, which is why it shows such good benchmark results.

Laptop Mode

Laptop Mode is a setting for the Linux kernel that changes how the kernel distributes disk I/O over time. Linux normally does disk I/O in small amounts, nicely spread out over time. But with all that I/O going on, your hard drive never gets a chance to spin down, wasting valuable power. For a laptop, disk activity must be concentrated into short stretches of time, with periods of inactivity between them, like I did with the benchmark. When you enable Laptop Mode, Linux does exactly that. You can get stretches of up to ten minutes without disk activity, which definitely improves your laptop's battery life.

How It Works

Let's take a look at what Laptop Mode does to get that kind of I/O behaviour. To create periods without disk activity, we need to do as much as we can during the active periods. After that, we need to hold off disk I/O for as long as possible. During active periods, we do a couple of extra things. First, we perform some read-ahead; if that data actually is needed during the inactive period, we've saved a spinup. Laptop Mode sets the read-ahead to 4MB by default. Second, we sync everything to disk at the end of every active period. This keeps your data safe; when the drive has spun down, you can be sure that everything done up to the spindown is stored safely.

During an inactive period, writes are the only kind of disk I/O we can hold off. We can keep the unwritten data in memory for as long as we like or until we're out of memory. Unfortunately, this was not so easy for us to implement, because Linux submits write requests from many places. We needed to tweak all those places to hold off their writes.

The first and most important tweak has to do with modified or dirty data. Normally, when a cached disk page has been modified more than 30 seconds ago, it expires, and the pdflush dæmon writes it to disk. Fortunately, the expiry interval is configurable through /proc/sys/vm/dirty_expire_centisecs. Laptop Mode sets it to ten minutes so that changes stay in memory for up to ten minutes before they're written to disk. Because every active period is ended with a sync, the inactive period starts without any dirty pages. Therefore, during the first ten minutes of an inactive period, we can be sure that no pages are written back because they expire.

The second tweak concerns journaling filesystems, which do a lot of disk I/O themselves. On most of the journaling filesystems supported by Laptop Mode, a change to the filesystem triggers a write operation within five seconds. For instance, in the ext3 filesystem, a filesystem transaction has a maximum lifetime before it is automatically committed, and committing means writing to disk. This maximum lifetime can be configured using the commit mount option. By remounting the filesystem with this option set to ten minutes, we stop ext3 from committing transactions during an inactive period. Again, we start every inactive period with a sync, so no transactions are open when the inactive period starts. Laptop Mode extends a similar treatment to the other supported filesystems, ReiserFS and XFS.

The final tweak occurs in Linux's memory management. If a lot of memory is allocated during an inactive period, the memory manager eventually has to select some memory pages that need to be dropped. It is possible to select a page that needs to be written to disk before it can be dropped, for instance a modified disk page or a page that needs to be written to swap space. But then, it has to spin up the drive to perform that write, and we don't want that to happen. Andrew Morton tweaked the memory manager so that when we're running in Laptop Mode, the memory manager first tries to select pages that don't require a write.

Using these tweaks, Laptop Mode can create up to ten minutes without disk activity. When you're not changing any files at all, you can get even longer periods without spinning up the disk. After all, if there's nothing to write, there's no reason to spin up the disk. Unfortunately, when you've mounted the filesystems with the default options, things change by themselves; the filesystem records access times. Access times are updated even when you're only reading files, and they must be written to disk eventually. To avoid this problem, Laptop Mode remounts all filesystems with the noatime mount option. This makes them stop recording access times, so you actually can get more than ten minutes of time without disk I/O.

As you might have noticed, we're doing some things typically done from user space, such as tweaking /proc. In fact, we've split Laptop Mode into a kernel component and a user-space script. You can use the script to enable laptop mode, and it enables the kernel support by setting /proc/sys/vm/laptop_mode. It then remounts your filesystems and tweaks some other settings in /proc as well.

Setting It Up

To set up Laptop Mode on your system, first make sure you have a kernel version that supports it. Laptop Mode is included in Linux 2.6 from version 2.6.6 upward. In the kernel source tree, you can find the documentation for

Laptop Mode in Documentation/laptop-mode.txt. Embedded in the documentation is the control script, which you have to extract and save as /sbin/laptop_mode. Give the script execute permissions: `chmod 700 /sbin/laptop_mode`.

To enable Laptop Mode, run (as root) `/sbin/laptop_mode start`. This does everything necessary, except it doesn't make your hard drive spin down. To do that, you must set the hard drive's idle timeout using `hdparm -S 4 /dev/hda`. The value 4 indicates an idle timeout of 20 seconds. If you want to disable Laptop Mode, simply run `/sbin/laptop_mode stop`.

You probably want to configure Laptop Mode so it starts whenever your laptop runs on batteries. If you have a laptop that supports ACPI, you can set this up like so: extract the files ac_adapter and battery.sh from the Laptop Mode documentation and install them in the indicated locations. Edit battery.sh to configure your hard drive's device name and your preferred idle timeout, and you're ready to roll.

Spinup Debugging

Sometimes your drive spins up for reasons you don't understand. When this occurs, it's time to start debugging. Laptop Mode includes a mode for debugging disk activity, block dump mode. Before you enable it, you first must stop syslogd from logging kernel messages or stop it completely. How this is done depends on your distribution. If you don't stop syslogd, you may put your machine in an endless loop; the debug output is picked up by syslogd, which writes it to disk, causing more debug output and so forth.

To enable block dump mode, run (as root) `echo 1 > /proc/sys/vm/block_dump`. The kernel output, which you now have to read using `dmesg` because syslogd is inactive, should show messages such as these:

```
bash(273): READ block 3242 on hda1
bash(273): dirtied inode 10237 (.bash_history) on hda1
pdflush(6): WRITE block 3242 on hda1
```

You can read this output as follows. A process named bash, with process ID 273, has read the block with number 3242 on device /dev/hda1. That same process then dirtied a file called .bash_history; the file was changed, but the changes weren't written to disk yet. The pdflush dæmon then wrote block 3242, which most probably is the block that bash modified earlier.

When you've got the debug output, it's time to diagnose your problem. If you're seeing a READ message somewhere, you don't have to look further. Find out why the process needs to read this data and decide whether you want to stop the process, change the application's settings so that it doesn't need that data anymore or perhaps read the data ahead when the drive is spun up. Reading a file ahead is no more difficult than doing `cat /my/file >/dev/null`, preferably twice, so Linux's read-once logic does not throw the file right out again. Now, if you're seeing only dirtied file messages, there's not much to worry about. Nobody is writing anything to disk; these messages tell you only that a process is making changes that need to be written eventually. If you get these, your disk spins up once every ten minutes to write back the modifications, and that's it.

If you're seeing WRITE messages more often than once every ten minutes, and you're not seeing any READs that could have triggered an active period, there's a good chance that some process is syncing a file explicitly. syslogd is notorious for doing this. If you see syslogd writing at unexpected times, you should adjust your syslog.conf. It probably contains lines like this one:

```
kern.*    /var/log/kern.log
```

This line tells syslogd to call fsync() after every log message matching kern.*. If you change the line to this:

```
kern.*    -/var/log/kern.log
```

and restart syslog, syslog no longer calls fsync() on these log messages. Be careful for which log files you make this change, though. For instance, if you care about security, you probably want to keep auth.log synchronized.

Tips and Tricks

MP3 Playing

If you want to play MP3s with your drive suspended, you can try increasing the READAHEAD setting in /sbin/laptop_mode. Alternatively, you can copy a whole set of MP3s to a small tmpfs RAM disk. Make sure you don't make the RAM disk so big that it gets pushed out to swap space or you defeat the purpose. A small RAM disk is good for anything for which you don't want disk activity to occur—but only if your data isn't important.

Custom Spindown Times

It's possible to adjust Laptop Mode's maximum spindown time. To do that, set variable MAX_AGE in /sbin/laptop_mode to the maximum number of seconds you want your data to remain in memory without being saved to disk. By default, it is set to 600 seconds or ten minutes. It really doesn't pay to increase the setting to a higher number, as you can see from the benchmark results that I got on the PowerBook. You can set it to a lower value if you care more about your data and less about those extra two minutes. And if you want to prevent losing some really important data that you've just saved, you simply can execute `sync`. Laptop Mode respects your sync requests, so this actually writes everything to disk, as it normally would. The sync also resets all timeouts, so that after the sync the disk can stay spun down for up to ten minutes again.

Spinning Down with cpudyn

If you are using cpudyn to control your CPU frequency dynamically, you might be interested to know that it also can spin down your hard drive. I have seen some drives that simply ignore their idle timeout setting if it's too low for their taste. Also, idle timeout settings are done in five-second increments, which means that you can't set it to something like eight seconds. cpudyn comes in handy, because it doesn't rely on your hard drive to detect idle periods, and it can spin it down whenever you want.

Smart Spindown

Laptop Mode performs a sync at the end of an active period and then waits for the disk to spin down on its own. It's even better to sync just before the drive spins down, because then you save data that is up to 20 seconds more recent. Laptop Mode can't do this, because it can't spin down the drive actively. I have written a script to do this called Smart Spindown, which works together with Laptop Mode. It's an unpolished piece of script, but if you really want to save every bit of power there is or if you want to keep 20 extra seconds of data safe, this might be for you; see the on-line Resources section.

Acknowledgements

Even though I don't have a laptop myself, I've had a lot of fun working on Laptop Mode. I want to acknowledge the efforts of the other guys who have contributed to Laptop Mode, including Jens Axboe, Micha Feigin, Andrew Morton and Kiko Piris. I also want to thank Jeroen Kruis for allowing me to abuse his brand-new PowerBook G4 for my experiments.

Resources for this article: /article/7647.

Bart Samwel initiated the effort to put Laptop Mode in Linux 2.6, even though he does not own a laptop. He studies Computer Science at Leiden University, The Netherlands, and earns his living writing proprietary software. He can be reached at bart@samwel.tk.

Archive Index Issue Table of Contents

Advanced search

Cooking with Linux

The Wireless Kitchen

Marcel Gagné

Issue #125, September 2004

Manage your wireless networking settings as you move from place to place, and keep an eye out for the spot with the best signal.

No, thank you, François, I don't need to sit down right now. Yes, I see that the tables are free. That's because our guests haven't arrived yet. *Quoi?* Ah, I see. You're wondering why I am wandering around with this notebook in my hands and not taking a moment to sit down. The reason is simple, *mon ami*. I've installed wireless access points in each wing of the wine cellar so we and our guests can have wireless access. I'm wandering around with my notebook to check signal strength. I want to make sure that no matter where our guests make themselves comfortable, they have access.

Welcome, *mes amis*, to *Chez Marcel*, home to the finest in Linux cooking, a superb wine cellar and, now, wireless Internet access. Please sit and make yourselves comfortable. François, head down to the wine cellar and bring back the 1998 Eden Valley Riesling from Australia, *immédiatement*!

Before you came in, *mes amis*, I was busy making sure that wireless access was available in all parts of the restaurant. In the case of a classic, wired network card, a little light let you know whether you actually were plugged in to the network. This isn't always so cut and dry with wireless cards. Factors such as distance, signal strength and even the location of access points dictate what kind of service you can expect. That's all fine, if you have that information at your disposal.

The tools on today's menu offer a variety of approaches to providing wireless card information, from signal strength to access-point availability. A number of these applications' authors thank Jean Tourrilhes for his work on the Linux kernel wireless extensions, so I take a moment here to thank him as well. As François pours the wine, it's time to sample the first item on today's menu.

