OpenWrt | Roku DVP | wview | iRobot Create | Handbrake | Player Project

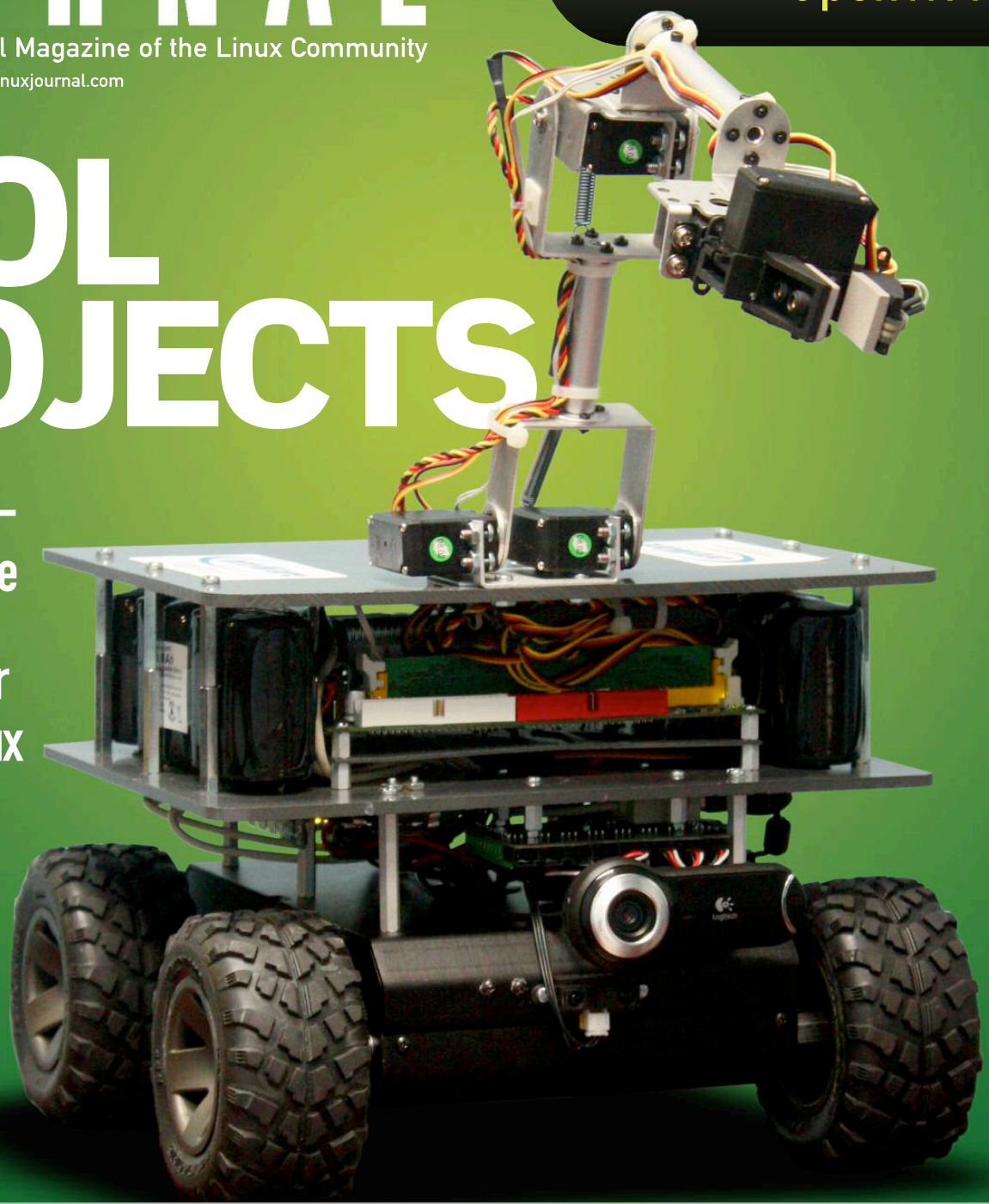
LINUX JOURNAL

Since 1994: The Original Magazine of the Linux Community
2010 | Supplement Issue | www.linuxjournal.com

Connect Rock Band Instruments to Your Linux Machine

Build Custom Firmware with OpenWrt

COOL PROJECTS



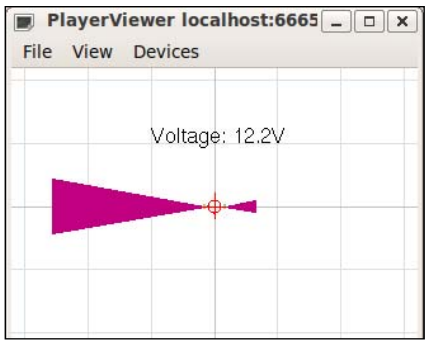
» Build Robot-Control Software

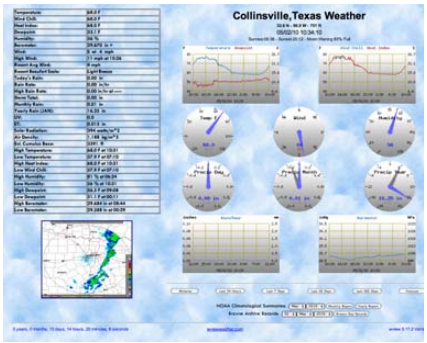
» Control Your Fridge with Linux

» Set Up a Weather Station and Publish Your Data with wview

HOW-TO: Handbreak Video Conversion

CONTENTS SPECIAL ISSUE: COOL PROJECTS





Linux Journal 2011 Lineup

JANUARY
System Administration

FEBRUARY
Desktop

MARCH
Linux in Your Pocket

APRIL
Web Development

MAY
Security

JUNE
Embedded

JULY
Cool Projects

AUGUST
Community

SEPTEMBER
Programming

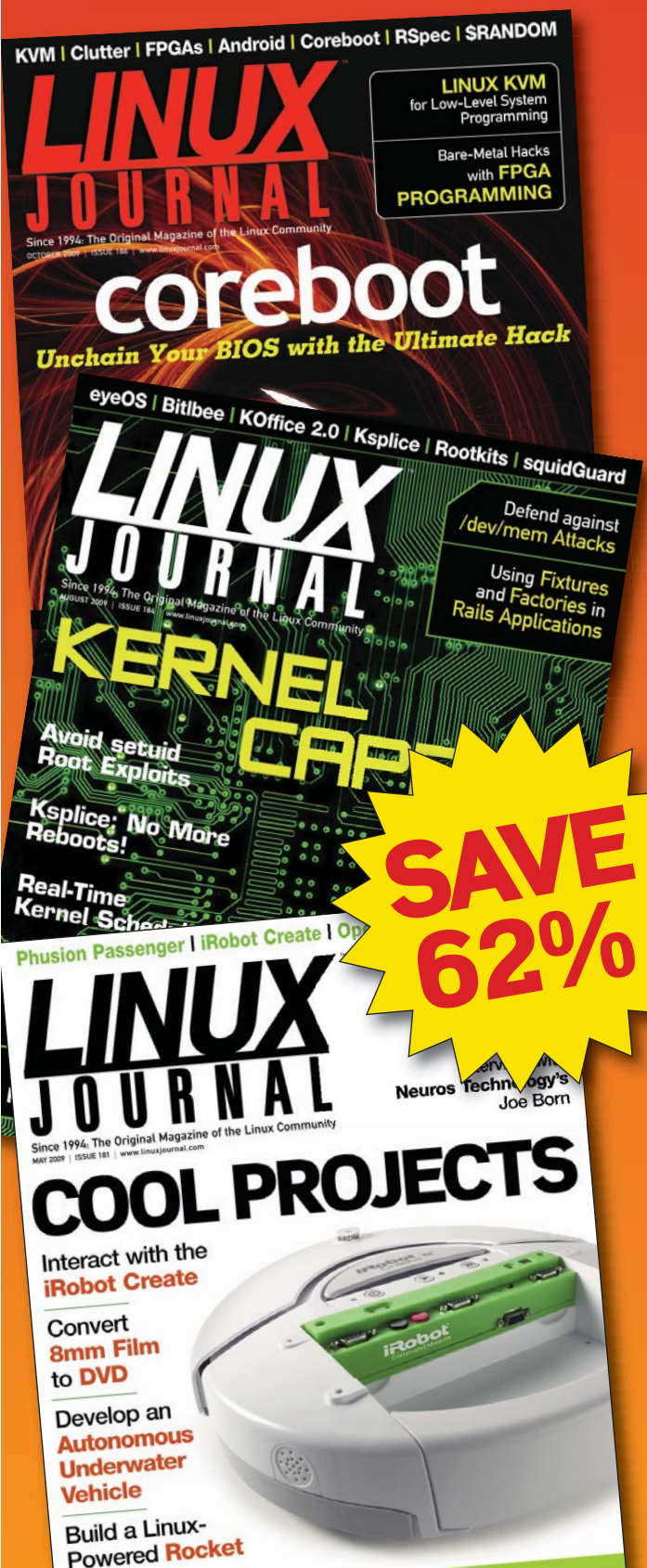
OCTOBER
Networking

NOVEMBER
Hack This

DECEMBER
Readers' Choice

USPS *LINUX JOURNAL* (ISSN 1075-3583) (USPS 12854) is published monthly by Belltown Media, Inc., 2121 Sage Road, Ste. 310, Houston, TX 77056 USA. Periodicals postage paid at Houston, Texas and at additional mailing offices. Cover price is $5.99 US. Subscription rate is $29.50/year in the United States, $39.50 in Canada and Mexico, $69.50 elsewhere. POSTMASTER: Please send address changes to *Linux Journal*, PO Box 16476, North Hollywood, CA 91615. Subscriptions start with the next issue. Canada Post: Publications Mail Agreement #41549519. Canada Returns to be sent to Pitney Bowes, P.O. Box 25542, London, ON N6C 6B2

If You Use Linux, You Should Be Reading LINUX JOURNAL™



SAVE 62%

- In-depth information providing a full 360-degree look at featured topics relating to Linux
- Tools, tips and tricks you will use today as well as relevant information for the future
- Advice and inspiration for getting the most out of your Linux system
- Instructional how-tos will save you time and money

Get *Linux Journal* delivered to your door monthly for 1 year for only $29.50! Plus, you will receive a free gift with your subscription.

SUBSCRIBE NOW AT:

WWW.LINUXJOURNAL.COM/SUBSCRIBE

Offer valid in US only. Newsstand price per issue is $5.99 USD; Canada/Mexico annual price is $39.50 USD; International annual price is $69.50. Free gift valued at $5.99. Prepaid in US funds. First issue will arrive in 4-6 weeks. Sign up for, renew, or manage your subscription on-line, www.linuxjournal.com/subscribe.

LINUX JOURNAL

At Your Service

MAGAZINE

PRINT SUBSCRIPTIONS: Renewing your subscription, changing your address, paying your invoice, viewing your account details or other subscription inquiries can instantly be done on-line, **www.linuxjournal.com/subs**. Alternatively, within the U.S. and Canada, you may call us toll-free 1-888-66-LINUX (54689), or internationally +1-818-487-2089. E-mail us at subs@linuxjournal.com or reach us via postal mail, Linux Journal, PO Box 16476, North Hollywood, CA 91615-9911 USA. Please remember to include your complete name and address when contacting us.

DIGITAL SUBSCRIPTIONS: Digital subscriptions of *Linux Journal* are now available and delivered as PDFs anywhere in the world for one low cost. Visit **www.linuxjournal.com/digital** for more information or use the contact information above for any digital magazine customer service inquiries.

LETTERS TO THE EDITOR: We welcome your letters and encourage you to submit them at **www.linuxjournal.com/contact** or mail them to Linux Journal, PO Box 980985, Houston, TX 77098 USA. Letters may be edited for space and clarity.

WRITING FOR US: We always are looking for contributed articles, tutorials and real-world stories for the magazine. An author's guide, a list of topics and due dates can be found on-line, **www.linuxjournal.com/author**.

ADVERTISING: *Linux Journal* is a great resource for readers and advertisers alike. Request a media kit, view our current editorial calendar and advertising due dates, or learn more about other advertising and marketing opportunities by visiting us on-line, **www.linuxjournal.com/advertising**. Contact us directly for further information, ads@linuxjournal.com or +1 713-344-1956 ext. 2.

ON-LINE

WEB SITE: Read exclusive on-line-only content on *Linux Journal*'s Web site, **www.linuxjournal.com**. Also, select articles from the print magazine are available on-line. Magazine subscribers, digital or print, receive full access to issue archives; please contact Customer Service for further information, subs@linuxjournal.com.

FREE e-NEWSLETTERS: Each week, *Linux Journal* editors will tell you what's hot in the world of Linux. Receive late-breaking news, technical tips and tricks, and links to in-depth stories featured on **www.linuxjournal.com**. Subscribe for free today, **www.linuxjournal.com/enewsletters**.

LINUX JOURNAL

Executive Editor	Jill Franklin jill@linuxjournal.com
Senior Editor	Doc Searls doc@linuxjournal.com
Associate Editor	Shawn Powers shawn@linuxjournal.com
Associate Editor	Mitch Frazier mitch@linuxjournal.com
Art Director	Garrick Antikajian garrick@linuxjournal.com
Products Editor	James Gray newproducts@linuxjournal.com
Editor Emeritus	Don Marti dmarti@linuxjournal.com
Technical Editor	Michael Baxter mab@cruzio.com
Senior Columnist	Reuven Lerner reuven@lerner.co.il
Security Editor	Mick Bauer mick@visi.com
Hack Editor	Kyle Rankin lj@greenfly.net
Virtual Editor	Bill Childers bill.childers@linuxjournal.com

Contributing Editors

Ibrahim Haddad • Robert Love • Zack Brown • Dave Phillips • Marco Fioretti • Ludovic Marcotte
Paul Barry • Paul McKenney • Dave Taylor • Dirk Elmendorf • Justin Ryan

Proofreader Geri Gale

Publisher	Carlie Fairchild publisher@linuxjournal.com
General Manager	Rebecca Cassity rebecca@linuxjournal.com
Senior Sales Manager	Joseph Krack joseph@linuxjournal.com
Associate Publisher	Mark Irgang mark@linuxjournal.com
Webmistress	Katherine Druckman webmistress@linuxjournal.com
Accountant	Candy Beauchamp acct@linuxjournal.com

Linux Journal **is published by, and is a registered trade name of, Belltown Media, Inc.**
PO Box 980985, Houston, TX 77098 USA

Advertising
E-MAIL: ads@linuxjournal.com
URL: www.linuxjournal.com/advertising
PHONE: +1 713-344-1956 ext. 2

Subscriptions
E-MAIL: subs@linuxjournal.com
URL: www.linuxjournal.com/subscribe
PHONE: +1 818-487-2089
FAX: +1 818-487-4550
TOLL-FREE: 1-888-66-LINUX
MAIL: PO Box 16476, North Hollywood, CA 91615-9911 USA
Please allow 4–6 weeks for processing address changes and orders
PRINTED IN USA

SHAWN POWERS

Cool Projects

I once built a fusion-powered, laser-sighted, stealth-enabled death ray. I never had enough LEGOs to build the complementary plasma sword, but rest assured, my quest to take over the world never was short on plastic-building-block weaponry. I also was quite skilled at spaceship building, but that's another story altogether. My point is simply this: geeks like to build things. Whether you started with LEGOs and action figures or your inner geek didn't emerge until you were old enough to solder, chances are, you like to build things. And, that's what this supplemental issue is all about.

Anyone planning a world takeover needs a team of henchmen. Whether you want to build a robot from scratch using the Player Project (Kevin Sikorski has a how-to) or build little iRobot minions to do your bidding (check out Zach Banks' article for more info), robots are cool regardless of design specs.

If your robots are wireless (and really, I recommend against robots with extension cords), you might want to hack a wireless router to control them remotely. Luckily, this issue shows how to hack a router using OpenWrt. Mike Petullo demonstrates how to route Internet traffic any way you like with custom open-source firmware. Granted, Mike doesn't specifically mention taking over the world, but his intent seems clear to us.

So, once you've taken over the world and held its natives hostage with your Linux-powered robotic minions, the next logical thing to do is control the weather. Oh sure, you could solve the entropy of the chaotic universe and control weather patterns by bending the butterfly effect to your whims, but that's a lot of math. We recommend starting small by controlling the humidity. Jeffrey Ramsey explains how to manage humidity using an embedded-Linux system. Of course, controlling the humidity is only the first step, but before wielding the power of hurricanes, it's nice to practice with something less likely to backfire. Then, watch how you're doing with wview (a program for monitoring and logging weather patterns—see Mark Teel's article on page 34), and move forward as you master smaller things like fog.

Not everyone is bent on world domination. If you fall into that category, there's plenty of fun for you here too. Sure, you could use the projects I already mentioned for more mundane and non-evil purposes, but we also have some things designed strictly for fun. If you just like to sit back and watch TV, Anthony Dean shows how to rip DVDs to your computer for some digital watching on whatever your favorite media-consumption platform might be. Dirk Elmendorf explains how to hack new channels into your Linux-based Roku device as well, so you can watch whatever you want, whenever you want.

Finally, it's possible that instead of taking over the world, you simply want to drink beer and play video games. Well, you're not alone. Kyle Rankin shows how to brew the perfect beer in a temperature-controlled homebrew fermenter. It's controlled by Linux, but consumed by humans. Add to that his article on connecting *Rock Band* instruments to your Linux machine, and you can have a weekend that'll remind you of your college years.

So whether you're a mad scientist or a bored geek, this special issue is pure fun. Oh sure, you might learn something along the way, but just like in the LEGO-building days of my youth, the coincidental learning is hardly noticeable—until the world gets attacked by giant alien robots that is, *then* you'll all be asking for my plasma swords.

We hope you enjoy this bonus issue of *Linux Journal*. If you read this whole thing and are still anxious to get your hands on more Linux and open-source-related information, but your first paper issue hasn't arrived yet, don't forget about our Web site. Check us out at **www.linuxjournal.com** for thousands more articles, tech tips, videos and more.■

Shawn Powers is the Associate Editor for *Linux Journal*. He's also the Gadget Guy for LinuxJournal.com, and he has an interesting collection of vintage Garfield coffee mugs. Don't let his silly hairdo fool you, he's a pretty ordinary guy and can be reached via e-mail at shawn@linuxjournal.com. Or, swing by the #linuxjournal IRC channel on Freenode.net.

Temper Temper

A $15 USB thermometer and some spare parts are all I needed to control my refrigerator with Linux.

KYLE RANKIN

If loving Linux ever became a crime and I were hauled into court, I think the prosecution's argument would go something like this:

> Your honor, I need to submit only two pieces of evidence to make my case. First, I present Exhibit A: a stack of *Linux Journal* magazines of which the defendant is a columnist.
>
> And your second piece of evidence?
>
> Your honor, the defendant's refrigerator is powered by Linux.
>
> *(The audience gasps.)* Order! I've heard enough! Guilty!

I can't help it. I mean, why *wouldn't* you power your fridge with Linux if you had the chance? In my case, we recently purchased a new fridge for our house, which meant our spare fridge was headed for the garage where I would use it for beer fermentation. Fridges are well-insulated, and it seemed ideal for the task at hand, but the problem I ran into was that the built-in thermostat for the fridge would go up to only around 45–50°F at its warmest. To ferment ales, I needed to maintain temperatures between 60–72°F.

When most people convert fridges to ferment beer, they purchase a purpose-built device from their local brew shop. Essentially, you plug your fridge in to the device, plug the device in to the wall, and then set the analog thermostat on the device to your desired temperature. A temperature probe goes into your fridge, and when it gets too warm, the fridge is powered on. These devices range from around $70 to more than $100, depending on whether they are analog or digital, and I almost bought one until I realized I could do the same thing with an old Linux laptop, a couple pieces of hardware and a few scripts.

If you are into home automation at all, you are familiar with the X10 suite of home automation gadgets. Essentially, you can connect lamps and appliances to different X10 gadgets and then power them on with a remote control. There's even a remote control that connects to a serial port, so you can control everything from a computer. Linux has a program called bottlerocket that works great with X10 serial port controllers, and I had used one to control my DSL modem for many years, but that's something for another column.

The TEMPer USB Thermometer

So, I had a laptop and could control the fridge power, but I still needed a thermometer that worked under Linux and was relatively cheap. I discovered a great little USB-powered thermometer made by a company named TEMPer. It's small, cheap

Figure 1. TEMPer USB Thermometer (photo from the Amazon product page)

(less than $15 shipped), supports temperatures between –40°C and +120°C, and with a little effort, it works under Linux. It turns out many Linux administrators are using these devices to monitor temperatures in their data centers.

Apparently, the older versions of this thermometer showed up as a USB-to-serial interface; however, the newer models, including the one I bought, show up as a USB Human Interface Device when you plug it in:

```
Apr 16 14:44:33 muriel kernel: [11601.992205] usb 1-1:
 ➥new low speed USB device using uhci_hcd and address 2
Apr 16 14:44:33 muriel kernel: [11602.182910] usb 1-1:
 ➥configuration #1 chosen from 1 choice
Apr 16 14:44:33 muriel kernel: [11602.188481] usb 1-1:
 ➥New USB device found, idVendor=1130, idProduct=660c
Apr 16 14:44:33 muriel kernel: [11602.188529] usb 1-1:
 ➥New USB device strings: Mfr=0, Product=2, SerialNumber=0
Apr 16 14:44:33 muriel kernel: [11602.188563] usb 1-1:
 ➥Product:  PCsensor Temper
Apr 16 14:44:35 muriel kernel: [11604.090148] usbcore:
 ➥registered new interface driver hiddev
Apr 16 14:44:35 muriel kernel: [11604.119323] input:
 ➥PCsensor Temper as /class/input/input7
Apr 16 14:44:35 muriel kernel: [11604.140885] input,hidraw0:
 ➥USB HID v1.10 Keyboard [ PCsensor Temper]
 ➥on usb-0000:00:07.2-1
Apr 16 14:44:35 muriel kernel: [11604.170151] input:
 ➥PCsensor Temper as /class/input/input8
Apr 16 14:44:35 muriel kernel: [11604.188677] input,hidraw1:
 ➥USB HID v1.10 Device [ PCsensor Temper]
 ➥on usb-0000:00:07.2-1
Apr 16 14:44:35 muriel kernel: [11604.188931] usbcore:
 ➥registered new interface driver usbhid
Apr 16 14:44:35 muriel kernel: [11604.188980] usbhid:
 ➥v2.6:USB HID core driver
```

At first I thought I could get the temperature from this thermometer through some /proc or /sys interface, but unfortunately, the thermometer is more proprietary than that. The Linux community is resourceful though, and a quick search turned up a number of guides on how to pull the temperature

from Linux (see Resources for the most helpful guide I found). Essentially, you need to install a few custom Perl modules, including a special one created just for this device that depends on Perl 5.10, so you need a relatively new distribution for this to work (I used the latest stable Debian release).

