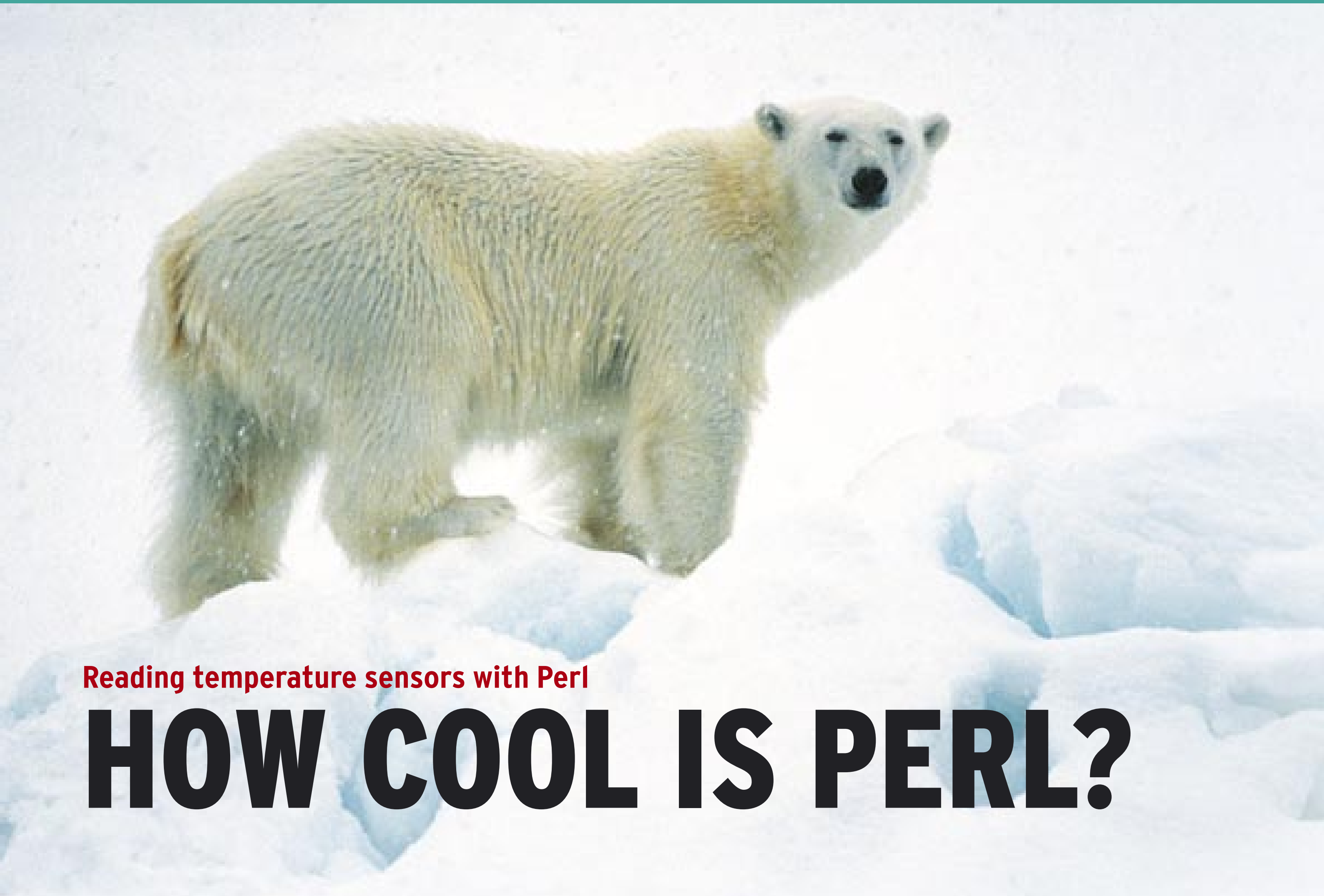


Reading temperature sensors with Perl

HOW COOL IS PERL?

Linux lets tinkerers connect their hardware creations to computers. We'll show you how you can use a simple Perl interface to check the temperature of your home or office while you're away.

BY MICHAEL SCHILLI

It's not so long ago that users had to write their own device drivers to add hardware. But with the standardization of USB, and with fully-functional hot-plugging support in kernel 2.6, things have become far easier.

The temperature sensor used in this article, a Dallas Semiconductor DS18S20 ([3]), can be controlled via the one-wire bus, which is driven by a USB dongle attached to the computer. The free *owfs* control software at [2] can request data via a Perl interface. Instead of one-wire, the bus should really be called two-wire, as two thin copper wires (typically in a single sheathing) connect the sensor to the USB dongle (see Figure 1). At the other end there is a western connector (RJ11), which plugs into the USB dongle.