As I've mentioned in the past, I'm rather fond of WindowMaker dock apps, those self-contained programs that take up so little room on the desktop. Consequently, I have three such apps for you starting with Jess Mahan's WmWiFi (Figure 1). This little WindowMaker application caught my eye partly, I suppose, because it looks a little like the signal meter on my old cell phone. When you download it (see the on-line Resources section), you can choose from Debian packages as well as the source. Building the package from source requires your old friend, the extract and build five-step:

```
tar -xzvf wmwifi-0.4.tar.gz
cd wmwifi-0.4
./configure
make
su -c "make install"
```

To run it, you can type `wmwifi` at the command line and leave it at that, but you may find that it doesn't appear too easily if you aren't running WindowMaker. Try this command instead, `wmwifi -bw`. The -bw, if you are curious, stands for broken window manager. Never let it be said that Jess Mahan did not have a sense of humor. Incidentally, the gray background is WmWiFi's default appearance, but clicking on the dock app turns on the backlight, which then glows LCD green.

In a similar vein, Carsten Schrmann's wmWAVE (now maintained by Jens Schrmann) offers a nice compact signal strength meter for your wireless card, plus a little bit more. There's also an overall link quality reading and a reading on the noise level. Building this app is easy. Simply extract the tarred and gzipped bundle, type `make` and `make install`. To run wmWAVE, execute the binary called wmwave

The third of my WindowMaker apps is Ico Doornekamp's wmifinfo. Unlike the others, this isn't strictly a wireless application. Instead, it automatically detects all of your network interfaces and provides information on each one. With a wireless card, it also displays a signal strength meter at the bottom of the dock app itself. See Resources for the source for wmifinfo.

Once again, this is an easy build: simply extract, type `make` and `make install`. To run it, type the command `wmifinfo`. If you are fond of the gray LCD look, use the `wmifinfo -l` option. To switch between interfaces, if you have multiple cards, left-click on the dock app. By using the -i option, you can specify a certain interface with which to start (`wmifinfo -i eth0`).

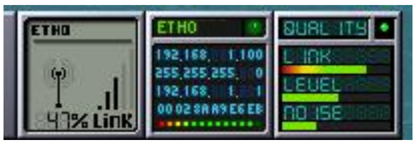


Figure 1. Three WindowMaker Wireless Dock Apps: WmWiFi, wmifinfo and wmWAVE

Longtime command-line fans such as myself always are excited about finding a nongraphical application that does a great job and manages to do it with style. Jan Morgenstern's wavemon is such an application. In a single ncurses display, wavemon offers a great deal of network information, including signal, noise and quality levels. The information is displayed in a dynamic bar graph; see Figure 2. wavemon also includes a histogram view and onscreen configuration.

Figure 2. wavemon shows signal and noise levels and more.

Building wavemon is the classic extract and build five-step again. To run wavemon, type `wavemon` at the command line. When the display starts up, you should see the summary information screen, which includes signal levels, TCP statistics, information on mode, encryption, bitrate, your access point, local network card and more. All of the functions are controlled with function keys. To switch to the histogram view, press F2 and then F1 to return to the summary screen. The configuration screen is accessed by pressing F7. Reminders about each function key's function are at the bottom of the screen.

With the release of KDE 3.2, users of the popular desktop environment will find a nice new application called KWiFiManager. This application, written by Stefan Winter, provides signal and quality information like the programs previously mentioned, but there is much more to KWiFiManager. It also makes it possible to maintain up to four different configurations. The road warriors among you who move from office to office should find this one particularly useful.

KWiFiManager may not be installed by default. The easiest way to check is either to look in your KDE menu or simply type `kwifimanager` to start the program.

When KWiFiManager starts, it displays a basic status window showing signal strength, connection speed, access point, local IP address and the frequency of the channel. Depending on the actual strength of the connection, you see words like TOP, EXCELLENT or ULTIMATE accompanying the numeric signal strength indicator. You also see a small icon in your system tray with a numeric and graphical signal quality indicator. Click File on the menu bar and select Connection statistics for a graph of signal quality over time (Figure 3).

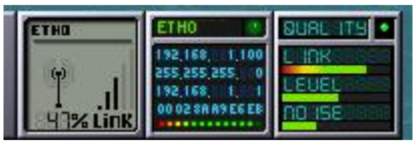


Figure 3. KWiFiManager reports real-time connection information and graphed statistics over time.

For those who feel cheated that computers aren't providing all the lights, beeps and science-fiction noises technology promised, KWiFiManager has an audible feature to satisfy that craving. A beeping tone that increases or decreases in pitch with the signal can be turned on. Click Config on the KWiFiManager menu bar and select Acoustic Scanning. I warn you, *mes amis*, that you may find this becomes tedious pretty quickly, but it is fun for short periods.

KWiFiManager goes well beyond being a signal strength meter. It can help simplify your network configurations by providing automatic connections to the various networks you may encounter. Click Config on the menu bar and select Configuration Editor. At first, you are asked to enter the root password and then a somewhat larger window appears.

Figure 4. Up to four preset wireless configurations are possible.

Along the top, you see four tabs labeled Config1 through Config4. Each of these areas allows you to prepare a configuration based on whatever network you are going to be accessing, whether at home or on the road. I tell you about these settings in a moment, but first I want to direct your attention to the bottom of the KWiFiManager window. There's a check box to allow you to Load preset configuration on KDE startup. If you check that box, you can select one of your four configurations as the default. If you generally are in your office when you start your notebook, you can select that as your default network setup for whenever you log in.

Now, back to the rest of these settings. The most basic aspect to each of these configurations is right at the top, the Network name. Each of the four configurations can have a network name associated with it that is specified in your access point. If you want KDE to activate your interface and connect you to whatever network is available, enter ANY as the network name. Off to the right of that, you can select the connection speed. I personally leave this set to Auto.

Right below these settings is a field labeled Execute script on connect. This can be any kind of a program you want to run, including a script. Perhaps you'd like to play a tune or have your system suggest a wine. I want to run custom firewall scripts depending on where I am and what services I like to make available on the network in question. Speaking of security, a little further down, you should see a check box labeled Use cryptography. If you are using a wireless access point with WEP encryption, make sure this box is checked. Some new fields appear in the application window where you can specify and enter your key passphrase information. Click OK (or Apply if you want to keep making changes) and you are done.

Incidentally, you also can access this configuration dialog through the KDE control center; the program name is kcontrol. Look under the control center's Internet menu.

Once again, *mes amis*, closing time has arrived all too quickly, but *Chez Marcel* is never too strict with the hours. Enjoy a final glass of wine and enjoy the wireless access. Until next time, *mes amis*, let us all drink to one another's health. *A votre santé Bon appétit!*

Resources for this article: /article/7643.

Marcel Gagné (mggagne@salmar.com) lives in Mississauga, Ontario. He is the author of *Moving to Linux: Kiss the Blue Screen of Death Goodbye!* (ISBN 0-321-15998-5) from Addison Wesley. His first book is the highly acclaimed *Linux System Administration: A User's Guide* (ISBN 0-201-71934-7). In real life, he is president of Salmar Consulting, Inc., a systems integration and network consulting firm.

Archive Index Issue Table of Contents

Advanced search

Paranoid Penguin

Rehabilitating Clear-Text Network Applications with Stunnel

Mick Bauer

Issue #125, September 2004

Put modern crypto onto your legacy applications without modifying them. Mick Bauer shows how to bring pre-SSL programs up to date.

The world is full of network applications that get things done, have withstood the test of time and have rotten security. Telnet? Brilliant in its simplicity and versatility, but it transmits your login credentials in clear text. rcp? Ever so ubiquitous and scriptable, but rhosts-based authentication's time has come and gone, thanks to IP spoofing.

Sure, you can swap those old warhorses for their encrypted successors—SSH instead of telnet and scp or rsync-over-SSH for rcp. Or, you can build an all-purpose IPSec tunnel to each remote host with which you communicate. But the latter often is overkill, and the former is easier said than done if your choice of applications in a given scenario isn't completely within your control. Surely there's a way to add strong encryption to legacy network applications.

Consider the mighty SSL wrapper, Stunnel. This month, I explain how to use Stunnel 4.0 in tandem with OpenSSL to bring your legacy applications into the modern era, security-wise. And what about wireless? If you're compelled to use a clear-text network application over a weakly secured wireless medium, such as 802.11b, can Stunnel help? Read on.

Background

You need to understand two things in order to use Stunnel. First, know how your network application uses the network. If it's a simple single-TCP-port application such as telnet, which does all its listening on TCP 23, Stunnel works. If it uses UDP, the portmapper service or any other dynamic port-allocation scheme, Stunnel can't help you. For example, RPC applications don't work, because they use portmapper. FTP uses TCP 21 for control traffic but dynamically assigns arbitrary high ports for data connections, so it's also disqualified.

Second, you need to understand the basics of public key cryptography but not necessarily the ins and outs of X.509 or Public Key Infrastructures. I've described how this works in several previous columns, such as “The 101 Uses of OpenSSH, Part II” [*LJ*, January 2001]. For now, suffice it to say that in public key cryptography, each participant has two keys: a public key that you share with the other participants and a private key that only you possess. Other people use your public key to encrypt things that they want only you to see; you use your private key to decrypt those things.

Digital signatures work backward from encryption. If you sign something with your private key, anybody can use your public key to verify that the signature was generated with your private key and therefore by you. Again, this depends on only you possessing your private key, no matter how many people have copies of your public key.

In the X.509 world, we call the public key a certificate, which, technically, is a public key bundled with digitally signed information about the public key's owner, including name and e-mail address. We call the private key simply the key. Somewhat confusingly, we sometimes refer to both of them, together, as a certificate. Context helps: when I talk about a passphrase-free certificate, you know I'm talking about a combined key/certificate because the certificate itself, being a public key, can't have a passphrase.

That's by far the most concise explanation I've ever given of public key cryptography and X.509. If it isn't enough for you to decipher the rest of this article, read the Stunnel FAQ or the mighty RSA Crypto FAQ (see the on-line Resources section) for more information. Now it's time to plunge into Stunnel proper.

Installing Stunnel

Chances are, your Linux distribution of choice includes a binary package for Stunnel. Recent releases of SuSE, Fedora and Red Hat Enterprise all include Stunnel version 4. Debian 3.0 (Woody) includes Stunnel version 3.22.

On the one hand, 3.22 is a stable version that's well documented and well understood. On the other hand, Stunnel version 4 is a major rewrite that, among other things, allows for easier management of multiple tunnels. It's the version I'm covering here. If you run Debian, I think it's worth your while to download the latest Stunnel source and compile it yourself.

Compiling Stunnel on any Linux distribution is quick and easy. First, make sure you've already got your distribution's packages for OpenSSL, probably called openssl; OpenSSL development libraries, openssl-devel or libssl096-dev; and TCPwrapper development libraries, libwrap0-dev on Debian, included as part of SuSE's and Fedora's base installations. Then, unpack Stunnel's source code tarball and do a quick:

```
./configure && make && make install
```

If for some reason that doesn't work, entering `./configure --help` lists advanced precompile configuration options you can pass to the configure script. Once you've installed Stunnel, it's time to create some certificates and start tunneling.

Most of what follows applies only to Stunnel v.4.0.0 and later. If you choose to use, for example, Debian's Stunnel 3.22 package, you need to refer to the documentation included with that package or to the examples on the Stunnel Web site (see Resources); most of that Web site still covers the older version.

Creating Certificates with OpenSSL

A Stunnel server—that is, a host that receives encrypted packets destined for a local clear-text service—requires a server certificate. A Stunnel client does not, however, unless you intend to configure your Stunnel server to use client-certificate authentication. Sadly, client-certificate authentication is beyond the scope of this article, so none of our examples require the client to have its own certificate—only the server does.

If you installed Stunnel after compiling from source and issuing a `make install` command, you've already got a server certificate (/usr/local/etc/stunnel/stunnel.pem). If you installed Stunnel from a binary package, the package's post-installation script may or may not have created a server certificate for you.