Install Libraries and Perl Modules

I've always considered Debian as the distribution with the most packaged CPAN modules, but even it didn't have many of the modules I needed for the TEMPer thermometer, so I had to install them from scratch. I'll warn you in advance, this process is a bit tedious, and it reminded me of what it was like to install programs on Linux more than a decade ago. It's amazing how much we take the hard work of package maintainers for granted. At least the first dependencies I had (headers for libusb and a build environment to compile the Perl modules) were available with packages:

```
$ sudo apt-get install libusb-dev build-essentials
```

Next I needed to install a few Perl modules with the cpan program. What you'll find is that many of these modules have their own set of dependencies, so when you are prompted to install dependencies, just tell the cpan program "yes". Also, the first time you run cpan, you might have to go through the initial setup program. If so, just accept the defaults, and you should be fine. Here are the different cpan commands you need to run in order to install the various modules:

```
$ sudo cpan Bundle::CPAN
$ sudo cpan ExtUtils::MakeMaker
$ sudo cpan Inline::MakeMaker
$ sudo cpan Device::USB
$ sudo cpan Device::USB::PCSensor::HidTEMPer
```

Next I downloaded a Perl script from **www.cs.unc.edu/~hays/dev/bash/temper/temper_mon.pl** and made it executable. When run, the script will print the temperature from the thermometer. In my case, I modified it so it output in Fahrenheit instead of Celsius:

```
#!/usr/bin/perl

use 5.010;
use strict;
use warnings;
use Carp;
use Device::USB;
use Device::USB::PCSensor::HidTEMPer::Device;
use Device::USB::PCSensor::HidTEMPer::NTC;
use Device::USB::PCSensor::HidTEMPer::TEMPer;
use lib;
use Device::USB::PCSensor::HidTEMPer;

my $pcsensor  = Device::USB::PCSensor::HidTEMPer->new();
my @devices   = $pcsensor->list_devices();

foreach my $device ( @devices ){
   say $device->internal()->fahrenheit();
}
```

I stored the script in /usr/local/sbin/temper_mon.pl and ran it a few times to make sure it output the correct temperature. Then I connected the thermometer to a USB extension cord that was long enough to reach inside the fridge.

My Custom Fridge Script

The final step in the process was to write a script that would pull the temperature and control the power to my fridge based on whether it was within the proper maximum and minimum temperature ranges I had set. I decided to separate the max and min by two degrees so that the compressor wouldn't kick on too much. I also wanted to write the results to a log so I could monitor how well the fridge maintained the temperature. Plus, I thought it would be cool to `ssh` in to my laptop from anywhere in the world and check on the temperature.

When I first set this up, the weather was cool in the evenings, so I discovered that my fridge would dive down way below the minimum! My solution was to buy a $15 electric heating pad from the drugstore, connect it to another X10 outlet, and put it in the bottom of the fridge. I figured the heat would be gentle enough to maintain the temperature at

I can't help it. I mean, why *wouldn't* you power your fridge with Linux if you had the chance?

night without the risks that a proper space heater would have. I set up the script so that it would turn on the heater only if the temperature dipped down one degree below the minimum. I saved my script in /usr/local/sbin/temper.pl:

```
#!/usr/bin/perl

use 5.010;
use strict;
use warnings;
use Carp;
use Device::USB;
use Device::USB::PCSensor::HidTEMPer::Device;
use Device::USB::PCSensor::HidTEMPer::NTC;
use Device::USB::PCSensor::HidTEMPer::TEMPer;
use lib;
use Device::USB::PCSensor::HidTEMPer;

my $pcsensor  = Device::USB::PCSensor::HidTEMPer->new();
my @devices   = $pcsensor->list_devices();
my $temp_min  = 71;
my $temp_max  = 73;
my $logfile   = '/var/log/temper.log';
my $time      = localtime();
my $temperature;

foreach my $device ( @devices ){
   $temperature = $device->internal()->fahrenheit();
}

open LOG, ">> $logfile" or die "Can't open $logfile: $!\n";
```

Figure 2. A West Coast-Style Red Ale in My Linux-Powered Fridge

```
# B4 = Fridge power, B5 = Heater power

# turn on heater if I'm 1F below the low temp
if($temperature < ($temp_min - 1)){
   system('br --port /dev/ttyS0 B5 ON');
   print LOG "$time\t$temperature\tHON\n";
}
elsif($temperature < $temp_min){
   system('br --port /dev/ttyS0 B4 OFF');
   system('br --port /dev/ttyS0 B5 OFF');
   print LOG "$time\t$temperature\tOFF\n";
}
elsif($temperature > $temp_max){
   system('br --port /dev/ttyS0 B4 ON');
   print LOG "$time\t$temperature\tCON\n";
}
else{
   print LOG "$time\t$temperature\t\n";
}

close LOG;
```

The way the logic of the script works, it allows the temperature to drop or rise naturally while the compressor or heater is on, respectively. It changes the power state of my X10 devices with the br command only when the temperature is outside the preset ranges. I set this script to run every minute with cron, and because I log all of the power states, it's easy to watch the temperature float between extremes. I did a bit of tuning at the beginning with the various ranges, and with the current script, the temperature floats between 1°F below the minimum temperature and 1°F above the maximum temperature, which is good enough for me. If I wanted more accuracy, I always could set $min and $max to be closer to each other.

Since the system has been in place, I've been able to maintain the temperature successfully for the first batch of beer I put in the fridge. If you look closely at Figure 2, you can see the little thermometer on the right-hand shelf. Even though my laptop is old, it has plenty of horsepower to spare, so eventually I will graph all of the temperature data and serve it out over Apache. If Bill were here, I'm sure he'd tell me to tweet the temperature.■

Kyle Rankin is a Systems Architect in the San Francisco Bay Area and the author of a number of books, including *The Official Ubuntu Server Book*, *Knoppix Hacks* and *Ubuntu Hacks*. He is currently the president of the North Bay Linux Users' Group.

Resources

Guide to TEMPer Support under Linux:
https://wwwx.cs.unc.edu/~hays/archives/2010/03/entry_25.php

One of Many Places to Buy the Thermometer On-line:
www.amazon.com/TEMPer-USB-Thermometer-w-Alerts/dp/B002VA813U

TEMPer Thermometer CPAN Module:
search.cpan.org/dist/Device-USB-PCSensor-HidTEMPer

Building Custom Firmware with OpenWrt

Using an inexpensive wireless router, you can build a file server, a print server and even a media server. Plus, you can put all of those together and build the network device that does what *you* want. MIKE PETULLO

OpenWrt provides an environment for building custom, Linux kernel-based firmware for a variety of embedded devices. Originally targeting the Linksys WRT54G series of routers, OpenWrt now provides support for a wide range of devices, including Openmoko mobile phones and routers from Linksys, NETGEAR and D-Link. This article focuses on the Linksys WRT160NL router, which I chose because of its support for 802.11n wireless networking, its USB connectivity and its reasonable price. Within this device's 8MB of Flash and 32MB of RAM, I explain how to fit a Kerberos authentication server, LDAP directory server, NFS file server, print server and iTunes-compatible media server.

In addition to the Linksys WRT160NL router, you'll need an external USB hard drive, a USB printer, a USB hub, a build workstation and a custom-built console cable. To help build the console cable, You'll also need a continuity tester (often available as a function of a multimeter). The first two items are available for a total of less than $200. Any USB printer should suffice, as long as your network clients support it. The build workstation should be running an up-to-date Linux distribution. Finally, the console cable is a modified Nokia DKU-5 connectivity cable; this type of cable has a USB connection and embedded USB-to-serial adapter. In this article, I use the example.com domain, someuser user account and server.local router hostname. You should replace these three items with your own names.

This project has seven steps: preparing the build workstation, downloading the OpenWrt build environment, configuring and building a firmware image, installing a firmware image onto the WRT160NL, configuring the external USB disk, performing a post-installation configuration of the WRT160NL and configuring a network client. In addition, this article describes how to rescue a misconfigured WRT160NL using the console cable. If you find your router will not boot at any time while following these steps, skip ahead to that final section.

The first step to building a firmware image using OpenWrt is preparing your build workstation. OpenWrt requires that gcc, g++, binutils, patch, bzip2, flex, bison, make, gettext, pkg-config, unzip, the glibc headers, the libz headers, the ncurses headers and the perl-XML-parser are installed on the build workstation. All major distributions provide packages for these items, although the package names may be slightly different from the upstream titles.

Once you have installed all of the requisite packages on your build workstation, download OpenWrt using the command:

```
svn co -r 20526 svn://svn.openwrt.org/openwrt/trunk
```

(Before downloading OpenWrt, you may want to check if a newer version is available and substitute its revision number in the -r option.) This creates a directory named trunk in the current working directory. Enter this directory with `cd trunk`. Inside, you'll see the core OpenWrt build system. Additional packages are provided by what OpenWrt calls feeds. Pull in the feeds provided by the default configuration by executing:

```
./scripts/feeds update
```

Finally, complete the installation of the files required to build your optional packages with the command:

```
./scripts/feeds install krb5-server \
                        openldap-server \
                        nfs-kernel-server \
                        p910nd \
                        mt-daapd \
                        ntpd
```

The third step is to configure your firmware and build an image containing it. The OpenWrt build environment is similar to many BSD ports systems, MacPorts or Gentoo Linux in that it allows users to define a list of packages that the system will download and compile. Unlike these systems' general use, OpenWrt often must cross-compile its packages. For example, although my build workstation has an Intel Core 2 Duo processor, the WRT160NL router has an MIPS32 processor. As a result, before downloading and compiling general-purpose packages from source, OpenWrt downloads and builds a cross compiler and other build tools from source.

OpenWrt provides a configuration system that is very similar to the Linux kernel's and can be invoked with the command `make menuconfig`. You can navigate the menu-based configuration tool using the arrow keys and select submenus by pressing Enter. Activate an item with the Y key, or in the case of a series of choices, with Enter. Conversely, pressing N deactivates an item. Press the Esc key to return to a previous menu screen.

Within the configuration menu provided by the `make menuconfig` command, first select Target System and choose the Atheros AR71xx/AR7240/AR913x option. Set the Target Profile to Linksys WRT160NL. In the Target Images submenu,

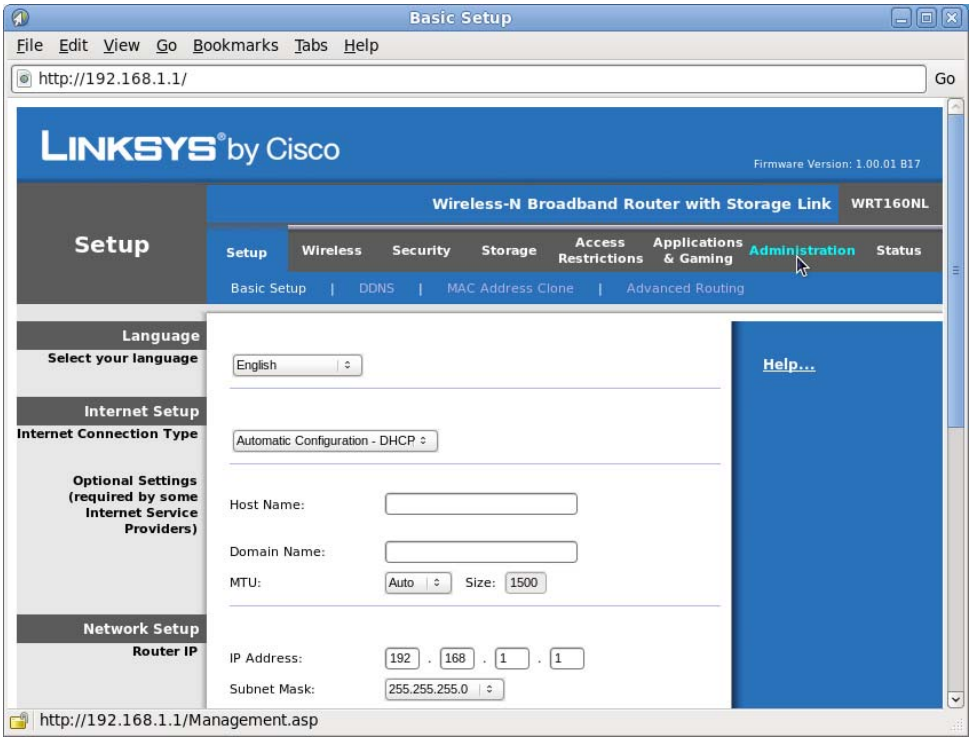


Figure 1. Firmware Upgrade Utility

ensure that only squashfs is activated.

Activate Advanced configuration options (for developers), and press Enter while this option is highlighted. Select Toolchain options, and press Enter again. Deactivate Build/install c++ compiler and libstdc++, because none of the packages for this example build require a C++ environment.

Return to the main menu using the Esc key. Select Base System, and then block-mount and block-hotplug.

Return to the main menu and select Network. Activate Filesystem, nfs-kernel-server; Time Synchronization, ntpd; Printing, p910nd; howl-mdnsresponder and openldap-server.

Return to the main menu and select Kernel Modules. Activate Filesystems, kmod-fs-ext3 and USB Support, kmod-usb-printer and kmod-usb-storage.

Return to the main menu, select Extra Packages, and activate krb5-server.

Return to the main menu, select Sound, and activate mt-daapd. Finally, Esc out of the configuration tool, ensuring that you save your configuration when prompted.

Now that your OpenWrt build is configured, execute the command `make V=99` to build the firmware image. This process takes a long time to complete because OpenWrt is compiling the build environment itself, followed by a Linux kernel and all of the firmware's programs. More than likely, this will take several hours.

The result of the build process is a firmware image stored at bin/ar71xx/openwrt-ar71xx-wrt160nl-squashfs.bin. You may install the OpenWrt firmware onto your router using Linksys' firmware upgrade tool, assuming you have not already replaced Linksys' default firmware. The Linksys firmware provides a Web-based configuration (Figure 1).

The Linksys firmware has a default IP address of 192.168.1.1, a default user name of "admin" and a default password of "admin". After configuring your build workstation by adding it to the 192.168.1.0 network, point your browser to 192.168.1.1. Click Administration and then Firmware Upgrade. Finally, click Choose File, and select the firmware image you just built, namely openwrt-ar71xx-wrt160nl-squashfs.bin. Follow the directions on the screen to replace Linksys' firmware with the OpenWrt firmware. Restart the router once the upgrade process is complete.

Now that you have installed your firmware on the router, it's an appropriate time to focus your attention on the USB disk that will serve as the router's data store. Connect the disk to your build workstation. You're going to create two partitions on the disk, one 64MB swap partition and one filesystem partition spanning the rest of the disk. You can do this with the following parted commands:

```
$ parted /dev/sdX
 (parted) mklabel msdos
 (parted) mkpart primary 0 64
 (parted) mkpart primary ext3 64 -0
 (parted) quit
$ mkswap /dev/sdX1
$ mkfs.ext3 /dev/sdX2
```

Remember to replace the X in /dev/sdX with the correct letter for the newly connected drive on your system. Finally, create a directory skeleton as follows (assuming /mnt is an unused mountpoint):

```
$ mount /dev/sdX2 /mnt
$ mkdir -p /mnt/Storage/Music /mnt/home /mnt/mt-daapd
```

Now, copy your music library to /mnt/Storage/Music.

One final step remains before you turn your attention to your clients. The final step on the router is the post-install configuration. Connect the newly initialized disk, the hub and the printer to your router, and restart the router with the router connected to your build workstation but not yet connected to a public network. Ensure that your workstation is configured to obtain an IP address using DHCP. Once the router has finished starting, connect to it using `telnet 192.168.1.1`. Change the root password using `passwd`. Once this is complete, the router will no longer accept telnet

connections. Instead, you will connect using secure shell: `ssh root@192.168.1.1`.

Now that you've logged in to the router, let's pause and address an issue you may be wondering about: the next firmware upgrade. Once the OpenWrt firmware has been installed, you can no longer use the Linksys firmware upgrade tool that you used previously. You will perform future firmware installations using an OpenWrt-provided command-line tool. First, use the `scp` command to copy a firmware image to the router's /tmp directory. Then, log in to the router, and execute `mtd -r write <path-to-image> firmware`.

Continuing work at the router's command prompt, now let's create nine configuration files. The first file configures the router's hostname and timezone, /etc/config/system:

```
config system
     option hostname server.local
     option timezone  UTC
```

The next three files configure the router's network parameters. First, you'll configure the router's Ethernet devices. The following configuration will cause all five Ethernet interfaces to be bridged to perform switching on one network, 192.168.1.0. The configuration sets the router's IP address to 192.168.1.2 (this is a change from the default, 192.168.1.1). This configuration assumes that routing between the 192.168.1.0 and upstream networks and Internet DNS resolution will be performed by another device (for example, a DSL device) with an IP address of 192.168.1.1:

```
/etc/config/network:

config interface loopback
     option ifname  lo
     option proto   static
     option ipaddr  127.0.0.1
     option netmask 255.0.0.0

config interface lan
     option ifname "eth0 eth1"
     option type    bridge
     option proto   static
     option ipaddr  192.168.1.2
     option netmask 255.255.255.0
     option dns     192.168.1.1
     option gateway 192.168.1.1
```

The WRT160NL has a total of five Ethernet ports, but the configuration option above shows two:

```
option ifname "eth0 eth1"
```

This is correct. The WRT160NL's four LAN ports perform switching functions and are collectively referred to by the Linux kernel as eth0. The kernel refers to the single WAN port as eth1. This configuration bridges all five Ethernet ports together to perform switching functions on one subnet. It also configures the switch to act as a host, assigned the IP address 192.168.1.2. After applying this configuration, all five Ethernet ports are equivalent switch ports.

In addition to the Linksys WRT160NL router, you'll need an external USB hard drive, a USB printer, a USB hub, a build workstation and a custom-built console cable.

Next, configure the router's wireless interface:

```
/etc/config/wireless:

config wifi-device radio0
     option type     mac80211
     option channel  5
     option macaddr  <MAC-ADDRESS>
     option hwmode   11ng
     list ht_capab   HT40-
     list ht_capab   SHORT-GI-40
     list ht_capab   DSSS_CCK-40

config wifi-iface
     option device   radio0
     option network  lan
     option mode     ap
     option ssid     <SSID>
     option encryption psk2
     option key      <WPA2 KEY>
```

Configure the DHCP service by writing /etc/config/dhcp:

```
config dhcp lan
     option interface    lan
     option start        100
     option limit        150
     option leasetime    24h
     # GW, DNS:
     option dhcp_option "3,192.168.1.1 6,192.168.1.1"

config dhcp wan
     option interface    wan
     option ignore       1

config dnsmasq
     option leasefile   '/tmp/dhcp.leases'
     option resolvfile  '/tmp/resolv.conf.auto'
```

Next, remove the default fstab using `rm /etc/config/fstab`, because all mounts and swap space will be set up dynamically by the hotplug system.

Configure the disk's two shared directories using /etc/exports:

```
/mnt/sda2/Storage *(fsid=0,rw,insecure,no_subtree_check,sync)
/mnt/sda2/home *(rw,insecure,no_subtree_check,sync)
```

Configure Kerberos by creating /etc/krb5.conf:

```
[libdefaults]
    default_realm = EXAMPLE.COM
    dns_lookup_realm = false
    dns_lookup_kdc = false
    ticket_lifetime = 24h
    forwardable = yes

[realms]
    EXAMPLE.COM = {
        kdc = localhost:88
        admin_server = localhost:749
        default_domain = local
    }

[domain_realm]
    .local = EXAMPLE.COM
    local = EXAMPLE.COM
```

OpenLDAP's configuration file is /etc/openldap/slapd.conf:

```
include     /etc/openldap/schema/core.schema
include     /etc/openldap/schema/cosine.schema
include     /etc/openldap/schema/inetorgperson.schema
include     /etc/openldap/schema/nis.schema
include     /etc/openldap/schema/autofs.schema

allow bind_v2

pidfile     /var/run/openldap/slapd.pid
argsfile    /var/run/openldap/slapd.args

database    ldif
directory   "/etc/openldap/ldif"
suffix      "dc=example,dc=com"
rootdn      "cn=Manager,dc=example,dc=com"
rootpw      "<PASSWORD>"
```

Configure the media server, mt-daapd, by writing to /etc/mt-daapd.conf:

```
web_root        /usr/share/mt-daapd/admin-root
port            3689
admin_pw        <PASSWORD>
db_dir          /mnt/sda2/mt-daapd
mp3_dir         /mnt/sda2/Storage/Music
servername      server.local
runas           nobody
playlist        /etc/mt-daapd.playlist
extensions      .mp3,.m4a,.m4p
```

Finally, use /etc/config/p910nd to configure printer sharing:

```
config p910nd
    option device         /dev/lp0
    option port           0
    option bidirectional  1
    option enabled        1
```

Because OpenWRT sometimes starts services before the kernel initializes the USB subsystem, you need to make one last modification. Edit /etc/rc.d/S50mt-daapd and add `sleep 5` as the first line in the start() function. This ensures that mt-daapd does not start until the kernel is aware of the USB disk containing your media files.

Reboot the router to ensure all configuration changes take effect.

The next step is to initialize the router's Kerberos database. Log back in to the router using `ssh root@192.168.1.2`. Initialize the account database using the command `kadmin.local`, which provides an interactive interface:

```
$ kadmin.local
  > add_principal someuser
  > exit
```

Returning to the build workstation, let's initialize the LDAP database. First, create a file named example.com.ldif with the following contents, a database defining a user account and automount configuration in LDIF format:

```
dn: dc=example,dc=com
objectClass: organization
objectClass: dcObject
o: Example Organization
dc: example

dn: ou=people,dc=example,dc=com
objectClass: organizationalUnit
ou: people

dn: ou=group,dc=example,dc=com
objectClass: organizationalUnit
ou: group

dn: cn=ldapusers,ou=group,dc=example,dc=com
objectClass: posixGroup
objectClass: top
cn: ldapusers
userPassword:: WFhYWA==
gidNumber: 1002

dn: uid=someuser,ou=people,dc=example,dc=com
uid: someuser
cn: Some User
objectClass: account
objectClass: posixAccount
objectClass: top
userPassword:: WFhYWA==
loginShell: /bin/bash
uidNumber: 1100
gidNumber: 1002
homeDirectory: /home/someuser
gecos: Some User

dn: automountMapName=auto_master,dc=example,dc=com
objectClass: top
objectClass: automountMap
automountMapName: auto_master

dn: automountKey=/home,automountMapName=auto_master,dc=example,dc=com
```

```
objectClass: top
objectClass: automount
automountKey: /home
automountInformation: auto.home

dn: automountMapName=auto.home,dc=example,dc=com
objectClass: top
objectClass: automountMap
automountMapName: auto.home

dn: automountKey=*,automountMapName=auto.home,dc=example,dc=com
objectClass: top
objectClass: automount
automountKey: *
automountInformation: server.local:/mnt/sda2/home/&
```

Next, load the file into the LDAP database on the router using:

```
ldapadd -x -D "cn=Manager,dc=example,dc=com" -W -f example.com.ldif
```

Finally, return to the router and create someuser's home directory, `mkdir /var/sda2/home/someuser`, and then do `chown 1100:1002`. Notice that the UID:GID arguments to the chown command match those assigned to someuser in the LDIF file above. Now you can connect the router to your real network.