The DS18S20 temperature sensor is available from most electronics stores for about US$5 (for example from digikey.com). It can handle a temperature range between -55°C and +125°C. The DS9490R one-wire USB dongle, to which multiple sensors can be connected using western plugs, costs about US$15 to US$25 (at hobby-boards.com for example).

The *owfs* project ([2]) at Sourceforge offers a number of interfaces for reading temperature values from the sensors. One uses the FUSE user filesystem and maps the sensor data on the filesystem, much like the */proc* hierarchy in Linux. Figure 2 shows what kind of data gets transferred from a dongle with two attached sensors. You can read the temperature values and also the unique sensor IDs, type designations, and other things. The sensor housing contains a microcontroller with surprising capabilities.

Measured values can be read simply by running *cat* against the FUSE-controlled files, although I used *perl -ple1* to add a newline to the output in Figure 2. Below the *10.B2A7C7000800/temperature* entry, you can see the value measured by the first sensor, 22.8125 degrees Celsius. The second sensor with the ID

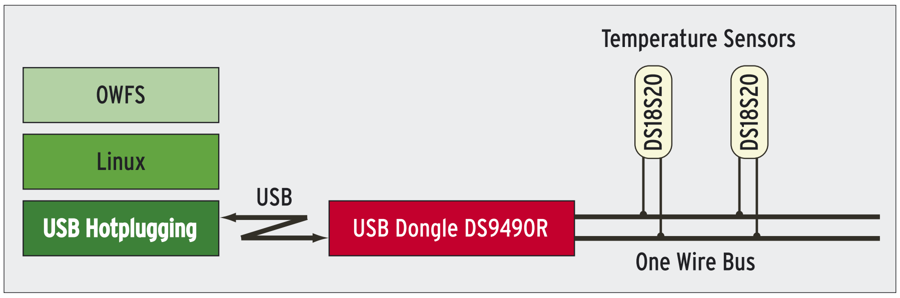
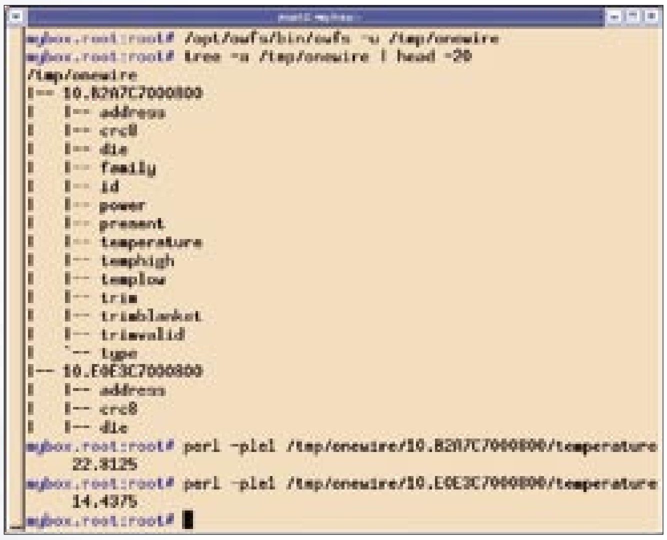


Figure 1: The temperature sensors connect to the USB dongle via the one-wire bus.



```
mybox.root:root# /opt/owfs/bin/owfs -u /tmp/onewire
mybox.root:root# tree -u /tmp/onewire | head -20
/tmp/onewire
|-- 10.B2A7C7000800
|   |-- address
|   |-- crc8
|   |-- die
|   |-- family
|   |-- id
|   |-- power
|   |-- present
|   |-- temperature
|   |-- temphigh
|   |-- templow
|   |-- trim
|   |-- trimblanket
|   |-- trimvalid
|   `-- type
|-- 10.E0E3C7000800
|   |-- address
|   |-- crc8
|   |-- die
mybox.root:root# perl -plel /tmp/onewire/10.B2A7C7000800/temperature
     22.8125
mybox.root:root# perl -plel /tmp/onewire/10.E0E3C7000800/temperature
     14.4375
mybox.root:root# 
```

Figure 2: Querying the one-wire USB dongle and attached temperature sensors at the command line.