Fedora Core 1's Stunnel RPM creates an empty certificate for some reason. Needless to say, you need a proper certificate, and the way to do that on Fedora is to change your working directory to /usr/share/ssl/certs, type `make stunnel.pem` and copy stunnel.pem to the directory /etc/stunnel. This certificate, as with certificates created by the Stunnel source-release Make script, has no passphrase.

The Danger of Passphrase-Free Certificates

Using a server certificate that's been generated automatically or semi-automatically by a script is quick and convenient, but there's one problem: such a certificate nearly always is created with no passphrase. On the one hand, you can and should set the permissions and ownership on your certificate to make it root-readable only, so at least unprivileged users on your system aren't able to read and thus use it. On the other hand, if your system is root-compromised, your passphrase-free certificate then can be used and abused by the attacker.

This may be an acceptable risk for you, and indeed, using passphrase-free server certificates is a common practice. After all, it's inconvenient to require a human being to enter a certificate's passphrase every time Stunnel starts up. However, as a matter of principle, crypto experts widely consider it reckless to use passphrase-free certificates. If a process is sensitive enough to warrant cryptography in the first place, the argument goes, it's sensitive enough to require a human being to start that process.

If your Stunnel server system needs a server certificate, or if your automatically generated certificate doesn't meet your needs—for example, it lacks a passphrase—you need to generate your own, using the OpenSSL command. Space does not permit me to give a full, proper explanation of this powerful tool. Luckily, the Stunnel FAQ (see Resources) answers some of the most common questions about how OpenSSL and Stunnel interact, and it includes pointers to further sources of information about OpenSSL.

Suffice it to say that for some users, there may be a compelling reason to create their own Certificate Authority (CA), install their CA certificate (but not key) on every system running Stunnel and use their CA certificate to sign all server certificates. This is necessary, for example, if you want your Stunnel server systems to accept client connections only from Stunnel client hosts with particular certificates.

For many if not most Stunnel users, however, it's good enough to use self-signed certificates and to not worry about being or not being a CA. Here's how to generate a self-signed server certificate using OpenSSL.

Listing 1. Creating a Server Certificate with OpenSSL

```
nearclient:/etc/stunnel# openssl req -x509 \
-newkey rsa:1024 -days 365 \
-keyout stunnel.pem -out stunnel.pem
Using configuration from /usr/lib/ssl/openssl.cnf
Generating a 1024 bit RSA private key
...++++++
.............................................++++++
writing new private key to 'key2.pem'
Enter PEM pass phrase: ************
Verifying password - Enter PEM pass phrase: ******
-----
You are about to be asked to enter information that will be
```

```
incorporated into your certificate request.
What you are about to enter is what is called a
Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) []:Minnesota
Locality Name (eg, city) []:St. Paul
Organization Name (eg, company) []:Wiremonkeys
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:nearclient.wiremonkeys.org
Email Address []:X.509master@wiremonkeys.org

nearclient:/etc/stunnel# chmod 600 stunnel.pem
```

In Listing 1, the real meat is in the first line. Parsing from left to right, this command tells OpenSSL to create a certificate request in the X.509 digital certificate format, using the RSA cipher with a 1,024-bit key, good for 365 days and with both its key and (public) certificate written to the same file, stunnel.pem. If you instead want to create a password-free certificate, you can include the -nodes option at the end. Read the sidebar “The Danger of Passphrase-Free Certificates” first, though.

Most of the rest of Listing 1 shows a dialog in which OpenSSL prompts me for X.509-type information to include in the certificate, including Country and Region. That stuff should be self-explanatory, but Common Name is very important. If you're creating a server certificate, the certificate's Common Name must be set to the fully qualified domain name (that is, the full hostname) of the host that is going to use the certificate.

Most SSL/TLS-enabled applications expect this Common Name to resolve by way of DNS to the IP address of your server. Stunnel does not unless you configure it to, but setting Common Name to your FQDN is a good habit to get into nonetheless.

In the last line of Listing 1, I've set the permissions of my new server certificate to 0600 (-rw-------). Because I ran my OpenSSL command as root, this file already is owned by root. It's important for any server certificate, whether it's password-protected or not, to be root-readable only. I ran my OpenSSL command from within /etc/stunnel, so my certificate was created in place, and I didn't have to move my new certificate there by hand.

Configuring Stunnel's Global Settings

Once you've got a suitable server certificate, it's time to configure yourself a tunnel. This is considerably simpler than the previous task. This part is also much more version-specific. In versions of Stunnel prior to version 4.0, Stunnel accepted all of its configuration parameters as command options. In current versions, however, the only actionable command-line argument it expects is the nondefault path to its configuration file.

If you installed Stunnel from source with default compile-time options, Stunnel expects its configuration file to reside in /usr/local/etc/stunnel. If you installed from a binary package, this path is more likely to be /etc/stunnel. Listing 2 shows the global settings from an abbreviated sample stunnel.conf file (abbreviated mainly in that I've omitted comment lines).

Listing 2. Global Settings in stunnel.conf

```
cert = /etc/stunnel/stunnel.pem
chroot = /var/run/stunnel/
pid = /stunnel.pid
setuid = nobody
setgid = nogroup
debug = 7
output = /var/log/stunnel.log
client = yes
```

The cert parameter tells Stunnel where to look for its server certificate; it therefore follows that you need this parameter only on your Stunnel server host, not on client hosts. The chroot parameter tells Stunnel which directory to chroot itself (reset / to) after starting up; this happens after Stunnel has read its configuration and server certificate files. You probably need to create this chroot jail manually and populate it with a few things, for example, its own etc/ hosts.allow and etc/hosts.deny files, if you want to use TCPwrappers-style access controls.

pid tells Stunnel where to write its process ID. This path is relative to that set by chroot; that is, Stunnel writes its PID after chrooting itself.

setuid and setgid tell Stunnel which user and group to demote itself to after starting. If Stunnel is to listen on any TCP ports lower than 1025 it must be started as root, but it demotes itself after reading its configuration file, reading its server certificate and binding to the privileged port.

By default, Stunnel sends its log messages of severity notice or higher to the local dæmon syslog facility. Fedora's version sends it to authpriv, which in turn logs to /var/log/secure. You can use the debug option to set a different log level. Seven is the highest level and is best if you're having trouble getting Stunnel to work. You can use the output option to tell Stunnel to send its messages to a specific file rather than handing its messages off to syslog.

The last line in Listing 2 sets the client parameter to yes, which means that on this particular system, I intend to initiate SSL transactions, not receive them. On the server with which I intend to communicate, I need to leave this parameter set to its default, no.

Configuring a Tunnel

Now, finally, we come to the payoff—an actual tunnel. For this example, we're going to tunnel telnet from the host nearclient to the server farserver. The global section in farserver's stunnel.conf file can be almost identical to the one in Listing 2, except that the client needs to be set to no. The major difference between the two hosts' configurations is in their service definitions.

Before I dive into that, however, let's flesh out the example scenario a little more. Suppose farserver already is configured as a telnet server; it already accepts telnet sessions on TCP port 23. But, we don't want nearclient to connect to the clear-text port; we need to use something else for an SSL connection. As it happens, IANA has already designated a port for SSL-enabled telnet (aka telnets): TCP 992.

Therefore, we want a tunnel from nearserver to TCP 992 on farserver. But how will our non-SSL-enabled telnet command and our equally non-SSL-savvy telnet server process know how to use this tunnel? That's a trick question; the tunnel is completely transparent to the sending and receiving telnet processes.

nearserver's Stunnel process accepts the connection on the usual port—TCP 23 although this is user-defined—and then encrypts the packets with SSL before forwarding them to TCP port 992 on farserver. farserver decrypts the packets and then forwards them to its local telnet process on TCP 23. Actually, xinetd or inetd receive the packets before in.telnetd does, but you get the picture.

In this way, when users on nearserver want to connect to farserver, they enter the command `telnet 127.0.0.1`, and connection is encrypted, forwarded to farserver and decrypted. farserver's reply packets follow the same path but backward. Each telnet process (telnet and in.telnetd) thinks it's communicating with a local user, but the packets in fact are traversing an SSL-encrypted Stunnel session. All of which is a wordy way of explaining the six total lines that comprise Listings 3 and 4.

Listing 3. Service Definition on nearclient

```
[telnets]
accept = 23
connect = farserver.wiremonkeys.org:992
```

Listing 4. Service Definition on farserver

```
[telnets]
accept = 992
connect = 23
```

As you can see, a service definition can be as simple as a pair of lines, an accept line to designate a listening port and a connect line to designate a destination port. Precisely what this means depends on whether a given system is a Stunnel client or a Stunnel server. On a client system, client = yes, Stunnel expects clear-text packets on its accept port and sends encrypted packets to the connect port. On a server system, client = no, Stunnel listens for SSL connections (encrypted packets) on its accept port and sends decrypted packets to the connect port.

Also notice that on a client, in your connect statement you probably need to specify not only a port but also a remote hostname or IP address, as in Listing 3, `connect = farserver.wiremonkeys.org:992`.

According to the way we've configured things in Listings 3 and 4, any host can connect to TCP port 23 on nearserver and traverse nearserver's tunnel to farserver. We can restrict this in any number of ways, including by using iptables. We can tell Stunnel to accept only local connections to the Stunnel client process, however, by using `accept = 127.0.0.1:23` on nearclient.

This technique would work on farserver as well. If farserver has multiple interfaces, eth0, wlan0, ppp0 and so on, you can write your accept statement to specify the IP address of the interface on which you want it to receive tunneled packets in the same way. On both Stunnel clients and servers, the default IP for accept statements is any (all local interfaces), and the default IP for connect statements is 127.0.0.1 (localhost).

What about the telnet server on farserver? After all, clear-text telnet sessions seldom are a good idea. Therefore, in practice you want to be sure to add the line `bind = 127.0.0.1` to your /etc/xinetd/telnet script, so that only local processes can connect to TCP 23.

Once you've finished configuring Stunnel on client and server, start Stunnel simply by typing the command `stunnel`. You don't need to worry about starting it on the server before starting it on the client; the client doesn't initiate a tunnel until you try to use it, for example, by trying to start a telnet session. If either or both hosts are using a password-protected server certificate, you are prompted for it now. Keep this in mind if you set up a Stunnel startup script. Once you've entered the certificate, you should be up and running.

You now can check for successful startup by issuing a quick `ps auxw` and looking for a Stunnel process—Stunnel returns no output to the console whether or not it starts cleanly. It does, however, send messages to your system's syslog facility, including startup messages.

Once stunnel is running on both client and server, our users on nearclient can trigger a tunnel by connecting to nearclient's TCP port 23, for example, with the command `telnet 127.0.0.1` (23 is telnet's default port). Listing 5 shows a sample encrypted-telnet session.

Listing 5. Sample Telnet Session

```
nearserver:/usr/local/etc/stunnel# telnet 127.0.0.1
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Fedora Core release 1 (Yarrow)
Kernel 2.4.22-1.2115.nptl on an i686
login: myfaraccount
Password: **********
Last login: Sun Jun 13 21:39:17 from localhost.localdomain
[myfaraccount@farserver myfaraccount]$
```

As you can see, from the end-user's perspective, connecting to farserver in this way is practically indistinguishable from any other telnet session. The bash prompt at the end of the listing, however, confirms that we have in fact connected to farserver.

Stunnel and inetd/xinetd

I have a confession. The examples in this article do not show the most efficient way to use Stunnel to encrypt telnet, although I assure you I've tested these examples, and they do work.

My justification is I wanted to illustrate the concept of tunneling generic TCP services quickly, using a technique that works for practically any single-TCP-port service. But telnet, pop3 and other services typically started by a super-server, such as inetd or xinetd, are supported through several different Stunnel features that are beyond the scope of this article.