To add additional users, review the fifth block in the LDIF file above, tailor it for each user and add them using `ldapadd`. Also, create each user's home directory as before.

Your router now should be fully functional, so it is time to configure the clients. Several existing articles document how to configure a client to operate in LDAP and Kerberos environments (see Resources). You may configure your clients by editing configuration files, or you can investigate your distribution's administrative tools.

The OpenWrt build environment is similar to many BSD ports systems, MacPorts or Gentoo Linux in that it allows users to define a list of packages that the system will download and compile.

One difficult point I've encountered has to do with the LDAP schema used by the automounter. Different UNIX flavors use different schemas. My instructions use the schema employed by Mac OS X, because it also is supported on Fedora. In order to instruct the Fedora automounter to use this schema, write the following to /etc/sysconfig/autofs:

```
MASTER_MAP_NAME="auto_master"
TIMEOUT=300
BROWSE_MODE="no"
USE_MISC_DEVICE="yes"

# Mac OS X 10.5-compatible schema:
MAP_OBJECT_CLASS="automountMap"
ENTRY_OBJECT_CLASS="automount"
MAP_ATTRIBUTE="automountMapName"
ENTRY_ATTRIBUTE="automountKey"
VALUE_ATTRIBUTE="automountInformation"
```

Other distributions may configure the automounter differently. If the automounter is configured to use the appropriate schema, it will learn of the NFS-provided home share from the LDAP entry you created earlier and mount users' home directories on-demand.

Refer to your client system's documentation for instructions covering how to connect to the network printer. Your OpenWrt firmware will share an attached USB printer using the HP Jetdirect protocol. This protocol is supported by Linux, Mac OS X, Windows and many other operating systems.

Your OpenWrt device's music share is accessible by iTunes, Rhythmbox and any other media player that supports the DAAP protocol. Compatible players usually will display the share as an option alongside their local music library list.

What if you install a defective firmware, and the router is left in a state that will not boot? This is where your console cable comes in. Figure 2 shows a completed console cable next to an unmodified DKU-5. There are three points on the WRT160NL to which you can connect a console cable. The first are two identical sets of leads within the WAN and LAN 4 RJ45 sockets,

Figure 2. Console Cable

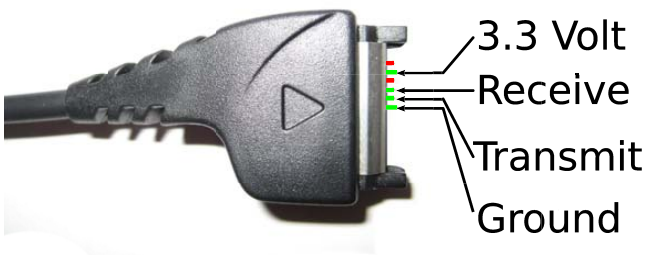


Figure 3. DKU-5 Connector

opposite the Ethernet leads. The second is a five-post connector on the WRT160NL's motherboard. In order to use the latter interface, you have to open the WRT160NL's case, which voids the device's warranty.

Figure 3 provides a picture of a Nokia DKU-5 cable connector. The other side of the cable has a USB connector. Cut the cable in half so that most of its length is on the USB connector side. Now, expose the six leads on the Nokia connector side. Use a continuity tester to identify which four of the six leads correspond to the functional pins noted in Figure 3. Note the insulator color of the lead connected to each pin. Now focus your attention to the other half of the cable, the one with the USB connector. Expose the four leads matching the colors noted earlier. Connect each of these leads to the four of five binding posts labeled in Figure 4, ensuring that the lead and post functions

What if you install a defective firmware, and the router is left in a state that will not boot?

match. You may connect them using a PCB connector, hook to pico hook jumpers or a more primitive technique.

Connect the USB connector of the console cable to your build workstation. Install a serial terminal emulator, such as minicom, on your build workstation and run the program. Configure the emulator to use eight data bits, no parity bit and one stop bit. Select 115200 baud. Boot the router and observe the emulator console. You should see the router print diagnostic information through the console interface. Pay close attention to the messages being printed through the console, and press a key when the U-Boot bootloader prompts you to "Hit any key to stop autoboot." You should see the U-Boot command prompt, `ar7100>`. Enter the command `upgrade code.bin` to instruct U-Boot to initiate a tftp server. Return to the build workstation command line, enter the command `tftp 192.168.1.1`, and then:

```
tftp> binary
tftp> put <firmware filename>
```

Observe the data transfer being displayed over the console interface. Return to the terminal emulator, and enter `go` at the U-Boot prompt.

The firmware you load using this technique may be an

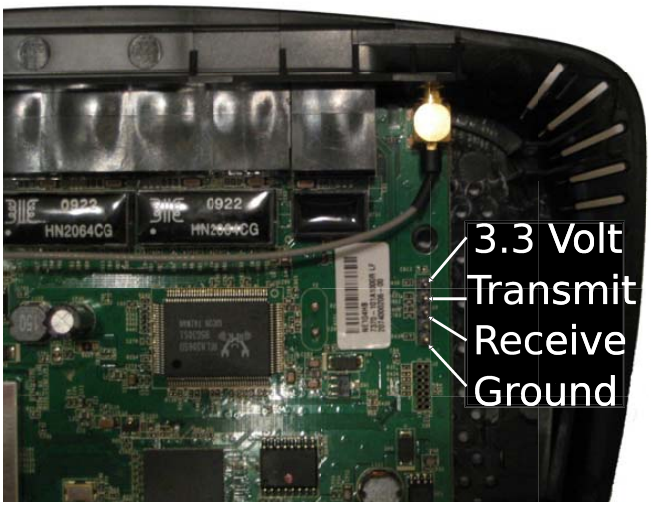


Figure 4. Router's Console Interface

official firmware obtained from Linksys or an OpenWrt firmware that you built.

Conclusion

This article demonstrates a technique for providing Kerberos, LDAP, network filesystem, print and media services using a Linksys WRT160NL wireless router. The result is a low-cost network server for the home or small office. OpenWrt has a wide range of packages available, so there is potential for many other solutions to be developed around this capable platform. For example, Samba could allow tight integration with Windows clients. Another option is Netatalk, which provides native Mac OS X file sharing, including integration with Apple's Time Machine backup software. Linux, open-source applications and popular network hardware like the Linksys WRT160NL provide a solid basis for developing innovative devices.■

Mike Petullo serves in the US Army and enjoys solving problems with innovative open-source software. He has been working with Linux since 1997 and welcomes your comments at mike@flyn.org.

Resources

"OpenLDAP Everywhere" by Craig Swanson and Matt Lung, *LJ*, December 2002 (see the section titled "Configure the Linux LDAP Client"): **www.linuxjournal.com/article/6266**

"Centralized Authentication with Kerberos 5, Part I" by Alf Wachsmann, *LJ*, January 2005 (see the section titled "Configuring the Clients"): **www.linuxjournal.com/article/7336**

"Serving Apples: Integrating Mac OS X clients into a Fedora network" by Mike Petullo, *Red Hat Magazine*, January 2008 (demonstrates how to configure Mac OS X clients): **magazine.redhat.com/2008/01/17/serving-apples-integrating-mac-os-x-c lients-into-a-fedora-network**

Remix the Internet and Your Television with the Roku DVP

Developing custom channels for the Roku.

DIRK ELMENDORF

This month, the challenge is to write about Linux and entertainment. MythTV is cool, but it's been covered. Boxee looks great, but someone beat me to that punch. I was getting worried I wouldn't find something appropriate to cover until my wife reminded me that the device on which we were watching *Farscape* runs Linux, and she suggested I see if there was something worth discussing beyond the great selection of documentaries. As luck would have it, there is quite a bit worth writing about.

The device is the Roku Digital Video Player, which launched in 2008. This tiny box uses Linux to stream Netflix video to your television. Until recently, that was the only reason to get one, but now, Roku has started announcing other channels of content (Major League Baseball, Facebook, Pandora and more). It is interesting to see the tiny little box expand its capability. Even more interesting is that Roku has opened a developer program that allows anyone to create custom channels for it.

Before I dive in, here's a little background. My wife is from Seattle, Washington, and is a huge fan and supporter of the KEXP radio station. I have done several projects in the past to bring KEXP to our home in San Antonio, Texas, but looking at the Roku's capabilities, it seems like the perfect platform for bringing KEXP on-demand to our living room.

The first step in becoming a developer is to go to the Roku.com Web site and create an account in its portal. After that, click the link for Developer. There is a small sign-up form where you agree to abide by the rules of the developer program. Once completed, you can download the SDK and get started.

Roku tries to honor its obligation to the GPL code involved in allowing the Roku to work. If you visit **www.roku.com/support/gpl_rdvp.aspx**, you can download tarballs of software. One thing that struck me in reading the developer agreement is the section that says "Subject to the Grace Period, Your Channel Application must at all times....iii. not contain any open-source code or other restricted code that could require Roku to publicly post or display any third-party notices or any modifications to such code." I understand what the Roku folks are trying to protect themselves from. Apparently, parts of the Roku platform are licensed from third parties (like the support for WMA), and Roku wants to keep parts proprietary. I assume the MIT license is fine, but it sounds like this environment might not be very friendly to the GPL. I posted a question to the developer forum but did not get any response by the time I had submitted this article for publication.

Assuming you still are interested in the platform, you have three different options for deploying a channel: push it to a local Roku, create it as a private channel or get it approved as a public channel.

The public channel requires Roku's approval. Once it's approved, Roku offers it in the channel store on the Web site. This is the path to take if you are building a channel for public consumption. The private channel allows you to roll out your channel to people. The upside is you don't need Roku's approval to roll it out. The downside is that more steps are involved in getting the channel onto a Roku. The local push is designed for the development phase of your channel. Once you enable development on a Roku (use a remote to enter Home 3x, Up 2x, Right, Left, Right, Left, Right), you can just upload your channel. The SDK includes a handy Makefile to make this incredibly easy. This is how I have done all of my development, as I'm developing channels for myself (and other developers). The main limitation of this last form of development is that you can have only one "development" channel installed at a time. That did not pose much of a problem for me, but if you're sharing code, it could make things more complicated.

Roku development is all done in BrightScript, which appears to be a custom language that combines ideas from Visual Basic and JavaScript.

Roku development is all done in BrightScript, which appears to be a custom language that combines ideas from Visual Basic and JavaScript. The main goal is to allow you to write code in a dynamic language that can easily compile down to efficient code for the Roku's embedded environment. This is accomplished by adding in BrightScript Components, such as default screens and media tools. You use BrightScript to customize the screens and build up your channel. There is even a PowerPoint presentation template to allow you to mock up your application screens easily.

Getting Down to Business

All the code for this article is being shared at **github.com/economysizegeek/linux_journal_roku**. This will make it a lot easier to see the different channels. Simply `git clone` that repository, and you will have the latest version of all the code. I assume you have turned on developer mode for your Roku. You also need to set an environment variable in your shell so that the Makefile will know where to push the code. In my case, I added `export ROKU_DEV_TARGET=192.168.210.244` to my .bashrc. Make sure you put in the IP address of your Roku.

Let's start by doing a simple Hello World! example. This will get you familiar with the tools and also confirm that you have everything working. In the git repository is a directory called hello_world. Inside that directory, you should be able to type `make install`, and it will push a new channel onto your Roku automatically. If that doesn't work, confirm that you have make, curl and zip installed (the Makefile depends on them). Also make sure your Roku is in Dev Mode (enter the key combo from this article). Additionally, ensure that you didn't set the environment variable (`echo $ROKU_DEV_TARGET` should print an IP), and check that you didn't set it to the wrong IP (the Roku's IP can be confirmed in Settings→Player Info).

Once you have sorted that out, you should be able to go to the Roku and see a new channel called "Hello World". Clicking on it with your remote will start the channel. It doesn't actually do anything other than say "Hello World!" The point is to confirm your environment and give you a quick-and-dirty tour of what you need to start making your own channel.

In the hello_world directory, you will see two directories and a few files. The images directory has a variety of images needed for this small application. You will notice that some have HD and some have SD in their names. This represents High Definition and Standard Definition. Because the Roku is connected to a television, you have to make sure any art you display is sized correctly for the TV screen. The Makefile is there to make it easy to compile, install and remove the channel. The manifest file is required by the Roku to build the package on the other side. That just leaves the source directory and the HelloWorld.brs file (Listing 1). The brs file is the BrightScript that actually sets up the channel.

Let's do a walk-through of the code, but if the version you get from the git repository is different, it means I fixed a bug after this went to press.

The first code listing is a crash course in BrightScript. All it does is create a Main that builds a single screen and exits on any button push. The print statements are spit out to the Roku console. The console also allows you to enter a debugger and see any syntax errors. You can access the Roku console by entering `telnet $ROKU_DEV_TARGET 8085`. I had to use it a couple times to find out why the channel wasn't showing up (which often means there is a syntax error).

As you can see, I created a function called Main. This is the entry point for all BrightScript applications. I included two different print statements. The first one works, but the second one only prints out a blank line, because a single quote is considered a comment in BrightScript. Then, I created two objects. The roMessagePort is where messages (events) are sent. The roSpringboardScreen is a BrightScript component. This is basically a screen you can reconfigure by providing information. Roku ships with a number of these screens for your use. I chose this one because it was the easiest to work with. I connect the screen and the port. This tells the system to send all events from the screen to the message port I created. There probably are cases where you need to have different message ports for different events, but so far, I have used only a single message port shared with every screen.

Listing 1. HelloWorld.brs

```
'Copyright 2010 Dirk Elmendorf

Function Main() as Integer
  print "Hello World! to the console"
  print ' you will not see this '
  p = CreateObject("roMessagePort")
  screen = CreateObject("roSpringboardScreen")
  screen.SetMessagePort(p)

  o = CreateObject("roAssociativeArray")
  o.ContentType = "episode"
  o.Title = "Sample App"
  o.Description = "Hello World!"
  o.SDPosterURL = "pkg:/images/episode_icon_sd.png"
  o.HDPosterURL = "pkg:/images/episode_icon_sd.png"
  screen.SetContent(o)
  screen.AddButton(1, "Exit")
  screen.Show()
  while true
    msg = wait(0, p)
    if msg.isButtonPressed()
      print "Goodbye World!"
      return 0
    endif
  end while
end Function
```

Then, I create a roAssociativeArray. This is very much like a hash (or a plain JavaScript object). I use it to set up up a number of predefined fields. This gets passed into the screen. Next, I create a single button on the screen and tell the system to draw the screen. The last step is to set up an infinite loop. The wait allows the system to sit until an event is triggered. Then, I can have the result of that event processed. In this case, it means the program exits.

Rocking Out

The next step was to add in another BrightScript component—namely roAudioPlayer. This component handles playing MP3 streams. I wrote a new script from scratch to do this. The channel to handle streaming is in the kexp directory. The main file (Listing 2) is in the source/KEXP.brs (at **github.com/economysizegeek/linux_journal_roku**).

Listing 2. KEXP.brs

```
'Copyright 2010 Dirk Elmendorf

Function Init() as Object
  obj = {
    port: CreateObject("roMessagePort")
    screen: CreateObject("roSpringboardScreen")
    player: CreateObject("roAudioPlayer")
    screen_options: CreateObject("roAssociativeArray")
    song: CreateObject("roAssociativeArray")
    status: ""
    drawScreen: function(description)
        m.screen_options.Description = description
        m.screen.SetContent(m.screen_options)
        m.screen.Show()
    end function
    playingNow: function()
        m.screen.ClearButtons()
        m.screen.AddButton(1, "Pause Stream")
        m.screen.AddButton(3, "Exit")
        m.DrawScreen("Live MP3 Stream from KEXP.org")
    end function
    play: function()
        if m.status = "" then
          m.player.AddContent(m.song)
          m.player.SetLoop(true)
          m.player.play()
          m.status = "playing"
        else if m.status = "paused"
          m.player.resume()
          m.status = "playing"
        endif

        m.screen.ClearButtons()
        m.screen.AddButton(3, "Exit")

        m.drawScreen("Buffering....")
    end function
    pause: function()
        if m.status = "playing" then
          m.player.pause()
          m.status = "paused"
        endif

        m.screen.ClearButtons()
        m.screen.AddButton(2, "Resume Stream")
        m.screen.AddButton(3, "Exit")

        m.drawScreen("Stream Paused")
    end function
    exit: function()
        print "Goodbye World!"
        m.player.stop()
        return 0
    end function
  }
  obj.screen.SetMessagePort(obj.port)
  obj.player.SetMessagePort(obj.port)


  obj.screen_options.ContentType = "episode"
  obj.screen_options.Title = "KEXP"
  obj.screen_options.SDPosterURL = "pkg:/images/episode_icon_sd.png"
  obj.screen_options.HDPosterURL = "pkg:/images/episode_icon_sd.png"
  obj.screen.SetStaticRatingEnabled(false)

  obj.song.Url = "http://kexp-mp3-2.cac.washington.edu:8000/"
  obj.song.StreamFormat = "mp3"
  obj.status = ""
  return(obj)
end Function

Function Main() as Integer

  app = Init()
  app.play()

  while true
    msg = wait(0, app.port)
    if type(msg) = "roAudioPlayerEvent"
      if msg.isStatusMessage() then
        print "Audio Player Event:"; msg.getmessage()
        if msg.GetMessage() = "start of play" then
          app.playingNow()
        endif
      endif
    else if msg.isButtonPressed()  then
        if msg.GetIndex() = 1 then
          app.pause()
        else if msg.GetIndex() = 2 then
          app.play()
        else if msg.GetIndex() = 3 then
          return app.exit()
        endif
    else if msg.isScreenClosed() then
      return app.exit()
    endif
  end while
end Function
```

Listing 2 is for this channel. It is different in two main ways. First, this time, the event loop actually looks at the button being pushed. The loop is now able to trigger a different behavior depending on the index value of the button that was pushed. The other change is that I created an application object. I wanted to demonstrate that Roku has provided an incredibly dynamic language, so I built it up much the same way you would in JavaScript. The one odd thing to notice is the use of "m." inside the methods for the app object. The "m." is a pointer to the object.

You can read it as "this.", because that is how Roku is resolving it when it gets compiled.

The event code is also tracking the state of the stream itself. The roAudioPlayer considers it an error if you try to pause a stream that is not playing or try to resume a stream that is not paused. The code handles those cases. It also handles shutting down the stream when you exit.

With a working streaming application in hand, I started trying to get the "on-demand" portion of the KEXP Web site integrated into it. The first step was to navigate to a show and find a URL for a single show. This took some work, because the on-demand programs are served using a different protocol and URL from the live feed. Eventually, I was able to get a URL by downloading and opening files that the KEXP server provides.

Unfortunately, I ran into a Roku platform limitation. It does not support the codec that KEXP is using. Currently, there is no way for a third-party developer to add additional codec support. Because the live stream is available in MP3, the Roku handles it fine, which has been the case for a number of the radio feeds I looked at. So, if the stream you want to use streams via Flash or some other codec, you may be out of luck as well. Roku also supports WMA, but that seems to be more for files than streams.

For this article, I wrote everything from scratch. When you are working on your own projects, you won't need to do that. Roku includes a number of different sample applications in the SDK. The code gives a great tour of the components and how they can be used, including audio, video, XML parsing and handling a registration process. Those examples also are a lot more polished than the code I have provided here (they include examples of how to create your own theme for the channel).

The next thing to play with is video streaming. I chose to focus on audio for this article, because it's very straightforward. Streaming video means obtaining a video source, as well as encoding it properly, and then you have to generate an XML document to describe it. You will need a Web server that can serve both the video and the XML. Once you have everything working, you should be able to run your own little Netflix-style video-streaming service. Now that you have your feet wet with Roku development, head over to Brian Lane's blog. He has documented some hacking he's done to get all of this working (**blog.brianlane.com/2009/12/20/streaming-local-video-with-your-roku**).

My goal with this article was to get streaming up for my wife (which I mostly achieved—I'm still working on getting the *Shake the Shack* show on demand). I was incredibly surprised at how easy it was to get up and hacking. Originally, I was turned off by having to deal with a new language, but BrightScript is close enough to JavaScript that it wasn't the hurdle I thought it would be. For me, the best part is that the Roku already had earned its place in my AV rack thanks to Netflix. Adding the ability to hack on it and the opportunity to try channels other people have built makes it that much more awesome.■

Dirk Elmendorf is cofounder of Rackspace, some-time home-brewer, longtime Linux advocate and even longer-time programmer.

Playing with the Player Project

The Player Project is a robot-control software framework for interfacing with PC-based robots. Learn how to use it to interface with sensors, actuators and even full research robots. KEVIN SIKORSKI

We've all heard that PC-based computers have increased in power and decreased in size, power consumption and cost. These improvements mean more people have access to them, but also that PCs are becoming more suited to being the brains of a mobile robot. It brings to robots a number of advantages, such as USB connectivity, greater memory capacity, more powerful processors and even allows for plugging in a mouse, keyboard and monitor to debug your robot *on* your robot. The largest cost associated with the choice of a PC-based robot, besides the dollar cost, is power consumption.

Programming your PC-based robot can be a little different from programming a robot that uses a smaller, embedded processor. As you would expect, PC-based robots can take advantage of programming features, such as threads, using multiple languages and leveraging third-party libraries to perform complex tasks.

What Is Player?

One such third-party library is the Player Project. The Player Project is a framework for building robot-control software. It provides a wide-reaching infrastructure that gives you a network protocol, data serialization support, a message loop and supports a large number of common off-the-shelf components, such as Webcams, laser range finders, RFID, GPS devices and interface boards. It even supports a large number of commercially available robots, and some robot manufacturers, such as CoroWare, provide Player drivers for their robots. And, of course, it's open source.

Requirements to Run Player

The newest version of Player runs on the majority of Linux distributions with little effort. It is also cross-platform, with version 3.0 supporting Microsoft Windows and even Cygwin under Microsoft Windows. It has low memory requirements and is pretty easy on your CPU. Although it's not completely cross-language, it does offer native support for C and C++, and has nice Python and Ruby interfaces.

You don't really need a real robot to work with Player. If you have a few sensors and actuators that you can connect to your laptop or desktop computer, you still can run the Player server and control those disembodied devices—think of it as a way to accomplish home automation. Player also can be used with some of its close cousins: Stage (a 2-D simulation system that integrates with Player) and Gazebo (a 3-D simulation system that also integrates with Player). In this way, you can simulate a full robot, or even a fleet of robots, without any special hardware.

What Does Player Look Like?

Several layers of the infrastructure are diagrammed in Figure 1, which should be familiar if you have written code that interacts with hardware devices.