For example, a Stunnel server can, rather than forward packets with a connect statement, pass them to a process that Stunnel invokes itself with the exec parameter in stunnel.conf. Alternatively, you can configure Stunnel itself to be invoked dynamically by inetd or xinetd.

For more information on using Stunnel with dynamically started dæmons or in conjunction with inetd or xinetd, see the stunnel(8) man page and the Stunnel FAQ.

Conclusion

And that, I hope, is enough to start you down the path of Stunnel mayhem. As usual, I've scratched only the surface. I leave it up to you to explore Stunnel's ability to authenticate tunnels with client-certificate checking, its support for TCPwrappers-style access controls and the myriad global and service-specific options supported in stunnel.conf. Let the stunnel(8) man page be your guide, and may your single-TCP-port-using unencrypted-TCP applications be eavesdroppable no more.

Resources for this article: /article/7646.

Mick Bauer, CISSP, is *Linux Journal*'s security editor and an IS security consultant in Minneapolis, Minnesota. He's the author of *Building Secure Servers With Linux* (O'Reilly & Associates, 2002).

EOF

Faster Training for Smarter Customers

Rich Bodo

Issue #125, September 2004

A new nonprofit training center combines open-source course materials, top-ranked instructors and realistic work environments together for classroom or on-site training.

More is involved in information transparency than mere software. In Mountain View, California, we are working on new methods of training using open course materials. The Freedom Technology Center (FTC) is an educational organization founded to provide high-level technical training to local and on-line communities. The curriculum is based on freely available technical courseware and software courses derived from free software or open-source software projects. What follows is an introduction to FTC and a brief report on what we are doing.

Doc Searls points out that IT is being driven more and more from the demand side, and that's the direction from which we approach training. We understand the trend of empowerment that open-source software is a part of, and we are designing training programs to fit into that paradigm. These days, virtually any IT task can be tackled by a smart self-learner who receives a good introduction from a professional. According to the MIT OCW Program Evaluation Findings Report (March 2004), self-learners are far and away the largest group using freely available training materials. FTC is here to give self-learners a solid introduction and, in many cases, a working sandbox. We kickstart self-learners on the road to competence with freely available tools, training and courseware.

For public benefit, we host free training events to boost awareness and competence on open technical courseware. This is the fun part. We have hosted free training events on spam filtering, trusted computing and open hardware. We ran a book donation drive for the Linux Users Group of Iraq and fund-raisers for FTC and the Free Software Foundation (FSF). As a rule, we try to support Linux users groups and organizations, such as the FSF and Electronic Frontier Foundation (EFF) wherever we can. In short, we support the generation, dissemination, preservation and protection of technical knowledge whenever possible.

We also kickstart corporate self-learners, providing high-quality, hands-on, modular courses. This is the hard work that keeps our lights on. The only way for us to compete is our quality. FTC offers the highest quality training experience available on the topics that we instruct. Our method is first to identify and recruit the single best trainer for each training class we offer. Our instructors all have written the book on their given topic. With the best possible instructor on a topic, and an all-inclusive, hands-on, practical offering, we keep quality at a maximum. We're not the cheapest for corporate training, but in general, if we can't be the best, we don't offer the course. Lucky for us, high-quality trainers seem to like our style.

Geographically, we serve the local technical community in Northern California and a larger on-line community that can access our course notes over the Internet. Our trainers take paid training courses on the road, matching free seminars for users groups and tradeshows on open courseware with paid events for professional users. If we administer it right, we can offer a self-funding free training event or two in any metropolitan area.

Content-wise, we're focusing on open-source applications that people are using in business right now, and that means Samba and OpenLDAP. We have Linux certification and spam-filtering courses, and we are adding modules covering mail and Web serving. Courses on Linux telephony and security issues will be available soon.

Without profit as our prime motive, we're not bound to train using traditional methods or business models. We can explore new technologies and teaching methods. Right now we're working on a model for training whole IT departments called Immersion Training. We have a lot of experience training in corporate environments, and we think the most efficient way to do that is to drop a training team into a company to merge with an IT team and lead it through the design and implementation of new infrastructure, hands-on. When we get there, the team has no infrastructure or training, but when we leave, it has a basic infrastructure and a completely trained and supported staff, with complete documentation of an entire modular, testable process.

Another thing we've been thinking about is how to get corporate trainers to develop and use open courseware. We think corporate trainers should be hired to customize and teach open courseware for their corporate clients. Open courseware is a great marketing tool for trainers, and more trainers should do it. There really is no financial downside for technical trainers if they know how to market their courseware. But we're not simply discussing the economics of new training models that use open courseware, we're proving them viable.

LJ Editor in Chief Don Marti is a board member of the Freedom Technology Center.

Rich Bodo is a parallel entrepreneur and the Managing Director of the Freedom Technology Center. When he is not discussing or implementing a new business, he likes to read and write business software, science fiction and telephony software. He welcomes your comments sent to rsb@freedomtechnologycenter.org.

AML's M7100 Wireless Linux Terminal

Tony Steidler-Dennison

Issue #125, September 2004

In its own way, the device opens new inroads for Linux to the infrastructure of commerce, the business back end that's a critical tool for management in commerce on any scale.

Product Information.

- Vendor: American Microsystems, Ltd.
- URL: www.amltd.com
- Price: $1,795 US

The Good.

- Custom Linux kernel.
- Durability.
- Built-in 802.11b.

The Bad.

- Default telnet for communication.
- Weight.
- Soft keypad.

Although we use the phrase “world domination” somewhat tongue-in-cheek, the growing scope of devices and applications controlled by Linux might make this philosophy a quiet though no less real possibility. Designed for nearly any business task requiring barcode scanning, the AML M7100 is a wireless Linux terminal from American Microsystems, Ltd. By utilizing open-source tools, the M7100 could provide a robust and cost-effective means for businesses large and small to track inventory. In its own way, the device opens new inroads for

Linux to the infrastructure of commerce, the business back end that's a critical tool for management in commerce on any scale.

The M7100 is built around a custom Linux kernel and the widely used Intel StrongARM processor, coupled with 16MB of RAM and 4MB of ROM. This is the same processor found in such devices as the Sharp Zaurus and Yopy Linux-based PDAs. Those devices utilize the StrongARM in 400MHz configurations, while the M7100 processor is a bit slower at 133MHz. Given the purpose-built nature of the device, the slower processor clearly is not a drawback. The M7100 is intended to gather and transmit data to a centralized application.

Weighing in at 16 ounces with a rechargeable battery, the M7100 is built as much for durability as for portability. The weight is due in large part to the device's rugged casing, an enclosure that's been drop-tested onto concrete from up to four feet and survived. This device is designed for use in such locations as warehouses, stockrooms and loading docks, where it could be destroyed easily in a single gravity mishap. To further counteract the weight issue, there's a non-removable nylon wrist strap that's large enough to fit around even my oafish hands.

The M7100 has an internal 802.11b wireless card. The card is functional out of the box, hunting for and connecting to the nearest available open access point without further configuration. Configuration options, including the 802.11b Wireless Equivalent Privacy (WEP) and static or DHCP addressing, are available from a menu.

Figure 1. The 55-key keypad features big number keys and alphabetical letters.

The screen and keypad of the M7100 also reflect the rugged business nature of the device. The screen is a 160×160 backlit LCD, and the 55-key alphanumeric keypad comprises the bulk of the device's front face. The keys are not arranged in the typical QWERTY configuration but alphabetically for the letters, with a block of larger number keys above. Directly above the numeric keys is a four-direction cluster of arrow keys straddled on either side by Enter buttons. Finally, just below the display are small power and backlight buttons surrounding a large and highly visible scan key. This key activates the Long Range Laser scanner for five seconds, allowing plenty of time to capture one barcode and move on to the next. The five-second feature also provides a measure of battery conservation, preventing the unnecessary power drain that would result from firing the laser continuously. With a complement of fully customizable function keys, the keypad is suited to nearly any purpose. The layout also lends itself to use by either right- or left-handed operators. Although the buttons feel a bit soft, they do provide enough tactile response to help prevent manual data entry errors.

The M7100 ships with a docking cradle that doubles as both an in-unit battery charger and a data-transfer dock. The cradle contains an additional slot intended to charge a spare battery, and bright LED lights on the front face signal power, as well as when main and spare batteries are charging. Power connects to the back of the cradle between single USB and RS-232 serial ports. These ports provide ample interfaces for connecting the cradle to a desktop or server. They also provide hard-line alternatives to Wi-Fi for transferring data from the terminal unit to a centralized location.

AML provides a pair of software-loaded CD-ROM discs. One disc contains the CommandLink software, a Microsoft Windows-based tool for data transfer and manipulation. According to the technical support staff at AML, however, the software merely is provided in case the customer doesn't have an existing communication and data-transfer package in place.

The real value in the software resides on the second disc. This disc contains a full API and configuration tools. Detailed documentation and a step-by-step HOWTO for configuring and compiling the StrongARM tools allow in-house developers to create business-specific applications with relative ease. For smaller companies or those with small development budgets, several compatible packages exist that communicate openly with the M7100. These include the TrackWare Application Engine, a GUI builder for creating applications to transform and interpret the data received from the device. In general, though, AML expects that its customers are sending their barcode data to legacy applications. The tools provided in the default configuration are focused specifically on data transfer.

I initially was a bit baffled as to how to test the M7100. Then, as I started to look around my cozy laundry-room office, I realized how ubiquitous the barcode has become in the past few decades. Linux distribution boxes, books, CDs, soda cans and old issues of *Linux Journal* all had barcodes that were scanned easily with the M7100—and that was just my office. The laser on the M7100 picks up the barcodes much more easily and quickly than other scanners. It also picks up labels from a considerably farther distance than others. I was able to scan the barcode on my Dinty Moore stew cans from as far as three feet.

The M7100 provides the capability to transfer this scanned data almost immediately, by way of a telnet session with the server. This session opens as soon as a wireless connection is discovered, communicating with a server defined in the network settings. Multiple servers may be defined in this setting, with simultaneous telnet sessions opened to each server if so desired. The device also fully supports FTP for both uploading and downloading data.

For as easily as the M7100 connects to a designated server on the network, the telnet connection does seem to be a downside. Call me paranoid but, given the security issues inherent in telnet, I'd much rather see this device connect using a more secure protocol such as SSH. The device clearly is intended for use on an internal network, but I find no comfort in using a clear-text protocol on a wireless LAN. Fortunately, AML works with individual customers at no additional cost to provide SSH-based data-transfer capabilities for the M7100.

Overall, AML's M7100 is a strong device, forging new territory for Linux in the warehouses of commerce. It's a well designed tool, achieving the proper balance of mission-specific features and cost. It's rugged yet easy to use. And, although a bit heavy to carry for an eight-hour day, the device includes a rugged wrist strap that should minimize the arm strain. The AML M7100 is available only through resellers.

Tony Steidler-Dennison is a freelance PHP/MySQL developer and open-source author. He spends his summers leaning into the two-lane curves of Iowa's back roads. He also contributes to and maintains three blogs: the Linux-focused “uptime” (uptime.steidler.net), the personal and political blog “Frankly, I'd Rather Not” (steidler.net) and the running tech narrative “The CodeMode Chronicles” (steidler.net/codemode). He carries on multiple simultaneous electronic conversations at steidler@gmail.com.

Archive Index Issue Table of Contents

Advanced search

America's Army for Linux

Gary Glasscock

Issue #125, September 2004

This cross-platform offering allows players to pick from various squad positions, each with its own weapon.

America's Army Operations, a game that's more interesting than common run-and-gun games, is available for Linux. Not any old first-person shooter, *America's Army* is squad-based. This game is free as in beer.

This cross-platform offering allows players to pick from various squad positions, each with its own weapon. Weapons available are the M-16 rifle, M-203 grenade launcher, Squad Automatic Weapon (SAW) and two different sniper rifles. At around 588MB, the game takes a while to download, but believe me, it's worth the time spent. Go to www.americasarmy.com to get your copy, and while you're at it, create a free-of-charge account.

Figure 1. You never know who's around the corner.

Now that you've installed the game and signed in through the Personnel menu, your training begins. Score a 36 or higher on the Marksmanship Training, and you can take Advanced Marksmanship Training. Complete that and you have the option of being a sniper. You also can go the Airborne School, Medic Training and Special Forces Training. The Airborne and Special Forces trainings are needed to be able to play specific maps in the game. Medic training offers you another way to gain points besides killing the enemy and completing the objectives of the map.

Completing the objectives is one of two ways for your team to win a round. The other is to kill everyone on the other team. You choose. But, what if you don't want to be a team player? It's up to you really. If you insist on being a lone ranger and killing everyone, you soon will find yourself kicked off the server by your teammates and your honor (level) will be decreased. So go ahead, get the game and join me on-line. Look for -=;DarkRain;=-; I'll be waiting.

From the Editor

A Spectrum of Great Projects

Don Marti

Issue #125, September 2004

Wireless communication on Linux is much more than merely a wireless session at a café. Spread out and enjoy the rest of the radio spectrum too.

Put “wireless” and “Linux” on the cover of the same magazine, and people are going to expect 802.11b, or Wi-Fi, networking. We won't disappoint you, as our master chef, Marcel Gagné, covers some desktop tools that you can use to manage your wireless connections (page 24).

Also on the desktop, if you're tuning in to satellite radio and want to control the receiver from Linux, you're in luck. Michael J. Hammel has a friendly GUI app for you. Follow along and learn how you can modify it or write your own applications (page 78).

We got a lot of positive response to Eric Blossom's introduction to software radio in our June 2004 issue. People want a beginner's software radio project to do, so Eric generously developed and wrote up the FM receiver on page 42. Even if you missed the first article, this one will have you listening to FM stations with software. You need a signal for testing, so now corporate radio finally will be worthwhile for someone other than the three Eagles fans who don't have a copy of “Hotel California” and the one guy who wants to buy a Toyota but doesn't know where the dealer is. Once you get started, there are plenty more projects to explore in software radio, so keep in touch when you invent something.

Ian McLoughlin and Tom Scott are quietly inventing a new RF communications mode, with the help of a Linux cluster, Linux FPGA tools and more. See how a modern engineering lab works with Linux on page 36 and get some ideas.

If you're a radio ham, and a little disappointed with the lack of OS diversity in the mainstream amateur radio magazines, you're sure to enjoy Volker Schroer's intro to PSK31 under Linux on page 50. In the early days of *LJ*, we had a lot of ham readers, so write in if you want more on amateur radio.

Not that ham radio isn't plenty educational and community-building, but radio amateurs settled for a bad deal with governments when radio regulation came into effect, nailed down in the US with the Radio Act of 1927. Amateurs were forbidden to broadcast music, news or general-interest programs. Hams in the 1920s put concerts and sports events on the air but found themselves relegated to talking about—well, mostly about radios. For details, see Bill Continelli's “History of Amateur Radio” on ham-shack.com.

What could have been the first peer-to-peer media went silent. Don't let it happen to new technologies—no matter how tempting it is to want to regulate spam, viruses or “bad software”, doing so could mean we'll wake up to find that all our favorite community Web sites can talk only about Cascading Style Sheets.

Whether you simply want a control panel for Wi-Fi and better music selection via satellite radio, or you're ready to break out the soldering iron and invent a whole new application for software radio, there's plenty to learn and enjoy in this issue. Have fun and see you next month.

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

Letters

Readers sound off.

Linux for Cats

My cat Toby has kept a tight lid on my productivity until recently by demanding too much attention, but I just discovered a new way to distract him...he is fascinated by the screensaver of the whales, which lets me work on my other Linux workstation. Kudos to the creators of that screensaver.

—
Michael Ferguson

Mark Kilgard's “atlantis” is part of XScreenSaver, at jwz.org. —Ed.

Ultimate Linux Box Wish List

GNU/Linux has never been about the hardware. GNU/Linux is about FREEdom. The distro that goes on the ULB, and what a user can do with that distro, is more important than the hardware that makes up the box. An effort should be made to use as much Free Software as possible and still be able to do all the desirable things.

At a minimum, the ULB should be able to do all the i-things that a Mac will do, and do them just as easily and well. Web video, Flash, radio, etc., should be as accessible as with a Mac. It should do the basic stuff like easy hookup of scanner and printer, icon mounting and umounting of USB removable drives and hardware. Really good stereo, and home theatre (including ripping DVDs and TiVo ability) seem essential for an Ultimate Box.

And gosh, wouldn't it be nice to be able to handle personal finance or small business needs, AND be able to do federal and state taxes? Plus, since Linux is inherently a server OS, and this box is the ultimate hardware, it might as well serve half-a-dozen thin clients so the whole house, or small office, can enjoy ULB power without the expense of multiple ULB boxes.

—
Eric Skalwold

Software patents keep free systems from supporting all the same audio and video formats that proprietary ones do. But, you'll be happy to know that we used the all-free Fedora for the Ultimate Linux Box this year. —Ed.

C++ Please

I am a new subscriber and I must tell you what an extreme pleasure it is to receive your magazine. I also subscribe to two other computer-related magazines at the present, although *LJ* is by far the best of them all by light-years.

Unlike most publications, you do not talk down to the readers; rather, you expect them to be beyond hand-holding to the point where they are actually willing to try things and experiment! From my experience with Linux over the past 3–4 years—I didn't know about *LJ* until recently—it is by taking bits of information I have read and trying things out, modifying things, simply experimenting.

I went from knowing almost nothing about Linux, except the name, to being in charge of a small LAN of about 25 computers running SuSE Linux Professional 9 and Microsoft Windows 2000 Professional, along with a server and backup server both running SuSE 9 as well. I have been constantly amazed at how easy both Linux and Windows clients work together on the network, much better in fact, than when the network was entirely Windows-based. Using a Linux server has also made the network much, much more secure than was possible using only Windows.

I have noticed that most of your program listings are in languages like Python, Perl, PHP and C. Could you do an article or two on C++, perhaps on the changes that would need to be made to a C program you have listed to make it compile with g++? Or an introduction to using g++? I think many readers might be new to programming and would enjoy a brief introduction to C++.

—
Frank Robinson

Check out the June 2003 issue for a bunch of C++ articles. —Ed.

Permissions on yum.conf

I enjoyed the article about Yum [*LJ*, June 2004]. However, yum.conf is a read-only file. It would have been very helpful if Mr Bauer had told how to make the file writable. I am a newbie to Linux and I need all of the input I can get on the fine points of any process. Yours is the first Linux magazine I have purchased.

—
Steve Starr

Mick Bauer replies: I'm sorry you experienced difficulty configuring Yum. On my Fedora system, /etc/yum.conf has its permissions set to `-rw-r--r--`. In other words, it IS writable, but only by root. Nobody but root has any business editing much of anything in /etc to begin with, because it contains system-wide configuration files. Having said all that, maybe I'm overdue to write an introductory article on filesystem security. Thanks for putting the idea into my head!

Hey! You Kids! Get Out!

Because of the regular appearance of photos of children in the Letters section of *Linux Journal*, I will not be renewing my subscription. Your kids are all adorable and all geniuses, no doubt, but this is not the place.

—
Anonymous

Thank You for Promoting SPF

This is a brief note of thanks for the amazing quality of *Linux Journal*! I must relate to you that when the *LJ* magazine shows up at my house, all else stops while I browse and read the first article of interest. It seems there is always at least one amazing piece!

I must add that the recent piece on SPF [*LJ*, May 2004] is a huge contribution to the industry at large. I am sure it will have a major influence for the better on all of us who are nearly totally dependent on e-mail. I found it especially helpful in communicating with the poor souls charged with keeping our network alive.

It used to be *Byte*, but now *LJ* rules!

—
Robert W. Carter

Une Leçon de Français

I love Marcel Gagnés articles and I love *Linux Journal* and I love the care you take in editing the magazine to ensure stylistic and orthographic decency. It is with a reluctant keyboard, therefore, that I beg you to fix the French tag in Marcel's column. It should be *A vôtre santé*, not *A vôtre santé*. *Vôtre* is the adjective form; *vôtre* the pronoun. As *vôtre* here is modifying *santé*, it is an adjective. *Mille remerciements*!

Arkeia User Report

I was a little disappointed with the Arkeia review in the July 2004 issue. Although the perspective of a user new to the product is certainly valuable, a lot of us are herding systems that will stress the software a lot harder.

The number one issue for me right now is that if one scheduled backup job runs long, and the next scheduled job starts, both hang. The support response is that jobs should be scheduled further apart, but this fails to address the issue. In any serious system, there will be occasional overlaps, and the software should cope in some sensible fashion. Either canceling the second job or holding it until the hardware resources were available would be better than a hang. Also, some indication of a problem is not too much to ask.

A related problem is that the software is licensed per server, per client and per simultaneous data stream. If one has eight machines to be backed up, and four streams, the software will back up four machines at a time until all eight are completed. However, a backup job must have at least one stream assigned to it to stay in queue. Thus, when a second job starts, even if it cannot run, it steals a stream from the running backup. If the hang problem ever gets addressed, this is still potentially troubling.

The database structure used by the server to track files and tapes is stored in a filesystem directory tree that mirrors the backed-up data. The software seems to take a significant performance hit in dealing with large numbers of small directories.

Arkeia strongly recommends that non-ark methods be used to back up the root filesystem of the backup server and the Arkeia installation itself.

There are a number of issues with the GUI as well. Mr Wilder covered some of them, and his comments are well taken. Others nits include inconsistent placement of buttons with similar actions in different functions of the GUI, and the fact that the “go back” button is sometimes missing from the function bar (or that in one or two places, the function bar is missing altogether).

The log browser functionality has two frustrating quirks: in viewing the full system log, one sees a calendar month at a time. It's not at first obvious how to change months, which is accomplished by clicking on the text above the log text box naming the displayed month. Said text bears no indication that it's *hot*. It's also practically guaranteed that one backup per month will be split across two displays. Perhaps displaying the most recent 30 days or so would be more useful.

It seems that users would be better served by a GUI that wasted less programming effort on flash, in favor of solid function.

—
Dennis Boone

Dan Wilder replies: I am curious as to whether you've looked into Amanda, an open-source backup system we use at *Linux Journal*. I believe it might address some of the difficulties you mention (www.amanda.org).

There's no GUI, but as I might have implied in the article, my personal opinion is that although a GUI might be useful for selling a backup software package, it is apt to be mostly a distraction for the accomplished user. Amanda configuration is performed entirely by editing text files, and backups are managed from the command line or from cron jobs.

Amanda acts as a scheduler and indexer for dump or tar, so it'll back up nearly anything UNIX-y in nature. Backups are centrally managed from a backup server. Amanda will handle any reasonable number of clients. Environments with many different processor architectures and OSes are not a problem.

There is no native client support for Windows operating systems. SMB shares exported from Windows systems can be backed up, but some attributes may be lost and backups of some files can fail due to share violations.

Amanda has a somewhat novel way of managing backups. You specify the volumes (hosts and their partitions or directories) to be backed up, along with some other constraints, and Amanda schedules the backup levels. Total dumped data volume is balanced from backup run to backup run by mixing levels for different volumes within a run. You wind up making much better use of scarce tape space and runtime, than you could likely do with a fixed schedule of backup levels. When possible, Amanda advances lower-level volume backups to exceed your constraints.