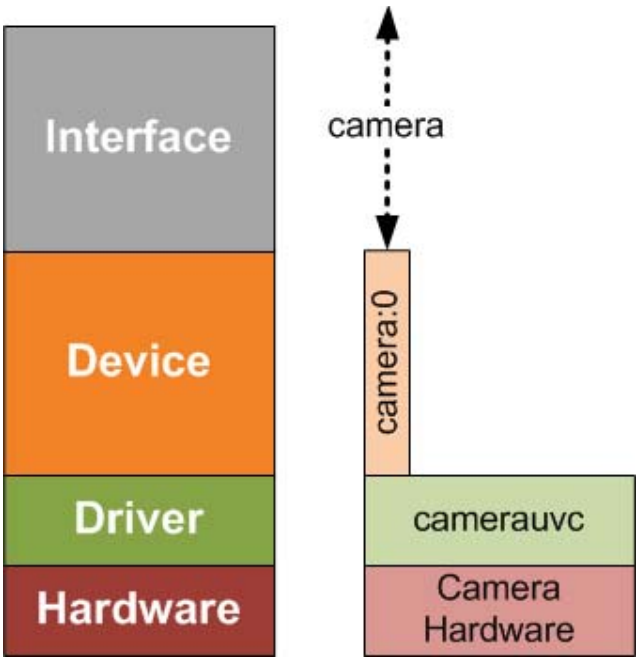


Figure 1. The relationship between Webcam hardware, the driver that interfaces with it, the device created by the driver and the interface that it exposes.

At the lowest level, we have the hardware layer. This corresponds to the hardware that embodies the sensor, actuator or any other physical component of the robot. The driver level sits on top of the hardware level. This is one layer where a programmer writes code. For example, a driver that interacts with a USB Webcam will provide whatever code is necessary to make a connection with the camera, read the sensor's output, package it up neatly and make it available to the rest of the system. A lot of drivers of this type are provided with Player as *static drivers*. This means you won't have to provide any special shared object libraries to use them, just the usual Player libraries. You also can write your own drivers, called plugins. The code for these drivers lives in a shared object library (.so file).

If you have a few sensors and actuators that you can connect to your laptop or desktop computer, you still can run the Player server and control those disembodied devices—think of it as a way to accomplish home automation.

You can create multiple instantiations of a single driver. When you do this, you need a way to refer to a specific instance. For example, if your robot has two cameras, one facing forward and one facing backward, you will need to tell them apart to know in which direction you are driving. In order to do this, Player gives you the concept of a device. Each time you instantiate a driver, you assign it a device name and number. For example, in your robot, camera:0 might refer to the forward-facing camera, and camera:1 might refer to the backward-facing camera.

Finally, we arrive at the concept of an interface. The interface is just like the API for a software package; it defines the software interfaces for getting data out of and putting data into the device. In our example, the camera interface defines a set of messages to get images out of the camera, and a set of messages to set up the camera and capture images.

Drivers don't have to exist solely for communicating with a piece of hardware. Player supports a number of algorithmic drivers, such as blob-finders or wavefront planners, or even camera image compressors. These operate just like normal drivers, consuming and producing data and exposing interfaces.

Player Configuration

Now that we have dealt with the vocabulary, we can get down to business. Let's say you've just acquired a CoroWare CoroBot robot (Figure 2). You've installed Player, and you want to make the robot do something interesting. You can type `player corobot.cfg` to run the Player drivers on the robot. This loads a configuration file that describes all the drivers Player must load to make your robot work. Here's an excerpt of a configuration file for the CoroBot:

```
driver
(
  name        "corobot"
  plugin      "libcorobotdriver"
  provides    ["position2d:0" "power:0" "ir:0"
               "limb:0" "gripper:0" "ptz:0"]
  requires    ["aio:0"]
  ssc32port   "/dev/ttyS1"
  ptzport     "/dev/video0"
)
```

We start off by telling Player that we are constructing a driver. The first two lines in the driver description tell Player where it can find the relevant code: the `name` line indicates the name of the driver in the code (a single library can supply multiple drivers), and the `plugin` line indicates the name of the shared object library that contains the code for this driver. In this case, it is stored in libcorobotdriver.so, in the same directory as the configuration file. The `provides` line specifies the devices that this driver makes available—in this case, the CoroBot driver exposes six devices. The `requires` line specifies the devices that this driver will consume. If any devices here are not present on the system when Player tries to instantiate the driver, Player will abort.

The last two lines in the driver description are not standard. Any driver is free to parse its driver description and make use of special identifiers. The CoroBot driver uses two of these: ssc32port specifies the Linux device (a serial port) through which it will communicate with its servo controller and a video device that controls its pan-tilt camera.

Player Tools

Player provides a number of tools to make working with Player easier. For example, you can use the playercam program to view the image provided by a Webcam. Let's say you have Player installed on your computer and a simple configuration file that brings up a camera driver:

```
driver
(
    name      "camerauvc"
    provides  ["camera:0"]
    port      "/dev/video0"
    size      [640 480]
)
```

Figure 2. CoroWare Robot

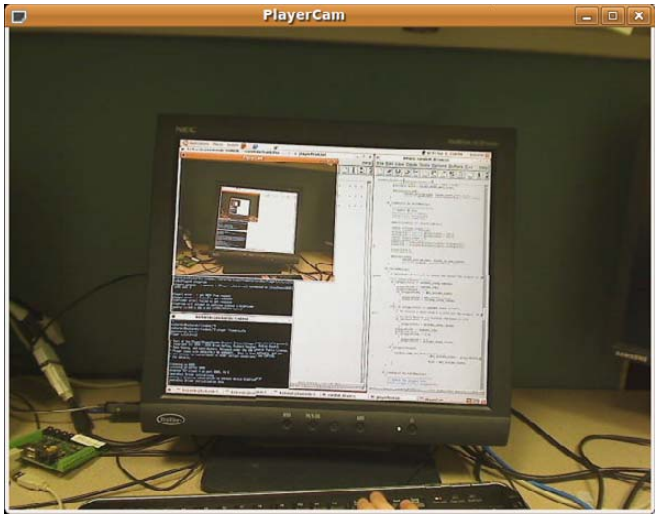


Figure 3. Player's playercam Tool, with the Webcam Pointed at the Computer Monitor

You can run Player with this configuration file with `player camera.cfg`, and then run `playercam` to see the camera image in real time (Figure 3). If your Webcam is on another computer—for example, on a PC-based robot—and connected by a network, you can just as easily see the Webcam output by running:

```
playercam -h hostname -p port
```

playerprint is a tool that works much like playercam, displaying the output of a device to the user. But, playerprint does this textually and can support a large number of interfaces, while playercam can support only the camera and blob-finder interfaces. For example, if we have a CoroBot running its Player drivers, we can display its infrared sensor readings with:

```
playerprint ir -h hostname -p port
```

Player also lets you control your robot, not just inspect it. playerv is a utility that also knows how to interact with many interfaces. Once you have started the Player server on your robot, you can run it with `playerv` (if you are on the same machine) or `playerv -h hostname -p port` (if you are on another computer). playerv will show a graphical display of the world around your robot, but it does not automatically connect with any devices. You will have to go to the Devices menu, and subscribe to the devices that you are interested in playerv plotting. In order to drive your robot around, you'll want to subscribe to the position2d device and select the option to "command" the interface. Then, you will be able to drag a small targeting reticle around the window to drive the robot (Figure 4).

Listing 1. Player's playerprint Tool Inspecting a GPS Device

```
#GPS (13:0)
#lat|long|alt|utm_e|
utm_n|err_horz|err_vert|num_sats|quality
47.6470103 -122.1414822  112.3       564477
5277425.1       0       0    6    1

#GPS (13:0)
#lat|long|alt|utm_e|
utm_n|err_horz|err_vert|num_sats|quality
47.6470107 -122.1414812 112.28    564477.07
5277425.14       0       0    6    1

#GPS (13:0)
#lat|long|alt|utm_e|
utm_n|err_horz|err_vert|num_sats|quality
47.6470113 -122.1414807 112.28    564477.11
5277425.21       0       0    6    1
```

Playing with Player

So far, our robot is awake, alert and ready to be told to do something interesting. Let's give it something to do. The CoroBot robot comes with a number of sensors and actuators—probably the easiest of which to interface with are the front- and rear-facing infrared ranging sensors and the mobility base's drive motors. Thus, we can write a small C program to talk to the Player server, read the IR sensors and drive the robot until it is 10cm away from an obstacle in front of the robot.

The first thing we have to do to interface with the Player server is open up a connection to it. For the sake of brevity, we will skip a lot of error checking, but you can download the full version of the code from the *LJ* FTP server (see Resources).

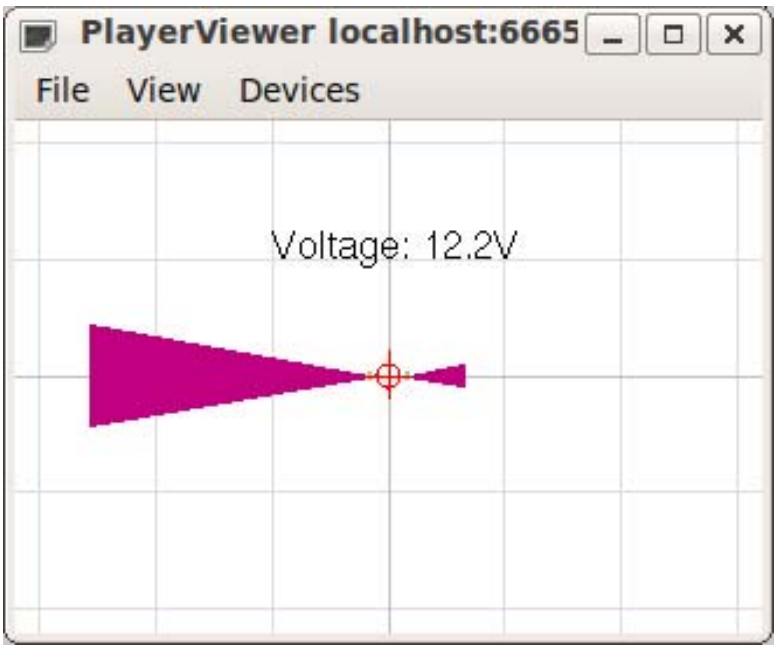


Figure 4. Player's playerv tool running on a CoroBot after subscribing to the IR and position2d devices. The large triangles are the cones shown to be obstacle-free by the infrared sensors.

The CoroBot robot comes with a number of sensors and actuators—probably the easiest of which to interface with are the front- and rear-facing infrared ranging sensors and the mobility base's drive motors.

This code defines the variables we will use to talk to the Player server and the device interfaces in which we are interested:

```
#include "libplayerc/playerc.h"
static playerc_client_t*      clientHandle;
static playerc_position2d_t*  positionProxy;
static playerc_ir_t*          irProxy;
```

The clientHandle is used for talking to the Player server itself. The second position2d interface talks to the position2d interface, providing us with encoder information about how the wheels are moving and allowing us to send motor commands to the robot. We'll ignore the encoder information for this example. Lastly, the IR interface gives us information about the distances that the robot's IR sensors are reporting.

The next code snippet uses these proxies to interface with the server and these devices:

```
playerc_client_connect(clientHandle);

// convert our interface to a PULL interface,
// only updates when we read
playerc_client_datamode(clientHandle, PLAYER_DATAMODE_PULL);

// tell the robot to drop older messages
playerc_client_set_replace_rule(
       clientHandle, -1, -1, PLAYER_MSGTYPE_DATA, -1, 1);

// create the position proxy (controls the motors)
positionProxy = playerc_position2d_create(clientHandle, 0);
playerc_position2d_subscribe(positionProxy, PLAYER_OPEN_MODE);

// create the IR proxy (controls the IRs)
irProxy = playerc_ir_create(clientHandle, 0);
playerc_ir_subscribe(irProxy, PLAYER_OPEN_MODE);
```

We start off by connecting to the Player server and configuring our connection. We want to get new messages from the server only when we are ready for them, so we configure the connection for a pull-type arrangement. And, because we want only the most recent information (we don't care what the IR sensors were indicating a second ago, we care about what they are saying right now), we tell the server to report only the most recent data. If we really wanted, we could let Player ensure that every IR message was delivered, but that might result in getting less-than-fresh data and possibly driving into a wall.

After our connection is configured, we open up the position2d interface on the Player server and subscribe to it. Then, we do the same with the IR interface. So far, so good. Now we need to get the state of the IRs from the robot and tell it how to move the motors:

```
while (!timeToQuit) {
  // attempt to read from the client
  if (playerc_client_read(clientHandle) == 0)
    continue;      // nothing to read, try again.

  // read the IR distances and verify we have good data
  if (irProxy->data.ranges_count == 2) {
    frontIr = irProxy->data.ranges[0];
    rearIr  = irProxy->data.ranges[1];
  }

  // figure out how to drive
  runController(frontIr, rearIr,
               &desiredTranslation,&desiredRotation);

  playerc_position2d_set_cmd_vel(
      positionProxy, desiredTranslation,
      0, desiredRotation, 1);
}
```

Each time through the loop, we try to read the newest data from the robot. After a little sanity checking, we take the ranges reported by the IR sensors and feed them into a controller function. This controller does some magic processing (we'll talk about that later) and returns information on how we should drive the robot. Finally, we pass these driving commands back to the Player server and start it all over again.

All that's left now is to provide a runController function that maps from IR sensor readings to drive commands. The CoroBot driver accepts numbers in the range of –1.0 to +1.0 to tell how to drive the robot forward and backward: +1.0 means 100% power forward, –1.0 means 100% power in reverse, and 0.0 means stop. It accepts the same range for telling the robot how to turn: –1.0 means turn full power left, +1.0 means turn full power right, and 0.0 means drive straight ahead. Noting that the IR readings are provided in meters, we can use the following P-controller to drive our robot forward until we are 10cm away from a front obstacle. We even get a bonus for free—if we are closer than 10cm away, the robot will back up a bit until it is at the proper distance:

```
void runController(double frontIr, double rearIr,
                   double *translation,double *rotation)
{
  // convert our IR readings into drive commands
  *translation = (frontIr-0.1) * 3.0;
  *translation = *translation > 0.9? 0.9: *translation;
  *rotation    = 0.0;
}
```

And finally, good programmers always shut down their server connections when they are done:

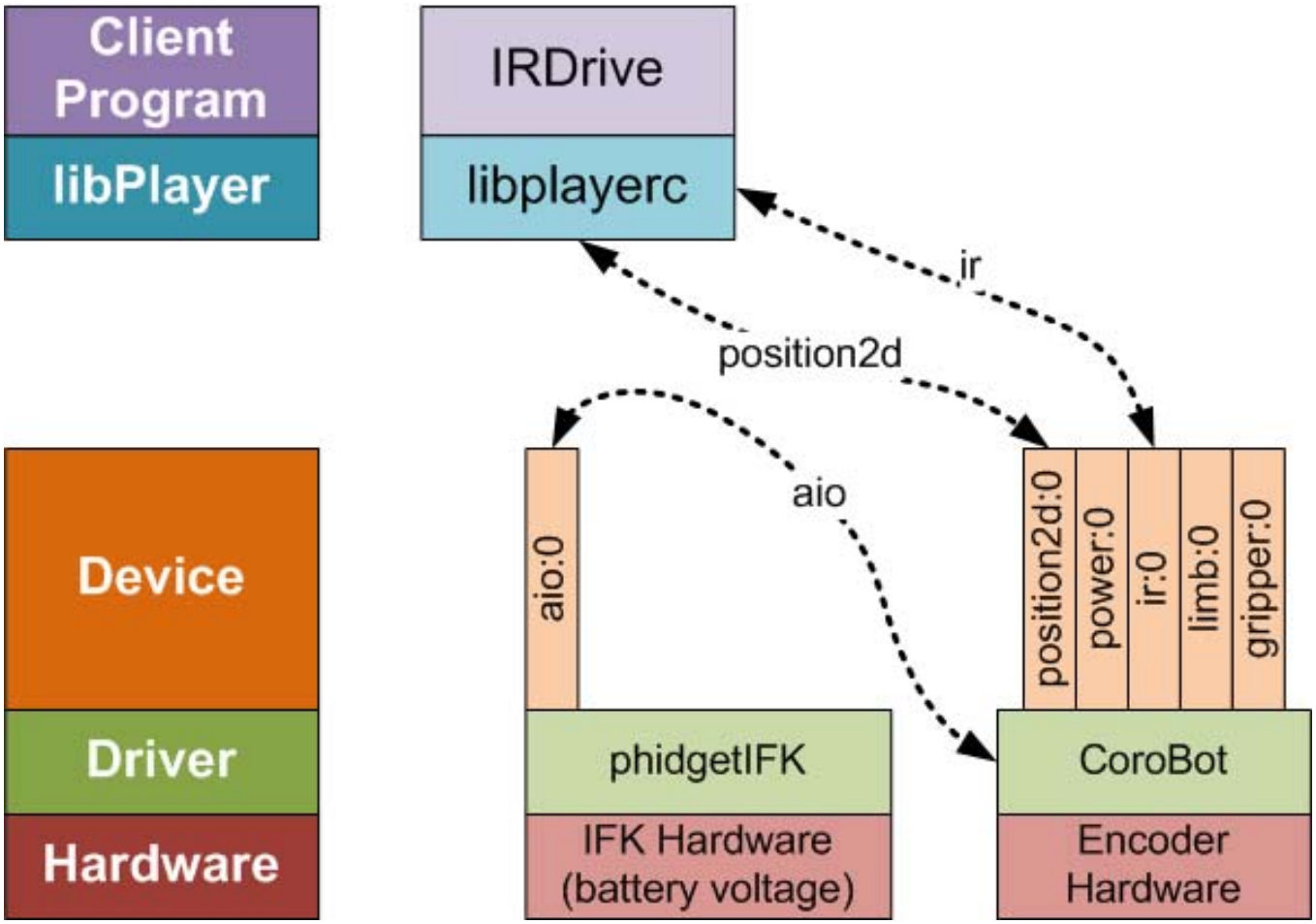


Figure 5. The Relationship between Several Devices and Interfaces When Using the Drive-by-IR Program

```
void shutDownProxies()
{
  // close down proxies we have opened
  playerc_ir_unsubscribe(irProxy);
  playerc_ir_destroy(irProxy);
  playerc_position2d_unsubscribe(positionProxy);
  playerc_position2d_destroy(positionProxy);
  playerc_client_disconnect(clientHandle);
  playerc_client_destroy(clientHandle);
}
```

Building on the design we showed earlier, we can see how our drive-by-IR program interacts with the Player infrastructure. The CoroBot configuration file loads the phidgetIFK driver, which exposes an aio:0 device. This device allows the CoroBot driver to read the robot's onboard infrared sensors. The CoroBot driver also exposes the position2d and IR interfaces, which the drive-by-IR program reads with the help of the libplayerc library (Figure 5).

The Player Project offers a lot of functionality that there just isn't room to get into in one article. This includes robot simulation, support for numerous commercial robots of many different prices and qualities, and support for a whole slew of readily available devices. Its plugin system even allows you to build your own drivers for new devices, either to support new hardware or to implement new experimental algorithms. Give it a try, and give your computer a chance to stretch its legs.■

Kevin Sikorski is a Robotics Architect at CoroWare Technologies where he designs, builds and programs mobile robots, and develops simulation software. In his spare time, he enjoys hiking in the Cascades and stargazing with his telescope.

Resources

The Player Project's Main Web Site:
playerstage.sourceforge.net

Full Source Code for the Drive-by-IR Program:
ftp.linuxjournal.com/pub/lj/listings/issue188/10566.tgz

CoroWare's CoroBot (a robot that provides drivers for working under the Player system): **www.corobot.net**

A List of Institutions That Use Player:
playerstage.sourceforge.net/wiki/PlayerUsers

Controlling the Humidity with an Embedded Linux System

Using an inexpensive embedded Linux board and a few extra devices, you can control things like room humidity.

JEFFREY RAMSEY

Charles Darwin, in his *Beagle Diary* that led to the book *Voyage of the Beagle*, wrote while in Peru, "On the hills near Lima, at a height but little greater, the ground is carpeted with moss, and beds of beautiful yellow lilies, called Amancaes. This indicates a very much greater degree of humidity, than at a corresponding height at Iquique." Like Darwin, I always have been conscious of humidity. For years, I've struggled with the humidity in my music room, as my Carlos Pina concert-grade classical guitar went out of tune frequently with wild swings in humidity. Pennsylvania winters are cold and dry, summers hot and humid, and this plays havoc on my classical guitar.

Commercially available humidifiers and dehumidifiers have humidity sensors that are far too coarse for certain applications. One such application is the humidity control for my music room. Being an embedded developer for my entire career, with a particular interest in embedded applications for Linux, I decided to build my own humidity controller for my music room. After a bit of research, I settled on a hardware architecture that includes a Cirrus EP9301 ARM9-based controller, several solid-state relays and a capacitive humidity/temperature sensor. Linux was my selection as the embedded OS, and with several Linux device drivers to control the relays and monitor the humidity and temperature, the basis of a humidity controller was born.

Linux was my selection as the embedded OS, and with several Linux device drivers to control the relays and monitor the humidity and temperature, the basis of a humidity controller was born.

I decided to use the humidifying and dehumidifying capability of my retail humidifier and dehumidifier units. The humidity controller that I built switches power on and off to the humidifier and dehumidifier, essentially assuming the role of the humidity sensor. To finish off the humidity controller, I added a Web interface that allows me to monitor and control the system through any network-attached browser, such as Firefox.

Before I began developing the embedded humidity controller, I had to decide on the system-level requirements. Even though this was for personal use only, it's always good practice to do a bit of systems engineering on the front end of the design process. I decided on the following requirements:

- The humidity control system should control humidity with a minimum range of plus or minus 3.5% rH.
- Humidifier and dehumidifier control will be through switching of 120V AC and neutral power lines.
- Current humidity and temperature will be displayed through a browser interface.
- Configuration of the desired humidity setting will be done through a browser interface.
- All humidity and temperature settings will be stored persistently in an SNMP MIB.
- All software will operate in an embedded Linux environment.