Support is provided by a low-traffic mailing list. Although this support does not offer the almost immediate turnaround I found with Arkeia, most installation questions raised on it get addressed, so long as they're really installation problems and not arguments about the fundamental design or demands for features nobody is likely to implement.

An FTP-like command-line browser is available for doing restorations. As with some other backup software, you're vulnerable if you happen to lose the on-line tape directory. At this point I am not aware of any reasonable way to rebuild the index from tapes. On the other hand, it's easy to locate the index and to make provisions outside of Amanda to keep copies. Indexes are kept in flat files mirroring the host/volume structure of the backup schedule, having one line per dumped file or directory. It is possible this scales better than the directory-mirroring structure you describe.

Some soft failover provision is made by Amanda's use of a holding disk for backups, from which they are streamed to tape. If you provide enough disk storage, an entire backup run can proceed on schedule even when a tape drive fails or when somebody neglects to load the expected tapes. The backup is then flushed to tape after the problem is corrected.

Support for stackers may require some scripting skills, there's almost no support for backing up to anything except tapes, and your first Amanda installation can be painful, as there are lots of ducks that have to be lined up precisely, so to speak. Multiple drives cannot be configured in sequence, so that when a drive is full, the backup simply moves on to the next. Multiple backup runs cannot be written to a single tape. If two runs to the same tape drive overlap, the second simply aborts. On the other hand, there are no license limitations to fret about. Once set up correctly, about the only thing you'll ever do to Amanda is change tapes, restore files and sometimes flush a failed dump to fresh tape.

BSD-Influenced Distro: Gentoo

This e-mail is in regards to the letter “BSD, Please” in *Linux Journal*, July 2004. The reader loves BSD for its ports. I thought that I would bring up Gentoo Linux (gentoo.org), one of the fastest-growing distros, whose package management system, Portage, is based largely on BSD's ports. It is a great source-based distro.

—
Nicolas Steinberg

Watch for an article on deploying Gentoo packages in an upcoming issue. —Ed.

Photo of the Month: Mmmm, Penguin Cake

I happily submit this photo of my husband Kevin, with his Linux-inspired 25th birthday cake. He is an avid reader of your magazine and user of Linux at home and work.

Photo of the Month gets you a one-year extension for your subscription. Photos to info@linuxjournal.com. —Ed.

Erratum

The article “RTAI: Real-Time Application Interface”, which appeared in the April 2000 issue of *Linux Journal*, is missing attribution for a list of issues affecting real-time support in Linux. Attribution should go to “The RTLinux Manifesto” by Victor Yodaiken, which appeared in the proceedings of the 5th Linux Expo in 1999.

UpFront

diff -u: What's New in Kernel Development

Zack Brown

Issue #125, September 2004

A new process-accounting project, called **Enhanced Linux System Accounting (ELSA)**, has been started for Linux. Mechanisms for tracking system behavior on a process-by-process basis already exist; what ELSA proposes is to collect processes together into banks and assess them as a group. In theory, other aspects of the system also can be grouped together into banks for ongoing analysis.

The **InterMezzo** filesystem has been dropped from the 2.6 Linux kernel on the grounds that **Peter J. Braam** has stopped maintaining it, and no one else has stepped up to take his place. Peter himself acknowledges that InterMezzo can be maintained adequately as an independent module if anyone cares to do it and has helped smooth over **Linus Torvalds'** decision to drop it. InterMezzo, intended for large clusters, server replication, mobile computing and other high-availability projects, appears not to have been maintained seriously since 2002, and its mailing lists have been dead for some time. Unless people become passionate enough about it to pick it up themselves, it is doubtful the filesystem will return to the mainline Linux kernel tree.

Although 2.6 continues to stabilize, the 2.4 kernel continues to be most users' first choice for their systems, and a lot of feature swapping takes place between the 2.4 and 2.6 trees. **Marcelo Tosatti**, the 2.4 maintainer, has been trying to slow down the adoption of new features in 2.4, hoping that users needing those features would migrate to 2.6. But until Linux 2.6 becomes firmly established, he has found himself under pressure to accept features like **Serial**

ATA (SATA) in spite of the fact that stable kernel series are not supposed to include new features that might destabilize the system. During all of this, Marcelo also has been hard at work on some 2.6 features, such as some recent per-user resource limits he submitted in April 2004 to **Andrew Morton**, the 2.6 maintainer (along with Linus until 2.7 forks off).

These changes were inspired by the discovery that users could execute a **denial-of-service (DoS)** attack by creating too many pending signals, among other things. Unfortunately, although these fixes certainly are needed for security reasons, it is possible that the various **shells** (bash, csh and others) may need to be updated as well, in order to take advantage of the new limits. This poses some compatibility problems, making it difficult for users to upgrade only the kernel if they want, without at the same time upgrading other software as well.

Every once in a while someone gets a really weird idea and then goes ahead and implements it. Some would say Linux is itself an example of this, but another example is the recent attempt by **Sergiy Lozovsky** to embed a **LISP** interpreter into the 2.6 Linux kernel. The purpose was to allow system administrators to control the security policies of their systems using a high-level language, so they wouldn't have to alter the kernel source code and recompile. Sergiy's LISP interpreter was much simpler and smaller than Common LISP, taking up much less RAM and running in a virtual machine that would protect the system from any LISP bugs introduced by careless system administrators. The worst that would happen is that their security policy wouldn't work. There is virtually no chance that this feature will be accepted into the kernel proper, for several reasons. First, the LISP language probably is not going to be a favorite with system administrators who would want such an interpreter, as they probably would prefer something more shell-like. And second, because virtually the same thing could be implemented in user space with no need of embedding the entire interpreter in the kernel at all. This also would obviate the need for a virtual machine to protect the system from system administrator programming errors. In spite of any objections, however, an embedded LISP interpreter in the kernel is fun to play with and might be the sort of thing to be maintained for a long time outside of the kernel, by enthusiastic folks who simply like the weirdness of it all (or who really think it's a good idea).

The entire 2.6 development process is somewhat unusual, even for Linux. First, Linus Torvalds seemed to hand off development to Andrew Morton, but then it turned out they would continue to work in tandem. Andrew continued to produce kernel versions with names like 2.6.6-rc1-mm2, the -mm being a holdover from when most of his patches had to do with memory management. Recently he also has begun to release a -mc series, standing for merge candidate, that is, sets of patches to be merged with Linus' tree. While doing this, he still has released -mm versions, although presumably these also are intended to be merged with Linus' tree. From the user perspective, everything is still clear and calm, because the 2.6.5, 2.6.6, 2.6.7 and so on, releases all still come out and are housed in known locations. But, unlike the ongoing 2.4 and 2.0 (and theoretically 2.2 as well, although its maintainer **Alan Cox** has been fairly quiet lately) development, 2.6 maintenance has seemed to bounce around a bit more than the others.

LJ Index—September 2004

- 1. Worldwide server sales, in billions of dollars, for the first quarter of 2004: 11.5
- 2. Year-to-year percentage revenue growth in server sales: 7.3
- 3. Year-over-year server unit shipment growth percentage: 22.4
- 4. UNIX year-over-year server revenue percentage decline: 3
- 5. Windows server revenue percentage growth: 16.4
- 6. Windows server shipment percentage growth: 26.5
- 7. Linux server sales percentage growth: 56.9
- 8. Linux server shipment percentage growth: 46.4
- 9. Number of consecutive quarters of double-digit Linux server revenue growth: 7
- 10. Number of billions of dollars in worldwide quarterly factory revenue approached by Linux: 1
- 11. Number of consecutive quarters in which Linux servers sales have exceeded $900 million: 2
- 12. Peak number of simultaneous listeners to the Linux-powered Radio Paradise: 5,647
- 13. Linux market opportunities in China, in millions of dollars by 2008: 41.9
- 14. Linux market opportunities in Japan, in millions of dollars by 2007: 105
- 15. Thousands of Linux-certified practitioners at IBM: 3
- 16. Thousands of people at IBM that have “some kind of Linux exposure”: 12
- 17. IBM revenue percentage growth in 2004 over 2003: 20.5
- 18. Number of Oracle servers at city of Bergen (Norway) moving to Linux: 8
- 19. Number of Windows application servers being consolidated to Linux servers in Bergen: 100

- 20. Approximate number of Linux servers to which Bergen's app servers are being consolidated on IBM eServer BladeCenters running Linux: 20

- 1–11, 17: International Data Corp.
- 12: Radio Paradise
- 13–14: International Data Corp. via InternetNews
- 15, 16: *Times of India*
- 18–20: ZDNet

They Said It

Linux servers are playing increasingly important roles in IT customers' computing infrastructure. They are taking on enterprise workloads, now that more ISV applications are available for both technical and commercial workloads on the Linux server platform.

—Jean S Bozman, research vice president, Global Enterprise Server Solutions, International Data Corp. (www.idc.com/getdoc.jsp?containerId=pr2004_05_25_194643)

The weather scares us a bit, but on the other hand we know a lot of people up there, and they all seem happy. And it's bound to be less crazy than Silicon Valley.

—Linus Torvalds, on moving from Silicon Valley to Portland, Oregon (management.silicon.com/careers/0,39024671,39121320,00.htm)

In addition to the IT-based benefits from migrating to Linux, we attain a business model that doesn't tie us to a single vendor's solution architecture. By migrating to Linux, the City of Bergen has a business model that is open and democratic and, we believe, that will ensure a greater degree of freedom of choice, more efficient operation and major cost savings that will benefit the citizens.

—Janicke Runshaug Foss, CIO of the City of Bergen, Norway (news.zdnet.co.uk/0,39020330,39157677,00.htm)

The new API is HTML, and the new winners in the application development marketplace will be the people who can make HTML sing.

—Joel Spolsky (www.joelonsoftware.com/articles/APIWar.html)

Archive Index Issue Table of Contents

Advanced search

On the Web

A Better Wireless Experience

Heather Mead

Issue #125, September 2004

Our Web authors are sharing the details of their projects aimed at creating and improving Linux wireless technology.

A while ago, I had conversations with two of my least technically inclined friends, both of whom brought up the topic of Wi-Fi within minutes. In fact, one friend, a dancer and choreographer, called me specifically to ask what he could do with “the Wi-Fi thing in my laptop”. He'd bought a shiny new laptop on the recommendation of another friend and now was confessing to me that although he knew what Wi-Fi was theoretically, he wasn't sure what he could do with it.

Fast forward to the recent 4th of July weekend, when I spent time with these friends and found out they've both figured out a whole lot they can do with wireless. The choreographer told me he is working on-line with dancers on a new production. He's also collaborating with the cellist writing music for the piece. The cellist lives in San Diego, and my friend is doing most of this work while running around Seattle and hopping on any of the many hot spots he locates thanks to SeattleWireless.net. And, although certainly not the most important function that wireless technology ever served, without cell phones with IM features, my other friend and I never would have located each other among the 30,000+ people present at the fireworks display the evening of July 4th.

For those of you who've mastered IM and who are able to locate a hot spot blindfolded, the topic of this issue is Everything Wireless, and the *LJ* Web site offers even more project news, advice and how-tos. Take, for example, “Getting On-line Anywhere with Bluetooth and GPRS”, by Sreekrishan Venkiteswara (www.linuxjournal.com/article/7525). This article offers an overview of how BlueZ and General Packet Radio Service (GPRS) work and explains how they can be used when designing embedded Linux products, including cell phones and PDAs. Venkiteswara writes, “Because Bluetooth supports device inquiry and service discovery, the Bluetooth devices automatically can use a nearby cell phone for Internet connectivity, without cumbersome configuration of such details as physical addresses.”

In “A Linux-Based Implementation of Mobility Using SIP” (www.linuxjournal.com/article/7380), the developers explain how they created the “first open-source SIP implementation that supports mobility” by adding mobility capabilities to Vovida's VOCAL VoIP software. They write, “The software supports seamless SIP signaling when the user moves from network to network, as well as automatic handoff if the user is in the middle of a call while changing networks.”

On the end-user side, for those of you struggling with wireless connectivity for your laptop, be sure to check out the latest installment of Doc Searls' adventures with his new IBM T40 laptop, “The Laptopia Odyssey, Part 2” (www.linuxjournal.com/article/7636). A big part of this stage in creating Doc's ultimate mobile workstation is getting Wi-Fi to work as well and as consistently as he needs. Things improved once he switched to SuSE Professional 9.1 and made a few other changes, but it's still not perfect. He writes, “Kismet seems to work, so far. It sees 802.11b and b/g access points, two of each (three here, one next door). I still need to figure out how to get the ThinkPad to switch easily from one to another.” Doc also writes about using FireWire and the browser and office applications he likes best for his laptop.

If you're doing something new and interesting with wireless or if you've improved your wireless experience, why not share it with the rest of the Open Source community? Send me an article proposal at info@linuxjournal.com.

Heather Mead is senior editor of *Linux Journal*.

Best of Technical Support

Our experts answer your technical questions.

Wireless Settings in Fedora

I've got my wireless NIC working in Fedora Core 2, and I'm able to communicate with the network after I configure with iwconfig. But, after I reboot, all the settings are lost, and I have to enter all the info again. Is there something I'm not doing, or is this problem something else?

—
Weoh G

weoh3@cox.net

You should use the system configuration tools to set up the device: Task Bar→System Settings→Network.

—
Christopher Wingert

cwingert@qualcomm.com

The wireless-tools command-line utilities (iwconfig, iwspy, iwpriv) do not stick through a reboot, as you have experienced. The thing to do is to implement the settings through the appropriate configuration file. In the case of Fedora, you probably can get what you need set permanently, by adding wireless-specific options to your ifcfg file (/etc/sysconfig/network-scripts/ifcfg-ethX, where ethX is the Ethernet device name assigned to your wireless card). You simply need to place the additional wireless-related options into that file. For example, on my laptop, which connects over an 802.11b interface to the router/firewall/dhcp server I hacked together in my house, the file is very simple:

```
DEVICE=eth1
BOOTPROTO=dhcp
ONBOOT=no
MODE=Ad-Hoc
```

```
    CHANNEL=1
    KEY=XXXXXXXXXX
```

The MODE, CHANNEL and KEY options were all I needed for my setup; yours are sure to be different, but anything you can do via the command line should be available as an option in the file. For a list of all available settings, take a look at /etc/sysconfig/network-scripts/ifup-wireless.

—
Timothy Hamlin

thamlin@nmt.edu

The following page contains specific instructions on how to set wireless networking on the Fedora Core Linux distribution: www.siliconvalleyccie.com/linux-hn/wmp11-linux.htm.

—
Felipe Barousse Boué

fbarousse@piensa.com

Hunting for a Single-Board Computer

In *Embedded Linux Journal*, issue 9, there was an article titled "Update on Single-Board Computers" that contained a picture captioned "An EBX Form Factor PowerPC-based SBC from Motorola". What I wanted to know is which company manufactures such a board—an EBX Form Factor PowerPC-based SBC.

—
Chirag Cheema

cscheema@tpcsed.com

With some research, I turned up at least two options: www.embeddedplanet.com/products/rpxc.asp and www.acttechnico.com/mot-ebx-motorola.html.

—
Chad Robinson

chad@lucubration.com

Try Ampro: www.ampro.com.

—
Felipe Barousse Boué

fbarousse@piensa.com

Getting System Info

We have a system-defined structure called MACHINE_STATIC to find machine architecture, including the IP address, processor speed, OS and so on. I need a similar structure on Linux to extract machine architectures. Do you know if one is available?

—
Rajesh Kumar Patnaik

rajeshp_80@yahoo.co.in

Not a structure, per se, although certain ioctl() calls can provide many of these details. But you should be looking toward the new mechanisms for extracting this information—procfs and the upcoming sysfs. Note that sysfs is new, and not many systems implement it yet. The files in /proc, when read, will provide you with many relevant system data elements. For example, /proc/cpuinfo provides CPU data, /proc/net/route provides network routes (IP addresses are in hex) and /proc/version provides the kernel version.

—
Chad Robinson

chad@lucubration.com

Tools for Disk Repair?

Does anyone know what tools are used in Red Hat 8 to find and repair disk and file problems—scandisk, scan registry?

—
moo

moochoo_86@hotmail.com

These tools are not specific to a distribution; they are specific to a filesystem type. Like other distributions, Red Hat supports a number of filesystems, including ext2, ext3, xfs, ReiserFS and many others. The “see also” section in the man page for fsck lists the file checking utilities for the most common filesystems.

Recovery options available is a key selection criteria for choosing a filesystem in the first place. New Linux users are often baffled by the array of choices and seek guidance regarding filesystem selection. I always recommend that the availability of recovery tools be part of this evaluation.

—
Chad Robinson

chad@lucubration.com

GUI Database Front End?

I am using Red Hat 9 and SuSE 9 and really like both of them as well as OpenOffice.org software. However, I am puzzled by the fact that I cannot locate any quality database programs similar to Microsoft Access. I would like to build a database that will work on a small intranet consisting of three or four machines.

—
Todd Hoover

thooveril@aol.com

If you want an open-source product, take a look at Rekall, which was recently open-sourced by TheKompany.com. For a proprietary but inexpensive alternative, take a look at Adabas D, part of Sun's StarOffice. Both products are younger than Access and do not implement every function provided by it, so if you are looking to migrate existing databases, complex applications may not be 1:1 portable. However, both products do provide significant functionality and may be suitable for your environment, especially if your needs are more focused on new applications.

—
Chad Robinson

chad@lucubration.com

OpenOffice.org can work with almost any ODBC or JDBC database, including PostgreSQL, MySQL and even Access. Go to Tools→Options→Data Sources to set up a connection.

—
Bruce Byfield

bbyfield@axionet.com

I suspect you are interested in the MS Access-like user interface. For that there are many front ends for PostgreSQL, going from pgaccess, www.pgaccess.org, to OpenOffice.org's database tools that interface to PostgreSQL and other RDBMS. As a side note, tons of companies use open-source databases, from small companies with small Web sites to Fortune 500-sized companies; one special case may be the dot ORG registry that relies on PostgreSQL for managing the overall .org domain on the Internet.

—
Felipe Barousse Boué

fbarousse@piensa.com

If your application's interface is a simple form-based one, consider doing it as a Web application instead. You won't have to maintain software on the client, you won't have to learn different tools for internal and customer-facing applications and users already know how to use a browser.

—
Don Marti

info@linuxjournal.com

Active Directory, Meet PAM

My company uses a mixture of Linux (Red Hat AS 2.1) and Microsoft Windows servers. I want to set up a central authentication server for both platforms. We use Active Directory, and it has been suggested that we might be able to use AD for Linux. Is it possible to use AD as a central authentication server for Linux, and what's the best way to do it? Or, would we be better off with a Kerberos or LDAP server?

—
Paul

pammann@execomm.net

In Linux, explore a security layer known as Pluggable Authentication Modules, or PAM. This allows you to authenticate users logging in locally against your AD servers.

For Apache, take a look into mod_auth_ldap, which allows you to do the same. Alternatively, you can use mod_auth_pam to instruct Apache to share your Linux server's PAM deployment. This is worthwhile if you do intend to have multiple applications use this data, because it will reduce your setup time. However, if Apache is your only application (and this is not uncommon) you might want to stick with a direct mod_auth_ldap configuration, as there are fewer steps involved in its configuration.

A third common configuration is Samba, and there is a good HOWTO for this configuration at this site: www.netadmintools.com/art172.html.

—
Chad Robinson

chad@lucubration.com

In the long run, you'll likely be happier with a cross-platform “single sign-on” plan based on LDAP, as described in “OpenLDAP Everywhere” in the December 2002 issue of *Linux Journal*. It works on Linux and Microsoft Windows and is more flexible and future-proof than a vendor-specific solution. But if you do plan to authenticate against Microsoft Active Directory using Kerberos and PAM, Tim Fredrick has some helpful notes at acd.ucar.edu/~fredrick/linux/ad.html.

—
Don Marti

info@linuxjournal.com

Dear Bill—I just saw your question in *Linux Journal* [July 2004] about setting up a Red Hat 9 and Windows 2003 server on the same PC, and I wanted to add something to the replies you already received. Namely, 128MB of RAM is too small for a Red Hat 9 or Fedora Core installation, if you use the GNOME desktop (the standard choice). You need at least 256MB of RAM for this. If you do not want to add memory (although it's pretty cheap these days), and you want to stay with Red Hat, then it is necessary to set up an alternative, lightweight window manager, like IceWM. This would involve extra trouble for a beginner on Red Hat, where IceWM does not come automatically installed.

SuSE 9.1 is a distribution that makes it easy to choose between KDE or GNOME (with heavy RAM requirements) or IceWM or other lightweight window managers. You can buy SuSE 9.1 directly from the SuSE Web site. They have a Personal edition for about $40, but it doesn't include server software. The Professional edition for about $90 does include it.

—
Robert Littlejohn

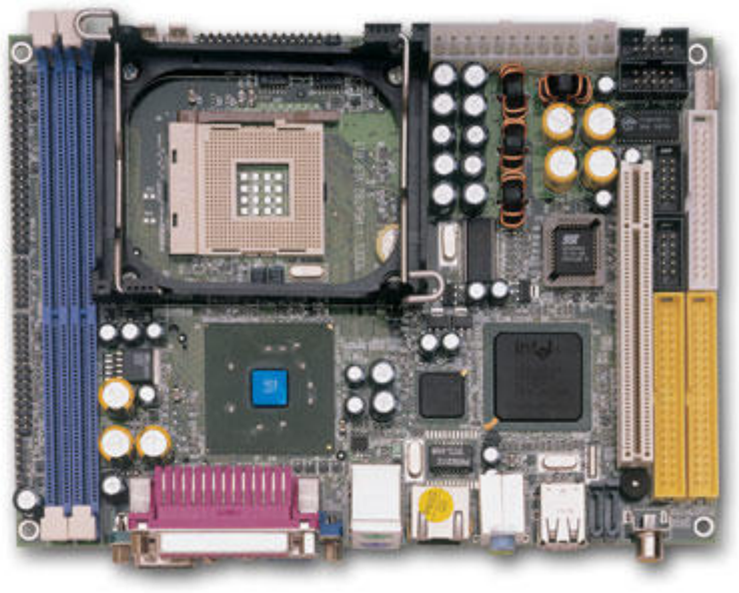
New Products

Redfone Enterprise Communication System, HS-4703 and HS-4702 Embedded Engine Boards, WaveStore 703 and more.

HS-4703 and HS-4702 Embedded Engine Boards

Boser Technology announced the release of two embedded engine boards using the Intel 82865GV chipset. The HS-4703 and HS-4702 boards offer dual-channel DDR 400 main memory, an 800MHz system bus, eight USB 2.0 connectors, four COM connectors, a PCI ATA/33/66/100 IDE controller, an AC'97 3-D audio controller and a Serial ATA controller. In addition, both boards come with four IDE drives supporting ATA/33/66/100, a 48-bit dual-channel LVDS panel interface and giga LAN. The HS-4702 comes with TV-out, supporting PAL or NTSC TV.

Boser Technology, USA, 453 Ravendule Drive, #F, Mountain View, California 94043, 650-967-3898, www.boser.com.tw.