Figure 1 shows the overall embedded hardware architecture

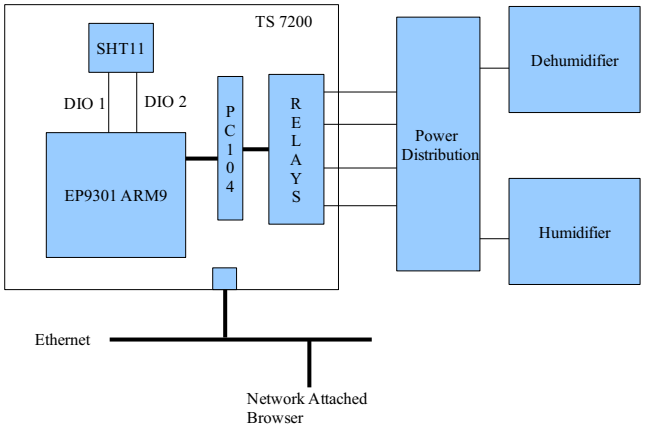


Figure 1. Hardware Architecture

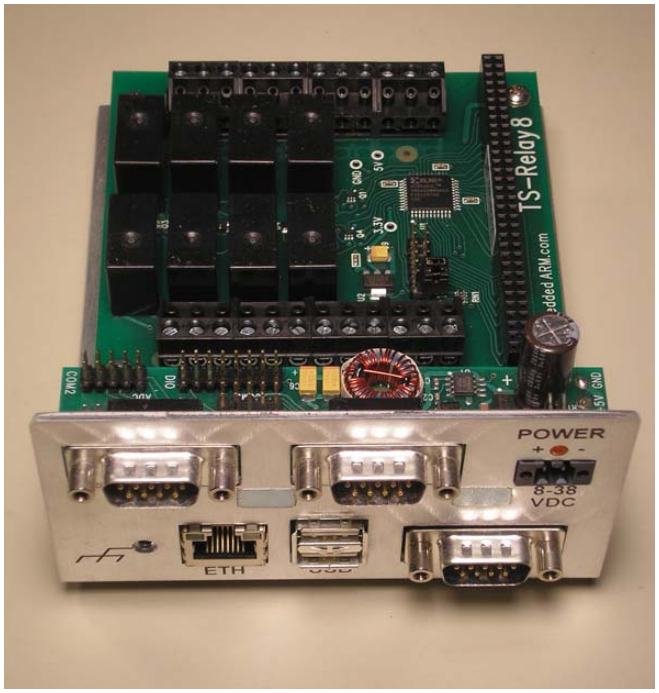


Figure 2. Hardware

of the humidity controller. The ARM9-based controller I selected is the TS-7200 from Technologic Systems. In addition to the controller board, I used a TS-RELAY8 peripheral board connected to the TS-7200's PC/104 bus. The daughter board contains eight SPDT relays. To house the system, I used a TS-ENC720 enclosure. Figure 2 shows the main board and peripheral board mounted on the back plate of the enclosure.

The capacitive humidity/temperature sensor is a Sensirion SHT11, which is controlled through a two-wire data/clock interface. The SHT11 control interface connects to two of the TS-7200's discrete I/O pins. Switching power on and off is accomplished with the single pole double throw (SPDT) relays on the peripheral board. I used a pair of relays for the humidifier and another pair for the dehumidifier. I used a pair as it seemed much safer to switch both the 120V and neutral lines, rather than just the 120V.

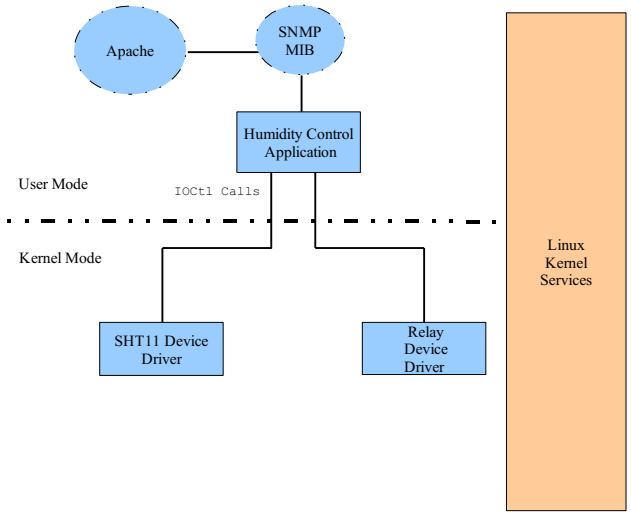


Figure 3. Software Architecture

The TS-7200 single-board computer (SBC) runs Linux on an ARM9-based processor. The system's software architecture is shown in Figure 3. Two Linux drivers are required: one to sense the humidity (and temperature, which came almost free) and the second to control the position of the relays. A user-mode application on top of the drivers periodically polls the humidity and temperature data, and controls the relay position depending on SNMP MIB configuration settings. The SNMP MIB is managed by the Linux snmpd dæmon. The SNMP MIB also serves as the basic bridge to an Apache custom module that exposes the MIB data to a Web browser for control and monitoring of the entire humidity control system. Each component of the humidity control system is described in more detail later in this article.

Linux Device Drivers

The two required Linux drivers, which I designed as loadable modules, are rather basic as far as Linux drivers go. They both are character devices with ioctl interfaces that provide access to the SHT11 sensor and control of the power relays. The SHT11 driver requires only two ioctl functions:

- SHT1X_IOC_READ_HUMIDITY: read the current SHT11 humidity.
- SHT1X_IOC_READ_TEMPERATURE: read the current SHT11 temperature.

With both the temperature and humidity, I have the option of calculating the dew point (even though the system is indoors, and the last thing I expect is dew to form on the components). The SHT11 driver reads humidity and temperature

Listing 1. Generate SHT11 Start Transmission Sequence

```
void writeSHT1xTransmissionStartSequence(void)
{
     writeSHT1xOne(DATA_SHT);
     writeSHT1xZero(SCK_SHT);
     udelay(2);
     writeSHT1xOne(SCK_SHT);
     udelay(2);
     writeSHT1xZero(DATA_SHT);
     udelay(2);
     writeSHT1xZero(SCK_SHT);
     udelay(2);
     udelay(2);
     udelay(2);
     writeSHT1xOne(SCK_SHT);
     udelay(2);
     writeSHT1xOne(DATA_SHT);
     udelay(2);
     writeSHT1xZero(SCK_SHT);
     udelay(2);
}
```

using a two-wire interface that is well defined in the Sensirion SHT11 data sheet. The clock has no real minimum frequency, but has a maximum frequency of 10MHz. I had no reason to run the clock at the maximum rate. In fact, the messages required to transfer the temperature and/or humidity data are so short, the clock rate could be anything within reason, so I decided to run the clock at 250KHz.

Accessing the SHT11 is relatively straightforward. A start and end sequence for each transfer is achieved using a prescribed combination of data and clock discrete I/O transitions.

Listing 2. Transmit Command Sequence

```
void writeSHT1xCommand(int iMode)
{
    unsigned char  ucBitToCheck;
    unsigned char  ucAckBit;

    driveDataLine(DATA_SHT);
    /* All 3 address bits always zero
     * so start with those */
    writeSHT1xZero(DATA_SHT);
    udelay(2);
    writeSHT1xOne(SCK_SHT);
    udelay(2);
    writeSHT1xZero(SCK_SHT);

    writeSHT1xZero(DATA_SHT);
    udelay(2);
    writeSHT1xOne(SCK_SHT);
    udelay(2);
    writeSHT1xZero(SCK_SHT);

    writeSHT1xZero(DATA_SHT);
    udelay(2);
    writeSHT1xOne(SCK_SHT);
    udelay(2);
    writeSHT1xZero(SCK_SHT);

    /* Now transmit the 5 command bits,
     * in the order of MSb to LSb */
    for (ucBitToCheck=0x10; ucBitToCheck != 0;)
    {
        if (iMode & ucBitToCheck)
            writeSHT1xOne(DATA_SHT);
        else
            writeSHT1xZero(DATA_SHT);
        udelay(2);
        writeSHT1xOne(SCK_SHT);
        udelay(2);
        writeSHT1xZero(SCK_SHT);
        ucBitToCheck >>= 1;
    }

    /* Now tri-state the data DIO so the
     * device can ACK the transfer */
    tristateDataLine(DATA_SHT);
    udelay(2);
    writeSHT1xOne(SCK_SHT);
    udelay(2);
    ucAckBit = readSHT1x(DATA_SHT);
    writeSHT1xZero(SCK_SHT);
    mdelay(250);
}
```

Listing 3. Read Sensor Data

```
unsigned int readSHT1xData(void)
{
    int            iLoop;
    unsigned int   uiBitRead;
    unsigned int   uiMSB=0;
    unsigned int   uiLSB=0;
    unsigned int   uiRetValue;

    /* Read MSB from SHT1x */
    for (iLoop = 0; iLoop < 8; iLoop++)
    {
        uiMSB <<= 1;
        writeSHT1xOne(SCK_SHT);
        uiBitRead = readSHT1x(DATA_SHT);
        udelay(2);
        writeSHT1xZero(SCK_SHT);
        udelay(2);
        if (uiBitRead)
            uiMSB |= 1;
    }
    /* Acknowledge sequence; must drive data
     * line as it is tri-stated at this point */
    driveDataLine(DATA_SHT);
    writeSHT1xZero(DATA_SHT);
    udelay(2);
    writeSHT1xOne(SCK_SHT);
    udelay(2);
    writeSHT1xZero(SCK_SHT);
    tristateDataLine(DATA_SHT);
    udelay(2);
    /* Read LSB from SHT1x */
    for (iLoop = 0; iLoop < 8; iLoop++)
    {
        uiLSB <<= 1;
        writeSHT1xOne(SCK_SHT);
        uiBitRead = readSHT1x(DATA_SHT);
        udelay(2);
        writeSHT1xZero(SCK_SHT);
        udelay(2);
        if (uiBitRead)
            uiLSB |= 1;
    }
    /* Don't acknowledge last byte so the device
     * doesn't transmit the 8-bit CRC as it isn't
     * really necessary for this application */
    uiRetValue = u8tou16(uiMSB, uiLSB);

    return(uiRetValue);
}
```

Listing 4. Read Humidity Sequence

```
writeSHT1xTransmissionStartSequence();
writeSHT1xCommand(SHT1x_MEASURE_HUMIDITY);
uiHumidity = readSHT1xData();
```

For example, in order to request the current humidity or temperature, a start of transmission sequence is issued that consists of the sequence of data and clock transitions as shown in Listing 1.

In Listing 1, note the use of udelay kernel calls. The timing requirements of the SHT11 two-wire access is satisfied using delays in the microseconds and, in some cases, milliseconds. This is most easily achieved using the kernel udelay call, and when millisecond delays are required, the mdelay call. I suppose there are some developers who shudder at the use of busy loops, but remember, this is a dedicated, embedded system. It does nothing but read humidity and check whether relays need to be switched on or off, and it repeats this forever.

After the start transmission sequence, the driver is free to write an 8-bit command sequence that identifies the operation to the humidity sensor, such as measure the humidity or temperature. A second procedure actually transmits the 8-bit command sequence and is shown in Listing 2.

Listing 2 not only demonstrates the bit-twiddling necessary to drive a two-wire interface solely with software, but it also reveals how the sensor acknowledges receipt of a valid command. The data DIO must be tri-stated (that is, not driven to either a 0 or a 1 by the ARM) in order for this two-wire interface to permit slave devices, such as the SHT11, to transmit back to the two-wire interface master—in this case, the SHT11 device driver in the ARM. In addition, note that the last line of code in the procedure will cause a 250-millisecond delay. This is because the SHT11 takes a good deal of time, relatively speaking, to measure either the temperature or humidity. The specification requires 210 milliseconds for the most accurate form of measurement, with a +–15% tolerance. This puts the worst-case delay at 241.5 milliseconds, which I increased to 250 milliseconds, just to be safe.

The third and final required piece of code necessary to read data from the SHT11 humidity sensor is shown in Listing 3. The Read Sensor Data procedure will read 16 bits of data from the sensor after it has measured either the humidity or the temperature. The SHT11 has the option of sending an 8-bit CRC at the end of the 16 bits of data, but I opted not to check the CRC, as it is unlikely the data ever will be corrupted due to environmental effects in my music room.

The procedures shown in Listings 1, 2 and 3 form the core of the SHT11 two-wire interface device driver code. When the driver receives an ioctl requesting the humidity, the three instructions shown in Listing 4 are all that is needed to read the current humidity from the sensor.

The second device driver controls the relays and switches the 120V AC and neutral lines to the humidifier and dehumidifier. The ioctl interface for the relay driver required the following ioctl functions:

- RELAY8_IOC_READ_RELAYS: read the current relay settings.

Listing 5. Read/Write Relays

```
unsigned char readRelay8(int iRelay8Address)
{
    /* Read Relay8 register and return the value */
    return(readb(iRelay8Address));
}

void writeRelay8(int iRelay8Address, unsigned char ucValues)
{
    /* Write Relay8 register with the values */
    writeb(ucValues, iRelay8Address);
}

// Excerpt from ioctl function:
    switch(cmd) {

    case RELAY8_IOC_READ_RELAYS:
        /* Read Relay8 relay values */
        ucRelayValues = readRelay8(relay8_base + RELAY8_CONTROL);
        if (copy_to_user((typeof(relay8_relayValues)) arg,
                         &ucRelayValues,
                         sizeof(relay8_relayValues))) {
            ret = -EFAULT;
        }
        break;

    case RELAY8_IOC_WRITE_RELAYS:
        /* Write Relay8 relay values */
        writeRelay8(relay8_base + RELAY8_CONTROL,
                    *(typeof(relay8_relayValues)) arg);
        break;

    default:
        ret = -ENOTTY;
    }
```

- RELAY8_IOC_WRITE_RELAYS: set the relays to the supplied state.

Reading the relay settings is used to ensure that the relays are in the desired position. The relay hardware actually includes eight relays, and all eight relay values are written in one shot. The data register used to control and report the relay positions consists of one 8-bit register. This register either is read to report the current relay settings or written to change the relay settings. Unlike the SHT11 driver, the relay driver can affect a change in a relay state with one writeb Linux driver C instruction. Listing 5 shows the relay read and write procedures, along with an excerpt from the ioctl processing that differentiates between read and write. It doesn't get much simpler than this!

User-Mode Application

I wrote a user-mode application that periodically polls the SHT11 device driver for the current humidity and temperature using the ioctl SHT1X_IOC_READ_HUMIDITY and SHT1X_IOC_READ_TEMPERATURE, respectively. Depending on

the desired humidity setting, the application determines whether the current humidity is either too high or too low, taking into account the tolerance of plus or minus 3.5% rH. If an actionable event is determined, the specific relays are turned either on or off using the relay device driver RELAY8_IOC_WRITE_RELAYS ioctl function. For example, when the user-mode application reads the humidity and determines that the humidifier must be turned on, it issues an ioctl RELAY8_IOC_WRITE_RELAYS function to switch on both relays that are dedicated to the 120V A/C and neutral lines of the humidifier. At the same time, the application also ensures that the two relays associated with the 120V A/C and neutral lines of the dehumidifier are switched off. Relay control can be one of three options: 1) the humidifier is turned on, and the dehumidifier is turned off; 2) the dehumidifier is turned on, and the humidifier is turned off; or 3) both the humidifier and dehumidifier are turned off. The application never turns both the humidifier and dehumidifier on at the same time. The application is loaded at Linux boot time and, like most embedded applications, runs perpetually.

Along with controlling the humidifier and dehumidifier relays, the application accumulates and saves statistics. In this control system, the actual data that is acted upon is

To finish off the humidity controller, I added a Web page interface that includes a recipe that uses a tad of HTML, a smattering of JavaScript and a pinch of AJAX with server-side scripting to create an end-user browser interface.

required to be persistent—that is, the humidity data must be saved somewhere for later use. The user-mode application is responsible for saving the data for later use by a browser, and it does so with the use of the SNMP (Simple Network Management Protocol) support provided by the net-snmp Linux package.

SNMP is a standard set of protocols and policies for managing networks and devices. The net-snmp implementation of SNMP consists of an agent, which runs as a Linux dæmon snmpd, and a database called a Management Information Base, or MIB. A MIB is structured as a tree, with branches grouping together similar items. I extended the standard Linux MIB that is shipped with the net-snmp package and added a new branch off of the "enterprises" node, which includes all the humidity controller items that I need (Figure 4). The snmpd agent acts on the MIB at the request of SNMP clients—that is, the agent reads/writes data from/to the MIB on behalf of client get and set requests. In this architecture, there are two clients: the user-mode application and the Web browser.

In order to adapt SNMP to any application, a MIB must be defined in a standard MIB ASN.1 format. I defined a MIB for my humidity controller and called it

Listing 6. mib2c-Generated C Code

```
netsnmp_register_scalar(
    netsnmp_create_handler_registration(
        "targetHumidity",
        handle_targetHumidity,
        targetHumidity_oid,
        OID_LENGTH(targetHumidity_oid),
        HANDLER_CAN_RWRITE));


int
handle_targetHumidity(netsnmp_mib_handler           *handler,
                      netsnmp_handler_registration  *reginfo,
                      netsnmp_agent_request_info    *reqinfo,
                      netsnmp_request_info          *requests)
{
    switch (reqinfo->mode) {

    case MODE_GET:
        break;

    case MODE_SET_RESERVE1:
        break;

    case MODE_SET_COMMIT:
        break;
    }

    return SNMP_ERR_NOERROR;
}
```

HUMIDITYCONTROLLER-MIB, which gets loaded when the snmpd dæmon runs during the Linux boot process. The MIB contains data items that are represented by object identifiers, or OIDs. An example of an OID definition from my MIB for the humidity controller targetHumidity variable is shown below:

```
targetHumidity OBJECT-TYPE
    SYNTAX      Integer32(0..2147483647)
    MAX-ACCESS  read-write
    STATUS          current
    DESCRIPTION
    "Target humidity."
    ::= { humidityEntry 3 }
```

The previous ANS.1 MIB definition phrase generates an OID with the value .1.3.6.1.4.1.2200.2.3. This rather cryptic-looking sequence of numbers is a scheme used to identify a leaf in the MIB tree. The branch that I added to the enterprises node is identified by the integer 2200. Under the 2200 node is the node identified by a 2, which contains all of the overall humidity controller items that the system needs. The leaf node identified by a 3 is the targetHumidity.

The Linux SNMP package contains a very useful tool called mib2c. mib2c takes a MIB definition, such as HUMIDITYCONTROLLER-MIB, and generates C code that

can be used to extend the standard Linux snmpd agent. Several options exist when generating code with mib2c. I used the more general option for generating C code from a MIB with the mib2c.scalar.conf configuration, which causes code to be generated for general-purpose scalar OIDs, as opposed to table-based OIDs. The generated C code is used by the snmpd dæmon. Listing 6 is a distilled example of the generated C code from mib2c for the targetHumidity OID that shows the code framework needed to support the SNMP get (MODE_GET) and SNMP put (MODE_SET_RESERVE1 and MODE_SET_COMMIT) operations.

The code shown in Listing 6 makes reference to the generated callback procedure, handle_targetHumidity, which is supplied in skeletal form only by mib2c. Not much code is needed in order to support scalar OIDs, which the humidity controller uses exclusively. Anytime a specific OID, in this case the targetHumidity OID .1.3.6.1.4.1.2200.2.3, has an operation performed, the snmpd dæmon will invoke this callback procedure with an indication of the requested operation being performed on the OID.

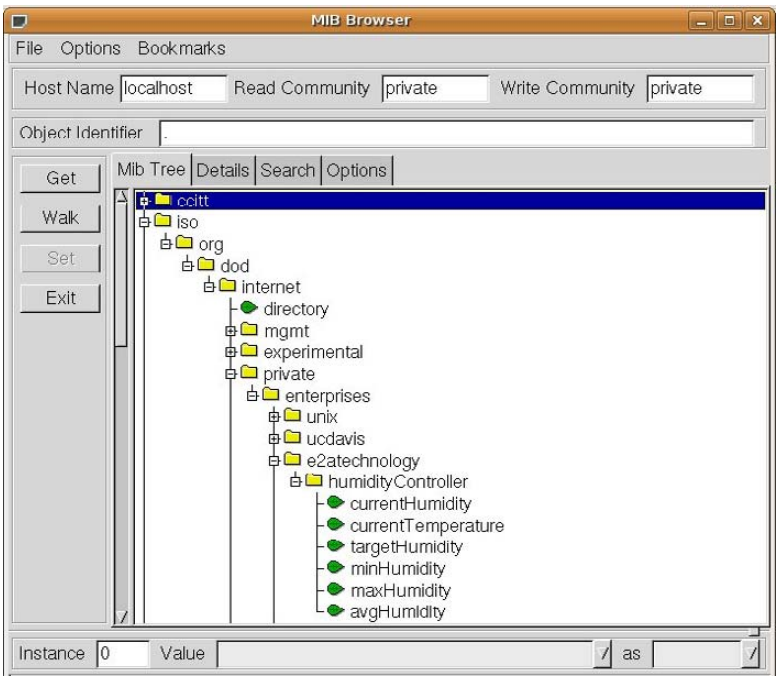


Figure 4. mbrowse Screenshot

I rebuilt the snmpd dæmon so that the newly created humidity controller MIB structure and generated framework code could be supported. Before rebuilding the snmpd dæmon, the new MIB must be configured into the build environment. This is accomplished easily with the following command:

```
$ ./configure --with-mib-modules="humidityController"
```

Once configured, the entire net-snmp package was rebuilt with the make command. Once the snmpd dæmon was rebuilt, I tested the new MIB structure by using the net-snmp command-line interface utilities snmpset and snmpget. For example, in order to set the targetHumidity OID to 50% rH, the following command can be issued:

```
$ snmpset -Ovqe -v 1 -c private localhost targetHumidity.0 i 50
```

Note the use of relative, symbolic OIDs in the snmpset command. The actual OID .1.3.6.1.4.1.2200.2.3 could be used as well, because it's statically defined and should never change. But, I prefer symbolic references where possible, as it helps in readability. The -Ovqe switch controls the output format that results from the snmpset. Although I built the net-snmp package to support all three major versions of SNMP (1, 2 and 3), I really needed only basic version 1 support, which is why the -v 1 switch appears. The SNMP community string is indicated by the -c private switch and appears in set operations because only private communities are permitted to set OID values (this is a one-time option when the snmpd dæmon is configured).

The humidity controller MIB can be viewed with a tool included in net-snmp called mbrowse. mbrowse is a GUI that bolts onto the system MIB structure and permits manipulation of specific OIDs. Figure 4 shows a screenshot of mbrowse and the humidity controller MIB tree branch.

Once the snmpd dæmon was complete with support for the newly added humidity controller OIDs, I was able to complete the user-mode application code. Listing 7 contains the complete user-mode application, and it is too long to print here, but it is available on the *LJ* FTP site (see Resources). It is very typical of an embedded application, as it perpetually reads data and then takes actions on the data. Note the use of snmpget and snmpset. The net-snmp

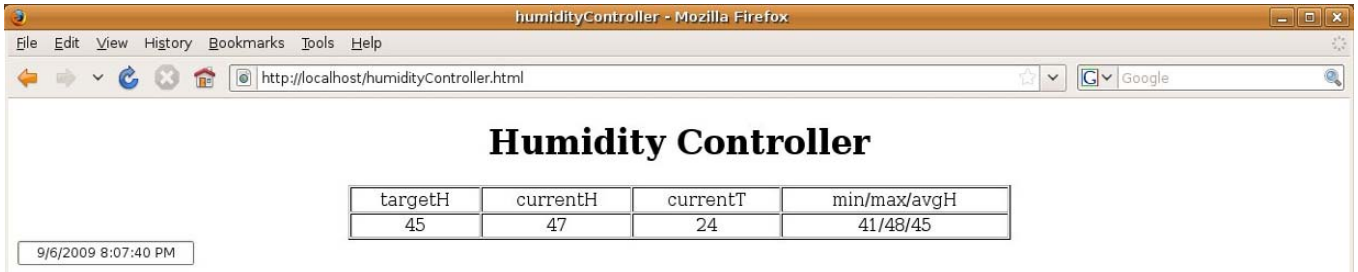


Figure 5. Humidity Controller and Firefox

Listing 8. Perl Script setTargetHumidity

```
use CGI;
$query = new CGI;
$targetH = $query->param('targetH');
$SNMP_SET_CMD = "snmpset -v 1 -c private";
$SNMP_TARGET = "localhost";
$SNMP_TARGETHUM_OID = "targetHumidity .0";
$SNMP_TYPE = "i";

chomp($retVal = `${SNMP_SET_CMD} ${SNMP_TARGET}
      ➥$SNMP_TARGETHUM_OID $SNMP_TYPE $targetH`);
```

Figure 6. Completed Humidity Controller with Humidifier and Dehumidifier Connected

package does include APIs for both C and Perl, but I decided it was simpler to leverage the existing snmpget and snmpset utilities.

To finish off the humidity controller, I added a Web page interface that includes a recipe that uses a tad of HTML, a smattering of JavaScript and a pinch of AJAX with server-side scripting to create an end-user browser interface. The humidity controller in a Firefox browser looks like what is shown in Figure 5. The targetHumidity (targetH) cell in the table has a JavaScript function associated such that editing is possible, and when a new value is entered, it is POSTed to Apache. Apache will invoke a Perl script to set the target humidity in the SNMP MIB. Listing 8 is an excerpt from the Perl code that shows the SNMP actions. The other cells are read-only and are refreshed periodically with values from the SNMP MIB with the help of a second Perl script, humidityController.cgi. This second Perl script pushes out only the data necessary to generate the table of values shown in Figure 5.

Figure 7. Carlos Pina Classical Guitar

The humidity controller (Figure 6) has been keeping my music room within a humidity range that makes my Carlos Pina classical guitar quite happy (Figure 7). The work involved to build the system was a real pleasure. But the best part is sitting down to play the opening arpeggios in Bach's *Prelude* and hearing the notes ring true without retuning my guitar. It not only makes me smile, but I think it would make Bach smile as well.■

Jeffrey Ramsey has been an embedded developer his entire career, and when not pouring through Linux kernel and driver source code, he can be found plucking a guitar. Jeffrey can be contacted at jeffreyramsey@e2atechnology.com.

Resources

Listing 7 (the Complete User-Mode Application):
ftp.linuxjournal.com/pub/lj/listings/issue188/10534.tgz

TS-7200 Main Board (from Technologic Systems):
www.embeddedarm.com/products/board-detail.php?product=TS-7200

TS-RELAY8 Daughter Card (from Technologic Systems):
www.embeddedarm.com/products/board-detail.php?product=TS-RELAY8

TS-ENC720 Enclosure (from Technologic Systems):
www.embeddedarm.com/products/board-detail.php?product=TS-ENC720

Wii Will Rock Linux

Why should your Wii have all the fun? Find out how to connect all those *Rock Band* instruments to your Linux machine and use them with a number of different audio programs. KYLE RANKIN

In my August 2008 column, I wrote about how to use a Wiimote from a Nintendo Wii on a Linux system as a general-purpose wireless joystick. In that column, I covered how to bind use buttons not only on the Wiimote, but also on the Nunchuck and Classic Controller, so that you could use them with a number of different video game emulators. Well, since that column, *Rock Band* for the Wii was released, and with it three extra peripherals: a wireless guitar, a microphone and a drum set.

Everyone knows that only old people play real guitars, so I couldn't pass up the opportunity to rock out with an entire band of plastic instruments on my Wii. I hadn't read too much beforehand about the instruments that came with Wii's *Rock Band*, so when I unpacked everything, I was surprised to note that all three instruments were connected to the Wii via USB. That left me with only one question, do these instruments work in Linux?

It turns out that not only do all three *Rock Band* instruments work in Linux, they also all work with very little extra effort. In this column, I describe how to configure Linux to see these instruments and highlight some applications you can use them with.

The Microphone

Probably the simplest instrument to get working with Linux was the microphone. The moment I plugged it in, I got dmesg output that identified it:

```
[  188.006918] usb 1-1: new full speed USB device
    using uhci_hcd and address 2
[  188.132102] usb 1-1: configuration #1 chosen
    from 1 choice
[  188.474088] usbcore: registered new interface
    driver snd-usb-audio
```

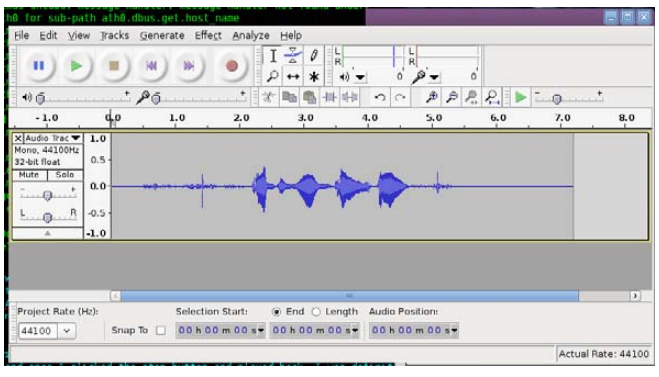


Figure 1. Audacity

I then fired up Audacity, one of my favorite sound recording programs, to see if the microphone would work. By default, Audacity was set to my system microphone, so I clicked Edit→Preferences, and under the recording section of the Audio I/O window I chose ALSA: Logitech USB Microphone: USB Audio from the Recording Device drop-down menu. I also changed it to be a Mono device.

After I clicked OK to accept my changes, I clicked the big red Record button on the main Audacity window and started talking. I was able to see my voice in the output immediately, and once I clicked the Stop button and played it back, I definitely was able to hear myself.

The Guitar

Considering how easy it was to use the microphone, I wondered what Linux would make of the wireless guitar. It appeared to connect a lot like a wireless mouse or keyboard with a small USB dongle that had a connect button you could use to sync with the wireless device. When I connected the dongle, I could see in the dmesg output that my Ubuntu Hardy install had detected the device as some sort of USB Human Interface Device (HID):

It turns out that not only do all three *Rock Band* instruments work in Linux, they also all work with very little extra effort.

```
[  775.322361] usb 1-1: new full speed USB device
    using uhci_hcd and address 3
[  775.369009] usb 1-1: configuration #1 chosen
    from 1 choice
[  775.525791] usbcore: registered new interface
    driver hiddev
[  775.531822] input: Licensed by Nintendo of America
    Harmonix Guitar Controller for Nintendo Wii as
    /devices/pci0000:00/0000:00:1d.0/usb1/1-1/
➥1-1:1.0/input/input10
[  775.545411] input,hidraw0: USB HID v1.11 Gamepad
    [Licensed by Nintendo of America  Harmonix Guitar
    Controller for Nintendo Wii] on usb-0000:00:1d.0-1
[  775.545444] usbcore: registered new interface
    driver usbhid
[  775.545451] /build/buildd/linux-2.6.24/drivers/hid/
➥usbhid/hid-core.c: v2.6:USB HID core driver
```

It appeared like a new gamepad device had been installed under /dev/input/js0, so I used the useful jstest utility (packaged by a number of distributions) to test whether the buttons on the guitar generated events. To use jstest, simply execute the program with the joystick device to test as an argument (in my case, /dev/input/js0). Each time a joystick event is registered, the output in the terminal updates. The four lines shown in Figure 2 are examples of the output when I pressed and released the green and red buttons on the guitar, respectively.

If you compare the lines, you can see that the green button corresponded to button 1, and the red button corresponded to button 2. Because the guitar interfaces directly with Linux like a regular joystick device, that means I can use its buttons with any game that supports joysticks.

Of course, probably the best game for the *Rock Band* guitar on Linux is *Frets on Fire*. *Frets on Fire* is an open-source guitar game written in Python and packaged for a number of distributions and operating systems. By default, it is designed to be used with your regular keyboard held in your hands somewhat like a guitar. The F1–F5 keys are frets on the guitar, and the Enter key can be used to strum. That works okay, but it certainly is nicer to use a guitar intended for the purpose, and sure enough, *Frets on Fire* supports remapping the default keyboard keys to joystick buttons.

Of course, probably the best game for the *Rock Band* guitar on Linux is *Frets on Fire*.

To configure *Frets on Fire* for my guitar, all I needed to do was start the game, go into Settings and then modify the key settings. I just went through each key configured for the game, selected it, and then when it asked me to press a new key to set it to, I chose the corresponding key on the guitar. After you change the keys in this method, you will notice that you can navigate the *Frets on Fire* game completely from your guitar. You can strum up or down to move through the menus and use the green button to make selections.

The Drums

The final *Rock Band* instrument is also my favorite—the drums. Although you could argue, I suppose, that the microphone is the closest to a real instrument in the game, the drums feel the most real to me. The big question, of course, was whether the drums registered in Linux. Upon connecting the drums to my machine, I had hope from the dmesg output:

```
[  400.997524] usb 1-1: new full speed USB device
    using uhci_hcd and address 7
[  401.059524] usb 1-1: configuration #1 chosen
    from 1 choice
```

Figure 2. jstest Output

Figure 3. *Frets on Fire* Key Settings

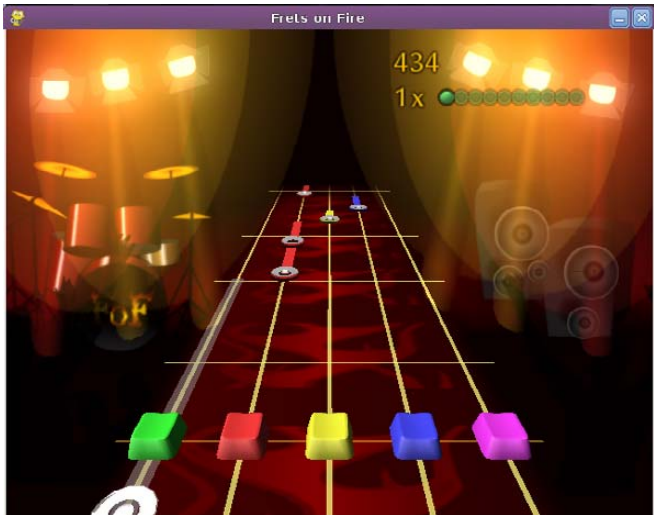


Figure 4. *Frets on Fire* Gameplay

```
[  401.078667] input: Licensed by Nintendo of America
    Harmonix Drum Controller for Nintendo Wii as
    /devices/pci0000:00/0000:00:1d.0/usb1/1-1/
➥1-1:1.0/input/input14
[  401.104320] input,hidraw0: USB HID v1.11 Gamepad
    [Licensed by Nintendo of America  Harmonix Drum
    Controller for Nintendo Wii] on usb-0000:00:1d.0-1
```

It turns out the drums show up as a joystick device, just like the guitar. I ran jstest (as with the guitar), pointed to the new joystick device, hit a few of the drum pads, and was able to see that they definitely generated button events. Specifically, I saw that blue was button 0, green was button 1, red was button 2, yellow was button 3, and the foot pedal was button 4.

Now, although I could presumably use the drums with *Frets on Fire*, or really any game that supported joysticks, unfortunately, I wasn't able to find a free game for Linux

Figure 5. Hydrogen with the Pattern Editor Window Selected

that specifically used the drums. Instead, I found something arguably better: a free Linux drum kit program called Hydrogen that lets you create your own drum tracks and can interface with the keyboard or a MIDI device. Hydrogen was packaged for my distribution, or alternatively, you can download and build it from the official site. Unfortunately, the Wii drum kit doesn't act as a MIDI device, and Hydrogen isn't set up to accept input from a joystick. Hydrogen does allow you to use keys on the keyboard to activate different parts of the drum kit, so I had to figure out a way to map the joystick buttons to key events. Lucky for me, such an application already exists called joy2key. joy2key is a pretty basic program. You run the program on the command line and tell it which joystick to use and which keys to map to particular joystick buttons. Then, you can click on the application to bind it to, and it will send all joystick events to that particular window.

joy2key also already was packaged by my distribution, and after it installed, I simply had to choose to which keys to bind buttons. The first five drum types are activated in Hydrogen by the Z, S, X, C and D keys, respectively. So, first I launched Hydrogen, and then in a terminal, I typed:

```
joy2key -X -buttons d c s x z -dev /dev/input/js0
➥-thresh 0 0 0 0 0 0 0 0 0 0 0 0
```

In addition to the -buttons option, the -X option tells joy2key to send X events. The -dev option points it to your joystick device, and the -thresh option sets the low and high thresholds to trigger events for each button. If you don't specify -thresh, joy2key prompts you to set the values each time you run it, and as these buttons are either on or off, I just set them to zero. After you run this command, your mouse icon should turn into a cross. Click on the Hydrogen window, and then joy2key will start sending events to Hydrogen.

The order of drum sounds and how they correspond to keys is set in the Hydrogen pattern editor (Figure 5). There are any number of different ways to arrange the sounds and button mappings, but probably the easiest order to keep straight sets the pattern editor as though you played across the Wii drum kit starting at the foot pedal. By default, this probably won't be set correctly to suit the joy2key settings, so click a particular drum sound to highlight it, and then press the up/down arrows on the top of that column to rearrange its order. On the bottom, put Kick, then a Snare, then a Hi Hat (like Open HH), then a Tom, then a Cymbal (Crash). Once you have arranged these sounds, hit some of the drum pads on the drum kit, and you should hear their corresponding sounds on your computer. Go ahead, play a drum solo or two to get accustomed to the current pattern.

Hydrogen is a complicated enough program to warrant its own article, but here are some of the many things you can do now that the Wii drum set works with it. For one, Hydrogen includes a number of different drum set samples from which you can choose, and you even can create your own, so you can experiment with a lot of different sounds for your drums. In addition, you also can use your drum set when recording different beat patterns. Finally, if you want, you could just hook up your computer to a loud set of speakers and start playing. Hydrogen includes a mixer for each sound, so you can adjust the relative volumes.

Hydrogen does allow you to use keys on the keyboard to activate different parts of the drum kit, so I had to figure out a way to map the joystick buttons to key events.

Well, if you weren't already tempted to buy a set of *Rock Band* instruments just for your Wii, now you have another excuse...er, reason...why you need them. It's a testament to how far Linux has progressed that you can get random devices like these working on your computer with minimal effort. As for me, I'm going to switch up the drum patterns in Hydrogen so that they feature more cowbell.■

Kyle Rankin is a Senior Systems Administrator in the San Francisco Bay Area and the author of a number of books, including *Knoppix Hacks* and *Ubuntu Hacks* for O'Reilly Media. He is currently the president of the North Bay Linux Users' Group.

Resources

Audacity: **audacity.sourceforge.net**

Frets on Fire: **fretsonfire.sourceforge.net**

Hydrogen: **www.hydrogen-music.org**

Using wview

If knowing what the weather is gets your blood pumping and your heart racing, you need to get wview and hook it up to your other great passion, which is, of course, Linux.

MARK TEEL

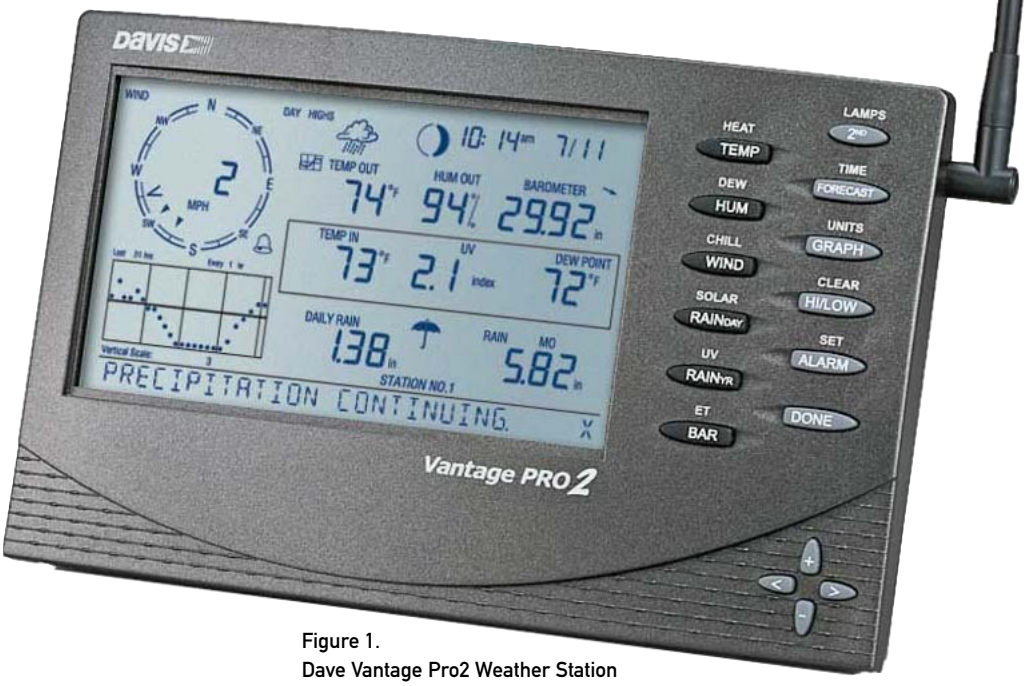


Figure 1.
Dave Vantage Pro2 Weather Station

Are you fascinated with weather? Do you often find yourself checking local weather conditions? Is the weather your favorite part of the news broadcast? If so, you may be a weather geek, and wview may be the application for you.

wview is an open-source weather application that retrieves sensor readings from a weather station. The sensor data is stored in SQLite3 databases. Aggregate data, such as minimums, maximums and averages, are computed and stored in the database back end. Optional uses of the stored data include weather Web site generation; generic file generation for external applications; data submission to third-party organizations, including Citizen Weather Observer Program (CWOP) and Weather Underground; and store-forward to remote data collection centers. A user-friendly HTML interface is provided for configuring your weather station as well as for optional features.

To set up your weather station and publish your data with wview, you need a weather station. Supported stations include Davis Vantage Pro/Pro2 (Figure 1) or Vantage Vue, Texas Weather Instruments, Vaisala WXT510/520, Oregon Scientific WMR9X8 and La Crosse WS-23XX. Next, you need a platform to host the wview application. Desktop computers of any vintage work well, but it often is desirable to host wview on a low-power, unattended system. The now discontinued Linksys NSLU2 has been a popular choice. The new SheevaPlug quickly is gaining popularity as a wview host also. Industrious people even have used a Western Digital Worldbook NAS as their wview host. Because wview is modular and designed for embedded applications, it can be hosted on low-horsepower systems.

Next, you need to install a Linux distribution of your choice. The Debian (and derivatives) wview packages provide the most idiot-proof installation path, but source installs also are straightforward for any Linux distribution.

Finally, you need an interface cable. This may be a simple 9-pin serial cable or perhaps a USB-serial adapter if your host has no serial ports.

Configuration

To configure wview, open your favorite browser and point

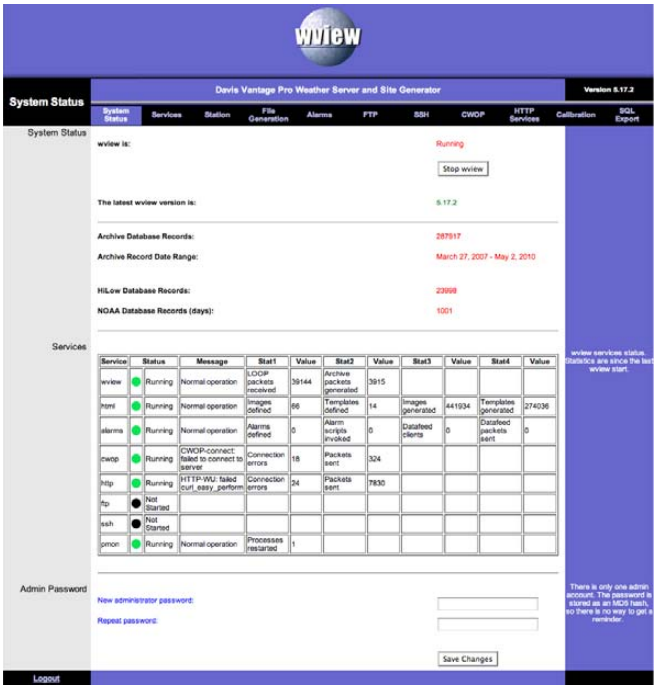


Figure 2. System Status

it to the wview management Web site, typically http://[your_wview_server]/wviewmgmt/login.php. An HTTP server is required on the wview host (this will be installed automatically if you use the APT packages). Use the default administration password "wview" (you can change this later). After logging in successfully, the System Status page is displayed (Figure 2). The System Status page displays the current state of all wview services as well as other status information.

Configuration is broken up into logical sections with context-sensitive help available by mousing over the configuration items. Click the Station tab to configure the station parameters (Figure 3).

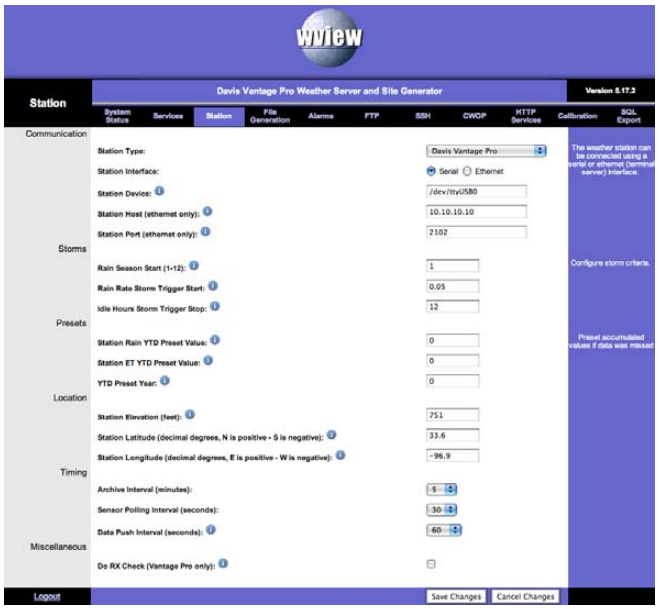


Figure 3. Station Configuration

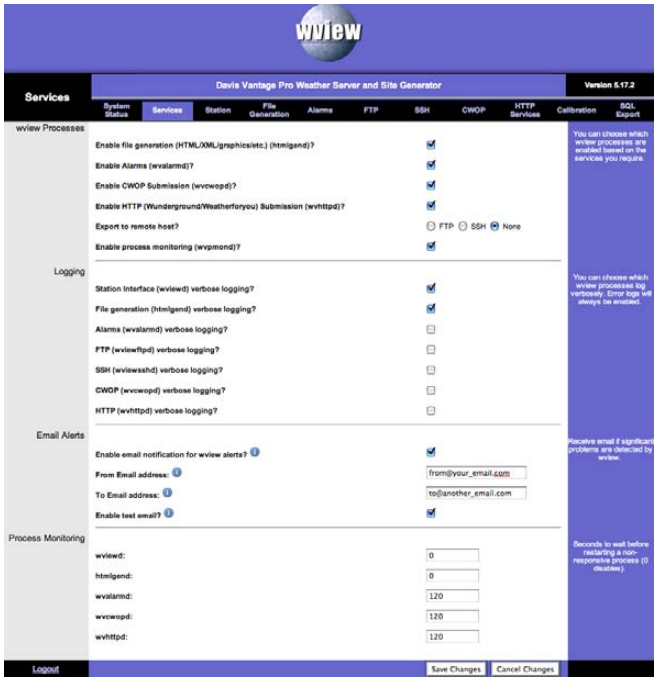


Figure 4. Services Configuration

The critical parameters here are the station type and the interface characteristics. Select Save Changes when you are done. Next, click the Services tab (Figure 4).