Redfone Enterprise Communication System

Redfone Enterprise Communication System (ECS) is a telephone system for connecting remote branch offices, call centers, remote agents and mobile employees. Designed for small to mid-sized businesses looking to replace PBX phone systems, ECS is built with Linux, Asterix open-source PBX software, off-the-shelf hardware and basic telephony components. ECS offers basic phone features, plus VoIP functionality, a Web interface to voice mail, conference calling, local and remote call agents and multiple mail folders. In addition, ECS enables voice mail and faxes to be sent as e-mail attachments that can be accessed by wireless devices, laptops, desktops and PDAs from any location.

Redfone Communications, Inc., 14380 SW 139th Court, Miami, Florida 33186, 786-544-1180, www.red-fone.com.

WaveStore 703

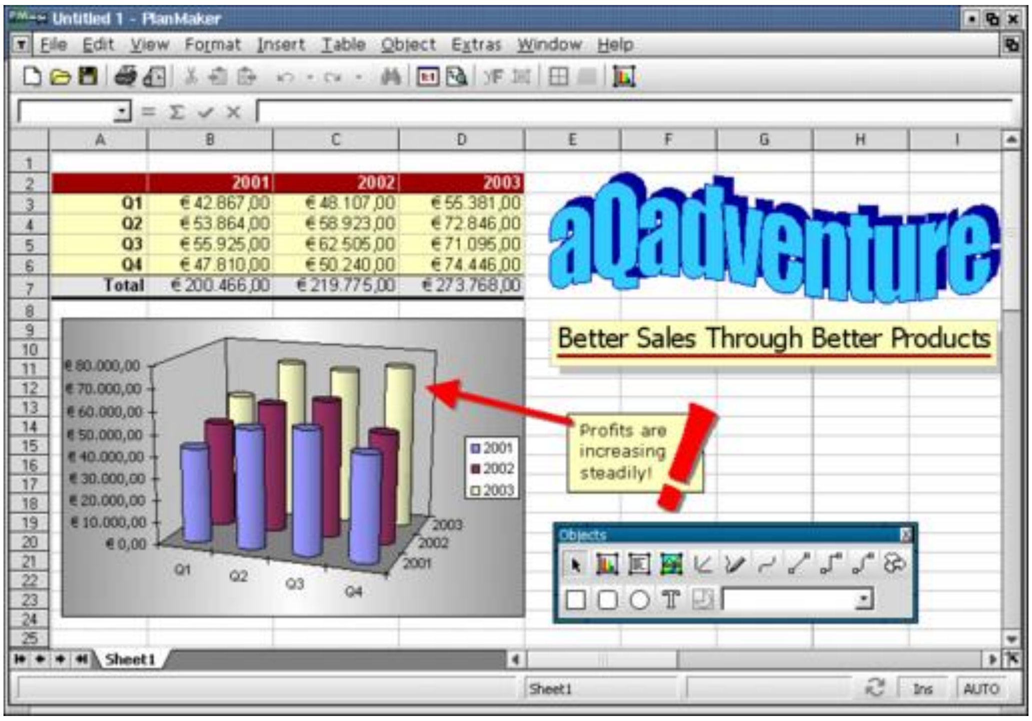
The WaveStore 703 is a signal recorder/playback unit designed as a desktop cube for work environments with limited space or for applications that require a transportable unit. The WaveStore 703 offers multiband signal recording/playback, a baseband or 70MHz IF analog interface, bandwidth programmable to 8MHz, tunable center frequency, 500GB storage capacity, continuous playback looping capabilities, DVD-R/W for data archiving and retrieval and optional external trigger signals. The cube contains a Pentium-based host computer, a Red River PCI transceiver card and complete suite of user-interface software.

Red River, 797 North Grove Road, Suite 101, Richardson, Texas 75081, 972-671-9570, www.red-river.com.

PlanMaker 2004

PlanMaker 2004 is spreadsheet software that operates on various OSes, including Windows and Linux, and offers the same feature set on all platforms. It also can operate on pocket and handheld PCs, as it requires limited RAM space. Over 320 calculation functions are built-in and cover areas such as date and time calculations, mathematics, statistics and data analysis. PlanMaker also offers features for preparing presentations, including stylesheets, various formatting options, an AutoShapes drawing module and freehand drawing. Charts and graphs can be created from any of the 70 different types of 2-D and 3-D charts that PlanMaker supports. PlanMaker also reads and writes Microsoft Excel files—Excel 5.0, 95, 97, 2000, XP, Excel 2003 and Excel for Apple Macintosh—for easy and safe file exchange with Microsoft Excel users.

SoftMaker Software GmbH, Kronacher Str. 7, D-90427 Nuernberg, Germany, info@softmaker.de, www.softmaker.com.

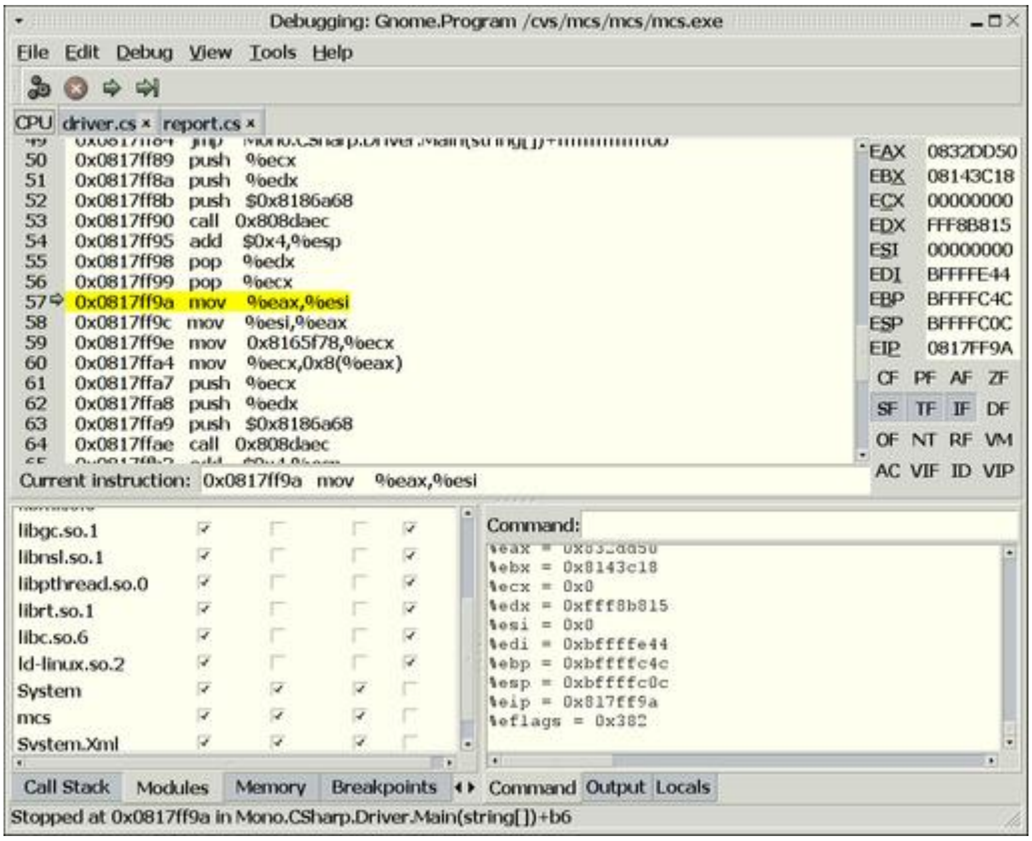

Untitled 1 - PlanMaker
File Edit View Format Insert Table Object Extras Window Help
aQadventure
Better Sales Through Better Products
Profits are increasing steadily!
Objects
Sheet1
Ins
AUTO

	2001	2002	2003
Q1	€ 42.867,00	€ 48.107,00	€ 55.381,00
Q2	€ 53.864,00	€ 58.923,00	€ 72.846,00
Q3	€ 55.925,00	€ 62.505,00	€ 71.095,00
Q4	€ 47.810,00	€ 50.240,00	€ 74.446,00
Total	€ 200.466,00	€ 219.775,00	€ 273.768,00

Mono 1.0

Mono 1.0, a community project sponsored by Novell, is an open-source development platform based on the .NET framework that allows software developers to build Linux and cross-platform applications. Mono 1.0 provides APIs and tools for developing Web services and client and server side applications that can be deployed on various platforms, including Solaris, Mac OS X, Windows NT/XP and various UNIX/Linux systems. Mono 1.0 uses the GTK# GUI programming library, so developers can target various platforms with a single code base from any of the Mono-compatible programming languages, such as Visual Basic, Python, JScript and Java. In addition, a new Web site, www.mono-project.com, has been launched with tools, resources and other information for Mono developers. Mono 1.0 can be downloaded at www.mono-project.com/downloads/index.html.

Novell Corporation, 404 Wyman, Suite 500, Waltham, Massachusetts 02451, 781-464-8000, www.novell.com/linux.


Debugging: Gnome.Program /cvs/mcs/mcs/mcs.exe
File Edit Debug View Tools Help
CPU driver.cs × report.cs ×

```
0x0817ff89  push  %ecx
0x0817ff8a  push  %edx
0x0817ff8b  push  $0x8186a68
0x0817ff90  call  0x808daec
0x0817ff95  add   $0x4,%esp
0x0817ff98  pop   %edx
0x0817ff99  pop   %ecx
0x0817ff9a  mov   %eax,%esi
0x0817ff9c  mov   %esi,%eax
0x0817ff9e  mov   0x8165f78,%ecx
0x0817ffa4  mov   %ecx,0x8(%eax)
0x0817ffa7  push  %ecx
0x0817ffa8  push  %edx
0x0817ffa9  push  $0x8186a68
0x0817ffae  call  0x808daec
```

```
EAX  0832DD50
EBX  08143C18
ECX  00000000
EDX  FFF8B815
ESI  00000000
EDI  BFFFFE44
EBP  BFFFFC4C
ESP  BFFFFC0C
EIP  0817FF9A
CF  PF  AF  ZF
SF  TF  IF  DF
OF  NT  RF  VM
AC  VIF ID  VIP
```

Current instruction: 0x0817ff9a mov %eax,%esi

libgc.so.1	✓			✓
libnsl.so.1	✓			✓
libpthread.so.0	✓			✓
librt.so.1	✓			✓
libc.so.6	✓			✓
ld-linux.so.2	✓			✓
System	✓	✓	✓	
mcs	✓	✓	✓	
System.Xml	✓	✓	✓	

Command:

```
%ebx = 0x8143c18
%ecx = 0x0
%edx = 0xfff8b815
%esi = 0x0
%edi = 0xbffffe44
%ebp = 0xbffffc4c
%esp = 0xbffffc0c
%eip = 0x817ff9a
%eflags = 0x382
```

Stopped at 0x0817ff9a in Mono.CSharp.Driver.Main(string[])+b6

L-Server

Linux Media Arts announced the availability of the L-Server, a 64-bit Linux digital disk recording (DDR) system. The L-Server employs AMD dual Opteron 64-bit technology, dual DDR memory, 1TB of storage, an AJA HD capture/playback card and eight-channel embedded AES/EBU audio. It has Serial RS232/422 control ports with Sony protocol for external deck control. It is available in an LMA mini-tower or 3U LMA rackmount configuration. Its open architecture supports any plugin, or users can build their own. The L-Server SDK allows developers to build plugins, codecs and special effects that can be dropped onto the video track timeline. Users can operate the L-Server and/or remote tape deck from a control panel or from a browser window. L-Server can capture directly from all HD VTRs and various HD cameras. Graphics can be shared across platforms using the standard NFS/SMB filesystem.

Linux Media Arts, Inc., 10442A Rockport Circle, Reno, Nevada 89521, 775-787-3768, www.lmahd.com.

Archive Index Issue Table of Contents

Advanced search