This page provides the configuration of wview services, log verbosity for the services and e-mail alerts. Services available are File Generation, Alarms, CWOP, HTTP (Weather Underground and Weatherforyou), File Export (SSH or FTP) and Process Monitoring. For now, let's not enable any additional services until you have confirmed your station interface.

Station Confirmation

Now, let's proceed to the station interface verification. Open a shell on the system that is running wview, so you can follow updates to the system log. At the prompt, enter the following:

```
$ sudo tail -f /var/log/syslog
```

This displays new system log messages as they are generated. Here, you will monitor wview startup and status messages. Open another shell, and execute the following:

```
$ sudo /etc/init.d/wview stop
$ sudo /etc/init.d/wview start
```

You will see a flurry of activity in the system log from the wview processes as they start up. It is a good idea to familiarize yourself with these wview log messages, as a wealth of detail is included that can be very helpful.

wview is an open-source weather application that retrieves sensor readings from a weather station.

Return to the System Status page and observe the status of the station interface and the file generation. If both are not status "green" and "Running", further investigation in the system log file will be required to find any configuration or station interface issues.

Default Web Site

If all is well, you now can view the default wview weather site. This is typically found at http://[your_server_url]/weather/index.html (Figure 5).

Current conditions are given in the table on the left and by the dials in the center and on the right. These values are updated every time station data is polled (default is 30 seconds). The weather site pages are regenerated every 60 seconds (configurable). Observing changes in the current conditions is an easy way to confirm proper station interface operation.

Historical data for the last 24 hours are presented as graphs. Graphs of the last 24 hours, the last 7 days, the last 28 days and the last 365 days are available on other site pages.

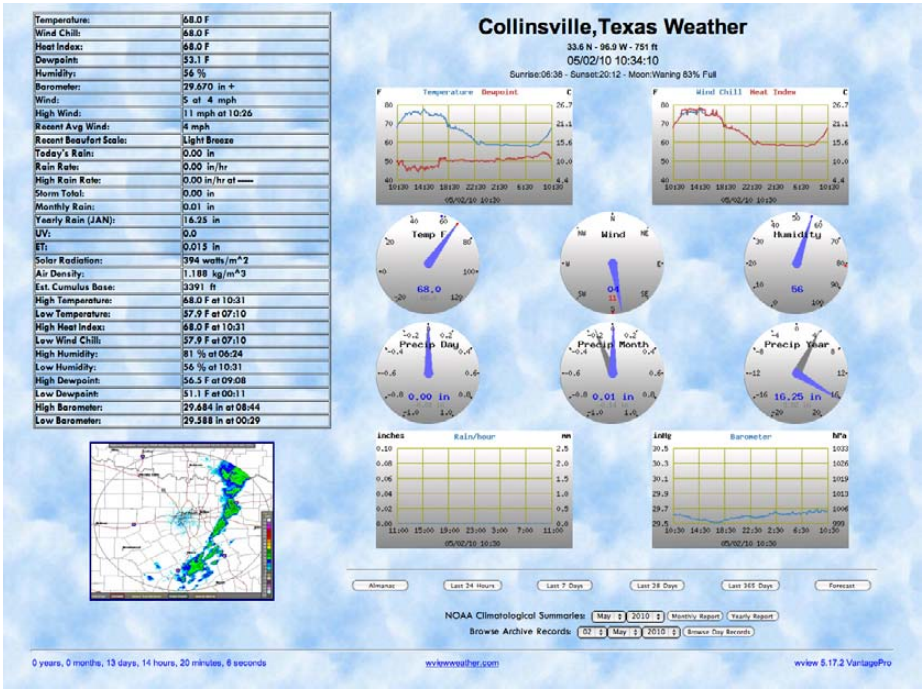


Figure 5. Default Web Site

Citizen Weather Observer Program (CWOP)

CWOP is a system by which individuals with weather stations and the proper software can submit their weather data to an APRS-based data storage system, so that others, including NOAA (National Oceanic and Atmospheric Administration), can use the data however they see fit. There are some really neat station display Web sites, including some Java apps to look up station data, position, maps and so on. See **www.findu.com/cgi-bin/wxpage.cgi?call=CW4097** for an example weather station.

When CWOP support is enabled and configured properly, wview transmits a new WX packet to the APRS server every ten minutes, based on the last digit of your callsign.

CWOP participation requires registering for an APRS callsign. Once you have configured wview for CWOP properly and confirmed your data on-line, you must contact the maintainers via e-mail to confirm your registration. Then your data will be available for anyone to see and possibly be used in NOAA forecast models and so on.

When CWOP support is enabled and configured properly, wview transmits a new WX packet to the APRS server every ten minutes, based on the last digit of your callsign.

wview supports the APRS-IS Rollover (Automatic Packet Reporting System-Internet Service) functionality by enforcing the definition of three APRS-IS server:port pairs. The goal is to avoid data loss to the CWOP system caused by connection errors. Select three different servers from the list at **www.wxqa.com/activecwd.html**.

Click the Services tab and enable CWOP submission and CWOP verbose logging. Click Save Changes. Next, click the CWOP tab, and enter your callsign, latitude and longitude (see the mouse-over help for format details), the CWOP servers (three should be entered) and port numbers. Go ahead and enable "Log CWOP Packet?"; you can disable it after submission is confirmed. Click Save Changes.

Now, restart wview:

```
$ sudo /etc/init.d/wview restart
```

You can monitor CWOP packet submission in the system log (and on the CWOP status pages).

Weather Underground

The Weather Underground (Wunderground) is a privately held organization that provides many weather services—some free and some not. Among the free services is the ability to register your weather station and submit your data to them, so you can access your data and some nice graphs from the Wunderground site. Weatherforyou.com also is a privately held outfit with similar capabilities to Wunderground.

Register for a Weather Underground Station ID (unless you already have one) at **www.wunderground.com/weatherstation/usersignup.asp**. Determine your accurate latitude and longitude: **www.topozone.com/viewmaps.asp**.

Click the Services tab and enable the HTTP service. Click HTTP Services and configure the Weather Underground settings. Click Save Changes.

Look in the system log for something similar to:

```
"WUNDERGROUND: configured to submit \
               station KTXCOLLI1 data to wunderground.com"
```

Confirm your data at the Wunderground server: http://www.wunderground.com/weatherstation/WXDailyHistory.asp?ID=XXXXXXX, where XXXXXXX is your Wunderground Station ID. This should start displaying your weather data graphically and as a packet list.

Data Archive and External Applications

A number of simple open-source weather station applications are available that do little more than extract the data from the weather station and archive it for later post-processing or retrieval by another server for multisite analysis. Researchers wanting to gather weather data for their own purposes is one example of such an implementation.

It is often (incorrectly) asserted that wview is "more

application" than is needed for simple archival purposes. In fact, wview allows for much configuration as to "how much" it does for you. Designed as a series of loosely coupled UNIX processes, wview easily can be configured as an archive-only server. It also is easy to add CWOP and/or Wunderground/Weatherforyou to the archive server—all without any "fancy" HTML or other file generation. If you don't want to generate a Web site, you don't have to have one!

After installation and typical configuration, disable all wview Processes (under the Services tab in wviewmgmt). The station interface process always is enabled and, thus, is not configurable. Start wview as normal. Only the wviewd_<station> dæmon will be running, collecting data from the station and archiving records and HILOW values in the archive databases.

Because wview stores archive data in SQLite3 databases, it is a simple matter to implement scripts or applications that access the data via SQL. Many wview users create their own custom Perl/PHP/Java/WordPress applications for their weather data.

Weather Site Customization

The default generation model for wview is to generate a weather Web site based on a series of HTML file templates and images. For any template file named example.[ext]x and listed in html-templates.conf, wview will generate a file named example.[ext]. Thus, myscript.phpx listed in html-templates.conf and found in $prefix/var/wview/html will have all wview tags replaced, and the resulting file will be named myscript.php. For any template file named example.htmx and listed in html-templates.conf, wview will generate a file named example.htm. The resulting files are stored at the location specified on the wviewmgmt File Generation page: "Generation Target Path".

Changing HTML templates in $prefix/etc/wview/html does not require you to restart wview. The changes you make will take effect at the next htmlgend (HTML generation dæmon) generation cycle. Changing the config files images.conf, html-templates.conf and (if supported) forecast.conf does not require restarting wview, but it does require an HUP signal to be sent to htmlgend to cause these files to be reread. Do this as follows (this also will toggle log verbosity):

```
$ sudo kill -s HUP `cat $prefix/var/wview/htmlgend.pid`
```

wview supports template macro file inclusion in template files. The meta-tag is <!--include filename.xxx-->. Any template macro file that is to be included in one or more template files should be listed *before* any templates including it in the $prefix/etc/wview/html-templates.conf configuration file. There is no restriction on the levels of inclusion, just be sure you specify macro templates early in the html-templates.conf file. The wview default Web site templates utilize several header macro files.

HTML template files (in $prefix/etc/wview/html) can be customized to your language and design preferences. The configuration file html-templates.conf specifies the template files to be used for generation. You may add or remove from this list as needed. Weather image captions can be edited in the $prefix/etc/wview/images.conf file for your language preferences. The configuration parameter on the File Generation Page "Enable Metric Units For Generation?" allows for configuration of metric units. If set to "yes", it causes wview to output all images (buckets and charts) as well as all values for HTML tags in metric units. The file images.conf can be edited to translate the English labels, titles and units to any language. By editing this file and the HTML template files, any language can be supported by wview. In fact, you easily can switch back and forth between US and metric units by toggling this configuration parameter and restarting wview.

Advanced Features

wview provides a number of features that allow advanced use of the weather data collected from your station. Alarms may be defined such that if an upper or lower bound is exceeded, a user-defined script will be executed. These scripts may send a notification e-mail or trigger an external application. It's also possible to connect to the wview server via TCP/IP socket and receive an unsolicited, periodic data feed of weather data. By using the "Virtual" station type, you can connect to another wview server remotely and receive the station data as if it were connected directly to the station hardware.■

Because wview stores archive data in SQLite3 databases, it is a simple matter to implement scripts or applications that access the data via SQL.

Mark Teel is the Software Engineering Manager for a major supplier of display and control systems for mass transit and commercial airline systems. He is also an advocate of open-source software development and has contributed to several projects, including CodeAnalyzer (a Java-based source code analyzer), radlib (Rapid Application Development Library) and wview.

Resources

wview Home Page and On-line User Manual:
www.wviewweather.com

wview Google User Group:
groups.google.com/group/wview

Citizen Weather Observer Program (CWOP):
www.wxqa.com

The Weather Underground (Wunderground):
www.wunderground.com

Fun with the iRobot Create

Let your computer reach into the physical world with the iRobot Create.

ZACH BANKS

Very little in the Linux universe interacts directly with the physical world. Although you may have peripherals that allow you to work with the computer, the computer has no way to interact with you. This is easily solvable by creating a robot for it to control. iRobot, famous for its Roombas, has created an educational robot called the iRobot Create, based on the Roomba, that is incredibly easy to work with. The Create provides a simple base to extend upon with very little effort. Some people even have mounted an old laptop to the robot to allow mobility, but that is overkill for most situations. It's not hard to create a link between a Linux box and the Create, even though it lacks official support.

The easiest way to interact with the Create is through a serial link using the cable that comes with the robot. For some computers, you may need a USB-to-serial adapter; however, they are readily available for less than $15. The connection will be a TTY serial, such as /dev/ttyS0, or if you are using a USB adapter, the connection most likely will show up as /dev/ttyUSB0.

In order to pass commands back and forth though the serial cable, the easiest tool to use is a serial port terminal. There are several versions of this type of software available. Here, I use gtkterm, a GUI terminal, but if you prefer CLI tools, both screen and minicom will work. After installing and launching gtkterm, you have to set the correct port under Configuration→Port. The port will be the device specified earlier, and if you are unsure which number to choose, you may have to try them all. The speed should be set to 57600 (baud). The other default settings (No parity, 8-bit, 1 stopbit and no flow control) are fine. I also prefer to turn on Local echo, which also is under Configuration and lets you see what you type.

To test the configuration, plug in the Create to charge and connect it to your computer. The terminal should start displaying lines such as the following every second:

```
bat:   min 0  sec 11  mV 16699  mA 566  deg-C 21
```

Unless you plan on mounting a computer to the robot itself, the serial cable will prove cumbersome as soon as the robot begins to move. To get around this, the robot needs to go wireless. Although 802.11 Wi-Fi has become ubiquitous on laptops, it is not common on embedded systems like the Create. Another candidate is Bluetooth, which also is becoming widespread; however, Bluetooth modules generally are expensive, have hit-or-miss Linux support and are very short-ranged. Recently, Maxstream's line of XBee radios have been gaining popularity in projects like this. They are very similar to Bluetooth modems and are better suited for this type of project.

All of the parts for this project can be purchased at SparkFun and are listed in Table 1. In addition to these items, you also will need some basic tools and supplies, such as a breadboard, wire and a soldering iron.

First, you need to configure your two XBee modules. To start, plug one of them in to the USB XBee explorer and connect it to your computer via USB cable (the USB XBee explorer is simply a serial-to-USB converter board that accepts an XBee module). Using gtkterm again, set it up to listen on a USB port (most likely /dev/ttyUSB0), and set the speed to 9600 baud. Type into the terminal `+++`, and the module should reply `OK`.

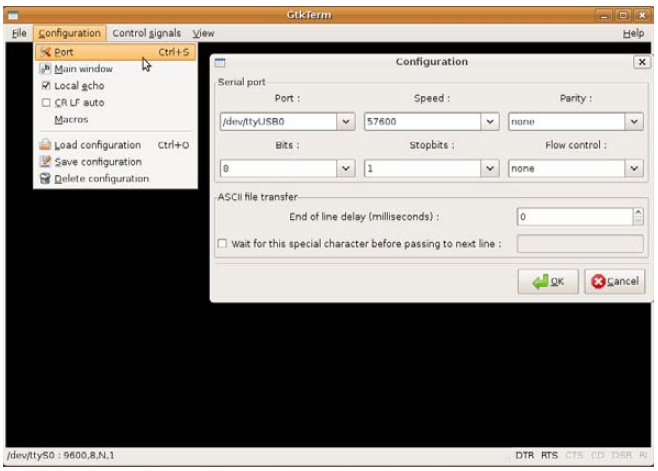


Figure 1. Configuration Options for gtkterm

The module now is ready to be configured. Type in `ATID3330,DH0,DL1,MY0,BD6,WR,CN`, and after each comma, the module will reply with `OK`. Remove this XBee, and insert the other one. Again, type `+++`, and wait for the `OK` to enter into configuration mode. This time, however, configure it with `ATID3330,DH0,DL0,MY1,BD6,WR,CN`. Each module is configured to be on network 0x3330 and to send data directly to the other at 57600 baud. One module is connected to the computer, and the other to the Create. The modules are interchangeable—either one can be connected to the computer or the Create.

Next, build the circuit to connect the XBee with the Create serially. This circuit connects the 3.3-volt XBee to the 5-volt Create. To start, solder the two sockets into the XBee breakout board. The easiest way to do this is to place the sockets on the XBee module itself, flip it over, and place the breakout board on top.

After the sockets are soldered, remove the module and solder four wires to VCC, DOUT, DIN and GND. After that, solder four more wires to the male DB-25 connector on pins 1, 2, 8 and 21. The pins should be labeled, although the markings are faint. Next, break off two six-pin lengths from the strip of male header pins, and solder them to each side of the level converter. Again, it is easiest if you use the breadboard as a jig to hold the pins straight as you solder them. Finally, assemble everything according to the schematic (Figure 2) and/or the breadboard wiring diagram (Figure 3). The completed breadboard is shown in Figures 4 and 5.

Table 1. SparkFun BOM

Part #	Description	Quantity	Price Each
WRL-08664	XBee Module	2	$24.95
WRL-08687	USB XBee Explorer	1	$24.95
BOB-08276	XBee Breakout Board	1	$2.95
PRT-08272	XBee Socket	2	$1.00
BOB-08745	Level Converter	1	$1.95
PRT-00116	Male Header Pins	1	$2.50
COM-00526	3.3V Regulator	1	$1.95
COM-08375	0.1 uF Filtering Capacitor	1	$0.25
PRT-08287	Male DB-25 Connector	1	$0.95

Plug the DB-25 connector in to the Create's expansion port, and remove the command module if present. With the other XBee plugged in to your computer, set up gtkterm to communicate with it at 57600 baud. As before, plug the Create in to charge, and with luck, you will see some output on the terminal, and the RX light on the USB explorer should blink. If not, check your connections and configuration.

Even if you did not decide to go wireless, you still can control the Create in exactly the same way. The Create, and most Roombas, implement the iRobot Open Interface protocol, or OI for short. On the computer side, let's use Python to communicate with the Create using iRobot's implementation of OI in Python. This allows you to work on a higher level and not worry about opcodes and such. You will need pySerial and openinterface.py (see Resources). There is a small bug in openinterface.py that can make it difficult to work with on Linux. The simplest way to solve this is to run this sed command in the same directory as the file:

```
$ sed -ie "803s/ - 1//" openinterface.py
```

Alternatively, you can remove - 1 manually from line 803.

The library is easy to use—for example, to drive the Create forward at full speed, do this:

```
import openinterface as oi
PORT = "/dev/ttyUSB0"            # change to your serial port
bot  = oi.CreateBot(com_port=PORT,mode="full")

bot.drive_straight(500)          # drive forward, full speed
```

In order to access sensor data, you need to request it. If you use `bot.stream_sensors()`, the Create will update the specified sensors in each argument automatically every 15 milliseconds. To stop, execute `bot.stop_streaming_sensors()`. Although you can specify manually which sensors you want to stream, it generally is easiest just to stream all of them.

Driving also is pretty simple. `bot.drive()` takes two arguments: speed and turning radius. Speed is an integer between 500 and –500, specifying the average speed of the wheels in millimeters per second, with negative values corresponding to going backward. Turning radius is a number between 200 and –200, specifying the radius of a turn in millimeters. Positive values turn left, and negative values turn right. There also are special methods that can be used for going straight and turning in place.

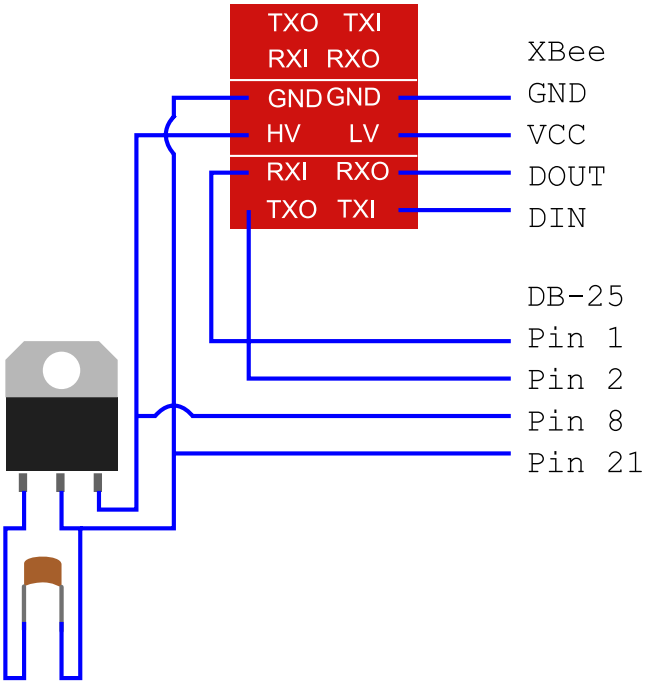


Figure 2. Schematic for the XBee/Create Interface

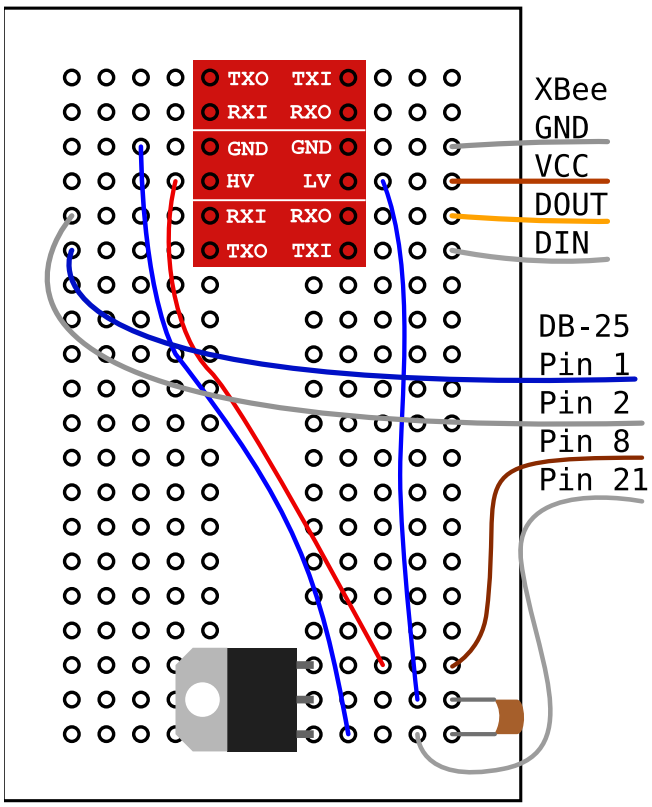


Figure 3. Breadboard Wiring Diagram for the XBee/Create Interface

The following code uses sensor data to drive and maneuver around obstacles:

```
bot.stream_sensors(6)                # packet 6 -- all sensors

while True:                          # loop forever
    if bot.sensors["bump-left?"]:    # is it pressed?
        bot.drive(-500, 10)          # spin to maneuverer
        bot.wait(5)                  # spin for 5 cycles
    elif bot.sensors["bump-right?"]: # other direction
        bot.drive(500, 10)
        bot.wait(5)
    else:
        bot.drive_straight(500)      # otherwise, go forward
        bot.wait()                   # prevents excess cycling
```

You can access the Create's song-playing abilities very easily too, and you can store songs in the 17 available song slots. Use `bot.define_song()` to store a song. The first argument is the song slot where the song will be stored, and you also use this value later to play the song back. The rest of the arguments are notes, represented by tuples of pitch and length. Length is measured in 64ths of a second. Call `bot.play_song()` to play the song. I'm no musical genius, so hopefully you can write a better tune:

```
bot.define_song(1,              # index of song
                ("G1", 16),     # note tuples
                ("G2", 16),     # note, duration
                ("G3", 64),     # 64 = 1 second
                ("G9", 16))     # up to 100 notes
# ...snip...
bot.play_song(1)
```

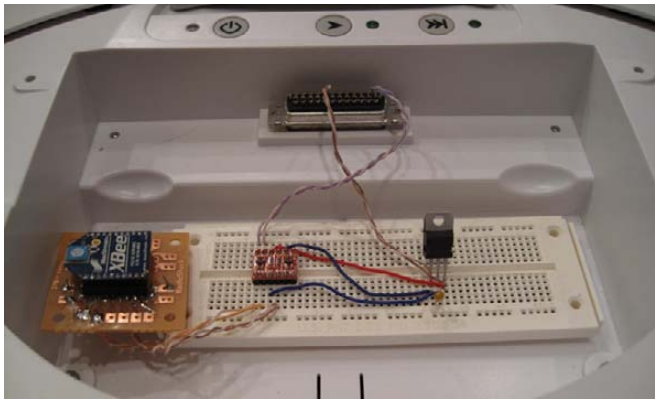


Figure 4. Overview of Electronics Components

Figure 5. Overview of the Create with All Components Installed

To control the Create wirelessly with a joystick and Python, we can use pygame (the full details of the pygame joystick API are beyond the scope of the article; check the documentation for more information):

```
import pygame
from pygame import locals
pygame.init()
js = pygame.joystick.Joystick(0)     # create joystick
js.init()

import openinterface as oi
PORT = "/dev/ttyUSB0"                # change to your serial port
bot  = oi.CreateBot(com_port=PORT,mode="full")

while True:
    if js.getAxis(0) > 0:
        turn = 1 - js.getAxis(0)
    else:
        turn = -(1 + js.getAxis(0))
    bot.drive(js.getAxis(1)*500, turn*200)
    bot.wait()
```

This code allows you to use a joystick (autodetected) to have primitive control over the Create. The x-axis value has to be manipulated so that when in a neutral position, the robot moves straight and does not spin.

Where to go from here? That's up to you. On the hardware side, you can attach additional hardware to the Create and control it through its digital inputs and outputs (see OI specifications for pin-outs). However, with just the base and some software, there still are tons of possibilities. For example, you could turn the Create into an alarm clock reminiscent of Clocky, the clock that drives around the room forcing you to get out of bed to shut it off. Or, if you are more mathematically inclined, you could use the the "distance" and "angle" sensors to map out a room.■

Zach Banks is an experimenter who is stuck between hardware and software. He's glad to accept comments and questions at zjbanks@gmail.com.

Resources

SparkFun Electronics: **www.sparkfun.com**

iRobot Open Interface Specification: **www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf**

pySerial: **pyserial.wiki.sourceforge.net/pySerial**

openinterface.py: **createforums.irobot.com/irobotcreate/attachments/irobotcreate/Create_Support/792/2/openinterface.py**

pygame: **www.pygame.org**

Full Speed Ahead with Handbrake

See how Handbrake, a powerful and easy-to-use video conversion program, can help you enjoy your favorite TV shows or films while at your computer, whether you're at home or away from home. ANTHONY DEAN

As an animation and comic-book fan, I like to watch various TV shows and movies featuring my favorite characters, ranging from the *Simpsons* and *Superman* to *Looney Tunes* and the *X-Men* films. Like most such fans, I've also bought the various DVD releases of these shows. However, since I'm not always at home, I also like watching TV programs and movies on my laptop or Palm Pre smartphone. It's great to be able to watch films while on vacation and other trips, or cartoons while commuting on the bus to work. Instead of repurchasing digital download versions of these programs, however, I find it's much cheaper (and easier, when factoring in the lack of Linux-friendliness of some on-line digital video stores) to convert my DVDs into more versatile, computer-ready video files. Doing this gives me such advantages as being able to store my video collection in an easily accessible format (without having to swap DVDs from my bookshelves), being able to convert the files for playback on various devices (such as my Pre or my old iPod), and keeping a backup of my DVD collection. Although various video conversion programs are available for Linux, the most user-friendly one I've found is Handbrake.

Handbrake, a cross-platform program (with versions available for Windows and OS X along with Linux), comes in both a command-line version and a GUI version for Linux. An open-source program, Handbrake offers the ability to convert from unencrypted DVD video sources (including VIDEO_TS folders, .VOB and .TS files, and .ISO images) and most multimedia files that FFmpeg can handle, including AVI (.avi) and MPEG (.mpg) files. Output formats include the Matroska (.mkv) and MP4 (.mp4) video containers; H.264, MPEG-4 and Theora video codecs; and AAC, MP3 and Vorbis audio codecs (along with AC3 passthrough).

Installation

Installation of Handbrake itself is straightforward. If Handbrake isn't offered in your system's package manager, the Handbrake Web site's downloads section offers you the choice of installing either a command-line version or a GUI version, for either 32-bit or 64-bit systems. For the purpose of this article, I cover the GUI version, which keeps with Handbrake's emphasis on user-friendliness. If the .deb or .rpm files offered (geared toward Ubuntu and Fedora users, respectively) aren't sufficient, another option is to download the source code and compile Handbrake.

Note that Handbrake does not unencrypt encrypted DVDs on its own, which includes most commercial DVDs available. Before installing Handbrake, make sure the libdvdcss library is installed first. The legality of converting DVD video for one's personal use may vary by locality, so check with the proper authorities before proceeding.

From DVD to Laptop

To cover the basics of Handbrake, let's use it to convert to a more laptop-friendly format DVD of the hit 2000 superhero film *X-Men* about Marvel Comics' classic mutant-powered superhero team. After inserting the DVD into the computer (and launching Handbrake), select the Source button (Figure 1).

If the DVD's directory isn't selected automatically, navigate to it; afterward, select OK, and Handbrake will read the DVD's tracks. The movie's available tracks will be displayed in the drop-down Title menu, next to the Chapters beginning and ending boxes and an Angle box. The movie itself generally will be the track with the longest time listed. Select the movie's track, then under the File destination box, type what you want to name the movie. If you don't want to convert all the chapters from the original source (say, if you just want the opening title sequence or a particular scene), type in the Chapters boxes the desired range of chapters. The Angle box can be

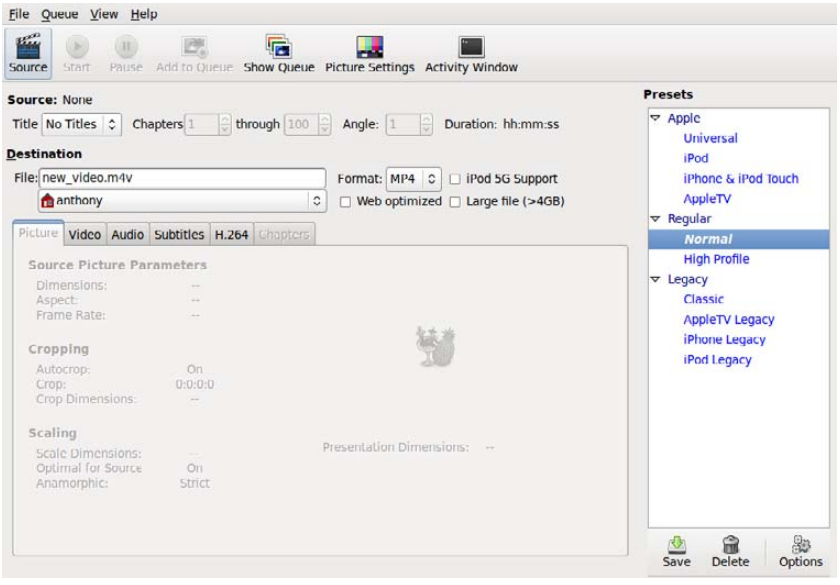


Figure 1. Handbrake's Default Interface, before Loading a DVD

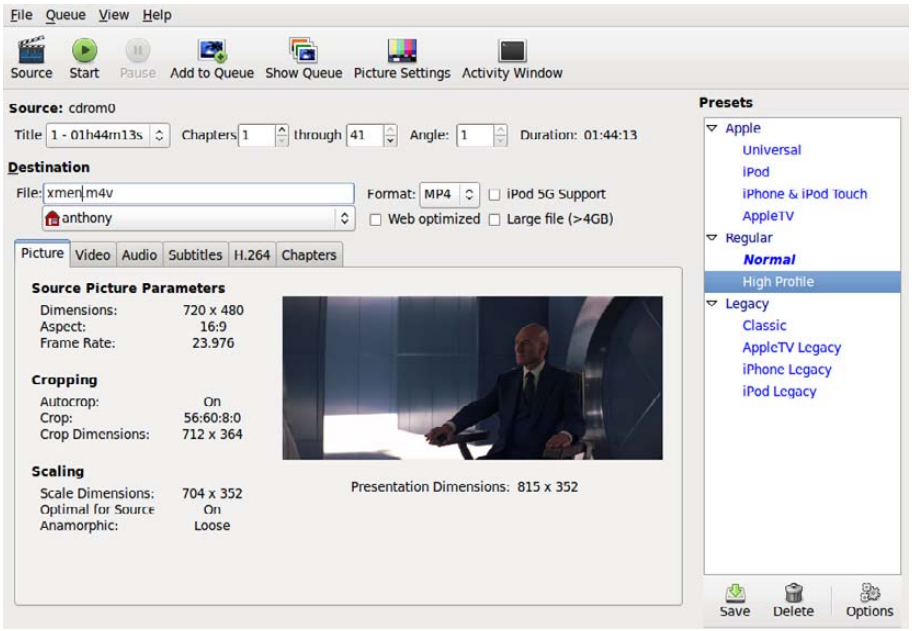


Figure 2. Handbrake after Loading a DVD

ignored by most users (including for our purposes here), given its infrequent use on DVDs.

The next step is to decide whether you want to use Handbrake's presets or your own customized settings. One of Handbrake's strengths is its various presets, which allow you to choose a convenient high-quality setting for various conversion purposes. Previous versions of Handbrake offered myriad presets, but as shown in Figure 3, version 0.9.4 (the most-recent version, at the time of this writing) has simplified the offerings for convenience and now include Normal (the default) and High Profile (high-quality) presets; presets for the iPod, the iPhone/iPod Touch and the Apple TV; and several now deprecated legacy settings, aimed at users of previous versions of Handbrake.

Figure 3. Handbrake's Presets

For my purposes (keeping the file to play on a laptop), I usually select High Profile, a high-quality setting, though I've also found the iPhone/iPod Touch preset helpful for making files specifically for watching on my Pre, which uses the same video settings as the iPod Touch and iPhone.

If you're fine with the presets settings as is, the next step is to select Start, and Handbrake will begin converting the video. However, if you want to convert multiple DVD tracks (say, the extras on a movie DVD or all the episodes from a TV show DVD), select Add to Queue first to add the current track, then select another track as previously described, and click Add to Queue. Repeat until you've chosen all the desired tracks from the DVD, then select Start.

If you're looking for finer control over the conversion settings, you need to do more than just the above, of course. As shown in Figure 1, Handbrake offers several different tabbed panes to adjust various settings. The panes include Picture (the default displayed in Figures 1 and 2), Video, Audio, Subtitles, H.264 and Chapters. Let's look at what each pane offers.

Picture

This is the default pane shown when Handbrake is launched. When the desired video or DVD to convert from is loaded into Handbrake, a preview scene from the video is displayed, along with information on the video's dimensions, including cropping and scaling. Click on the Picture Settings button in the top bar to adjust the final video's cropping and scaling settings manually, as well as the detelecine settings (which are mostly used when converting TV programs from DVD).

I generally leave the dimensions settings alone, along with keeping decombing turned on when converting TV programs (and off when converting movies, although turning it off isn't necessary).

The Picture Settings window also can play a brief preview of what the final converted film will look like, based on the current settings.

Video

From the Video pane, two drop-down menus offer settings for the video codec and framerate. Video codec choices include H.264, MPEG-4 and Theora. There's also an option to turn on two-pass encoding (a setting that scans the video source twice to ensure a higher-quality result), as well as radio buttons offering a choice of converting based on either the bitrate,

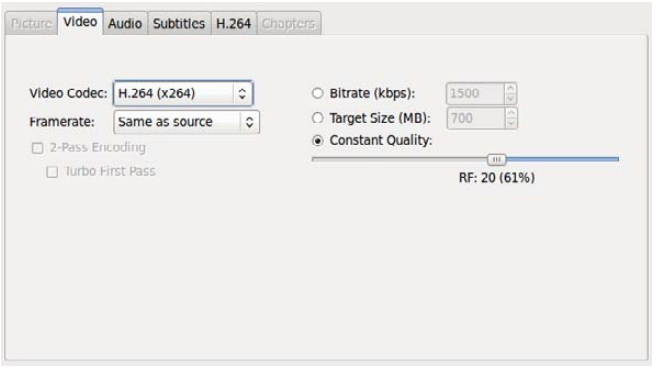


Figure 4. Video Pane

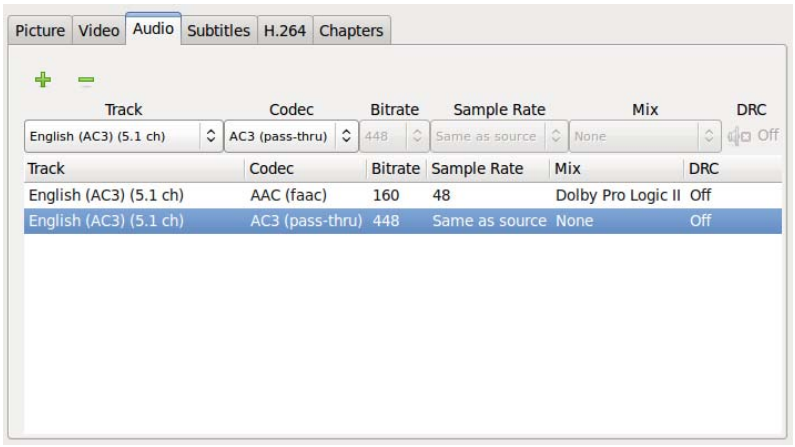


Figure 5. Audio Pane

a specific targeted file size or a constant quality rate.

I generally keep H.264 selected (Handbrake's default setting) and leave the constant quality setting as is, unless I need a specific bitrate or file size. If so, I also turn on two-pass encoding and select Turbo First Pass, a setting that speeds up the rate of the first pass. If adjusting the bitrate manually, I usually choose a figure between 1,000–1,500kbps (though I lean toward the conservative side).

Audio

In the Audio pane, depending on the video source or the preset chosen, one or several audio tracks may be displayed. The number of tracks may be added to or subtracted from using the + or - buttons shown. This is useful if you want to include, for instance, the director's commentary track from a movie or alternate languages (a potentially popular feature for, say, Japanese anime fans).

To adjust the tracks even further, drop-down menus are displayed: Track, Codec, Bitrate, Sample Rate, Mix and a button for DRC (Dynamic Range Compression). Track provides the option of choosing which audio track(s) to encode, including foreign-language tracks or director's commentary. Codec allows you to choose to which audio codec to encode (depending on which video container is chosen): AAC, MP3, Ogg Vorbis or AC3 (the default audio on a DVD). Bitrate and Sample Rates allow you to set to which bitrate and sampling rates, respectively, to encode. Mix allows you to select output mixes: mono, stereo, Dolby Surround, Dolby Pro Logic II and 6-channel discrete. Finally, DRC allows (if activated) you to adjust the limits of compressing the dynamic range of the audio output. This setting would be used for audio with very loud or very quiet portions by altering the resulting audio to increase the sound on very soft portions and reducing the sound on very loud portions.

For the purposes of encoding my *X-Men* movie, I'd choose at least one track (in English, the only language I fluently speak or read), encoded to AAC with a bitrate of 160kbps, with a mix of either stereo (my very basic audio setup at home consists of just a pair of speakers) or one of the surround sound settings. I leave DRC and Sample Rate alone. If I opt for a second track (say, to include a separate track for surround sound if I ever upgrade my home audio setup), I'd choose AC3, which would encode the audio as is from the video source. I also might decide to include a third audio track for the director's commentary (so I can hear Bryan Singer expand upon some of the film's plot points). Keep in mind that adding extra audio tracks, depending on the settings, may make the resulting file larger.

Subtitles

The Subtitles pane allows you to include subtitles and closed captions in the output file, either from the original video source or from an external file. Although I very infrequently use closed captioning (not having hearing difficulties and very rarely watching non-English-language material), this option would be highly useful for someone converting a subtitled video source.

The options in this pane include selecting one or more tracks (via the + Subtitle, + Import SRT and - buttons), then choosing a language for each track. Forced Only should be checked if you want the subtitles displayed only when the video demands it (say, when a scene has someone speaking a foreign language that must be translated and displayed on-screen for plot purposes). For this option, choose Foreign Audio Search from the track drop-down

One of Handbrake's strengths is its various presets, which allow you to choose a convenient high-quality setting for various conversion purposes.

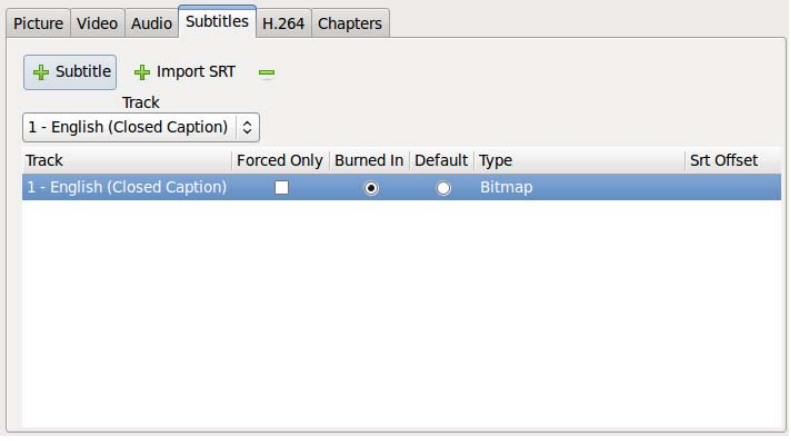


Figure 6. Subtitles Pane

menu. Burned In usually will be chosen automatically if it's a subtitle track, placing the subtitles on the movie with no way of turning them off. Closed captioning, meanwhile, doesn't use this feature, and it can be turned on or off in the resulting file's video player. There's also the option of choosing which track is the default (via the Default radio button). If you've chosen + Import SRT, then a different set of options will be presented, allowing you to choose which external file to use to import subtitles.

H.264

The H.264 pane allows Handbrake users to tweak various advanced x264 settings to their specifications. For the average user (such as myself), this pane can be left alone. Those interested in the settings here can consult the Handbrake Web site's user guide page, although as the page notes, it (at the time of this writing) hasn't been updated yet for 0.9.4.

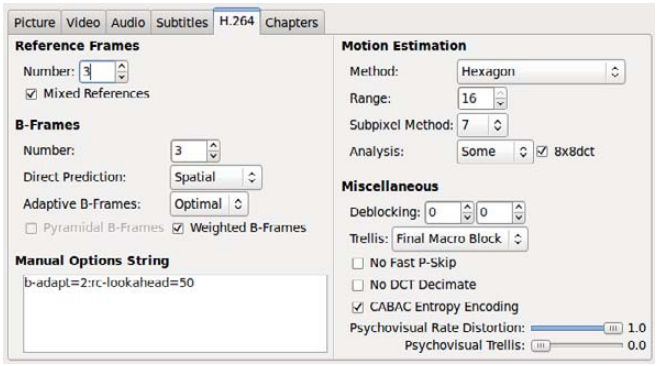


Figure 7. H.264 Pane

Chapters

The Chapters pane gives you the option of including chapter markers at the same places the chapter breaks exist on the original DVD. To activate this, select the Chapter Markers box. The length of each chapter is displayed, as well as generic titles reading "Chapter 1", "Chapter 2" and so forth. If you want to create specifically named chapter titles (such as "That cool scene where Magneto makes the guns hover" or "Statue of Liberty slugfest"), click on the chapter title of choice and type a new name.

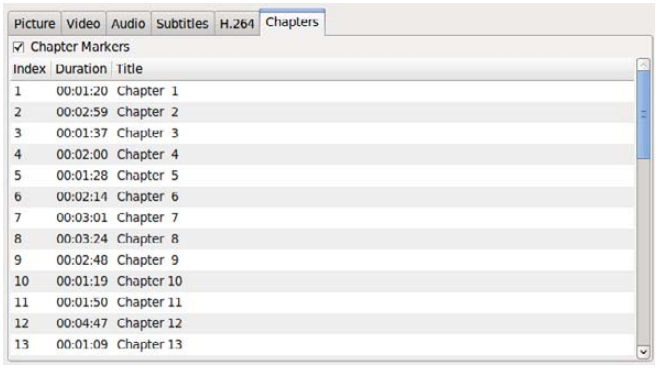


Figure 8. Chapters Pane

For my usage, I keep the Chapter Markers box checked, although I usually don't bother renaming the chapters.

Customized Presets

After performing all the previous customizing steps, you might want to save these settings for future reuse. To do this, click the Save button shown in Figure 3 at the bottom of the Presets pane, and give the desired settings a custom name in the box that appears. The new preset then will appear in the Presets pane, ready to use for future video conversions.

Length of DVD Video Conversion

After finishing all of the above, to begin conversion, click Start. For my computer, an HP laptop with 4GB of RAM and a 2.1GHz Intel Core 2 Duo T6500 processor, running Ubuntu 9.10 (64-bit), it took two hours and 43 minutes to convert the *X-Men* movie (which runs an hour and 44 minutes) using the High Profile preset. The end result of all this, of course, is that I now have a high-quality movie file that I can view (or use) as I wish.

Figure 9. Finished Video File Playing in Totem

Conclusion

As you can see, Handbrake is a great program for those wishing to convert their video media, particularly their DVDs, to a more versatile digital format. Thanks to Handbrake, I've converted much of my DVD collection for easy access and viewing on my laptop, allowing me to enjoy watching Hugh Jackman (or Bugs Bunny) in action while away from my TV set or away from my apartment entirely.■

Anthony Dean works as a file clerk for the city of Milwaukee, Wisconsin. He's been using Linux as a primary operating system since 2005. Anthony can be reached at adean33@gmail.com.

Resources

Handbrake: **handbrake.fr**

Handbrake Downloads: **handbrake.fr/downloads.php**

Handbrake's User Guide to x264 Options: **trac.handbrake.fr/wiki/x264Options**