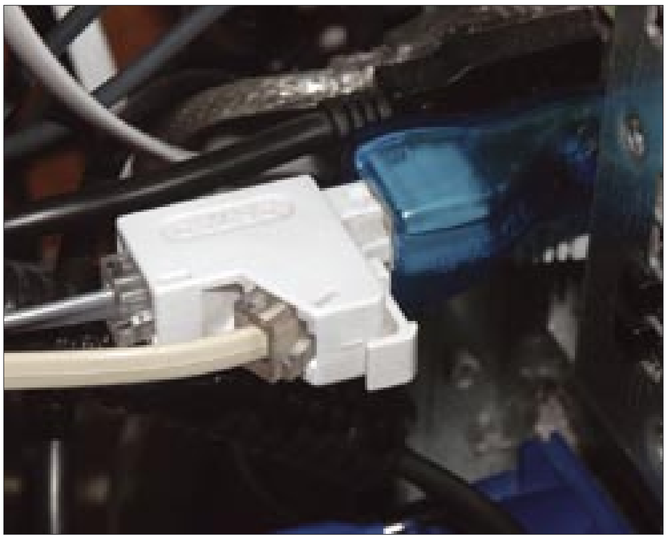


Figure 3: Two sensors connected to a one-wire USB dongle via a phone line splitter; the dongle plugs into the Linux system.

of *10.E0E3C7000800*, which I left outside overnight in San Francisco, returns a cooler value of 14.4375 degrees (it never gets really cold in California). The *type* entry gives us the sensor type designation ("*DS18S20*"); this allows you to query the type of sensor connected to the one-wire bus via the interface. The vendor, Dallas, has all kinds of equipment on sale, including switches, and voltage and current meters, which can all be attached to the same one-wire bus.

Fire Up Your Soldering Iron

The USB dongle has a western socket (RJ11). To connect the three-legged sensors to the socket, I first had to solder a long wire with a western plug onto the temperature sensor pins.

The easiest way of doing this is to buy a normal phone extension lead with plugs at both ends, and just cut off one end with wire cutters. Then strip the insulation and you should see either two

advertisement

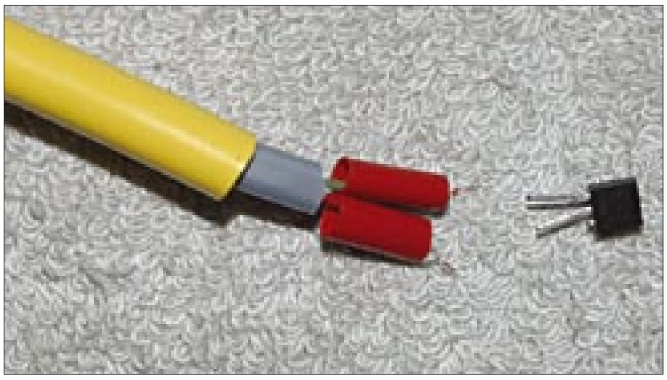


Figure 4: Before soldering: green connects to the left pin and red to the right pin on the DS18S20.

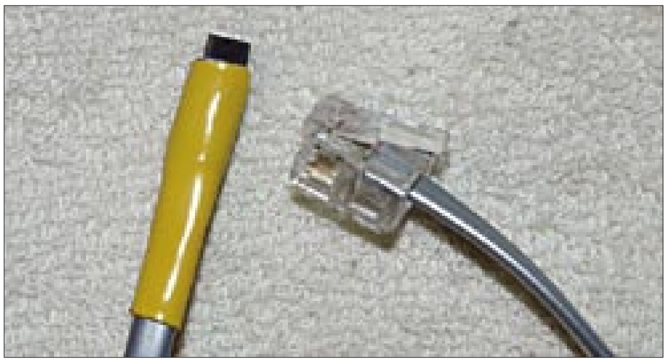


Figure 5: The finished sensor in the heat-shrunk tubing.

or four thin wires.

We will just be using the red and green wires, so we can just cut the remaining ones. The temperature sensor has three pins, one of which is not needed – this is the pin on the right (when the flat side of the sensor case is facing towards you with the pins pointing down). This pin is used to provide a power supply to the sensor, but the sensor is quite happy to use the current flowing through the data line ([8]). You can use wire cutters to remove the superfluous pin. Then go on to prepare the telephone wire by adding heat shrinkable tubing, as shown in Figure 4. We will be heating this tubing later to shrink it, making the appearance of the sensors easier to live with.

Now solder the green inner wire of the telephone lead onto the left pin of the DS18S20, and solder the red wire onto the pin in the center. Hold your soldering iron near the two red inner heat shrinkable tubes to shrink them over the stripped wire ends. If the tubing does not shrink as much as you would like, you can always add some insulating tape. This is just to prevent the bare wires touching and causing a short circuit. Then adjust the position of the thicker (yellow) heat shrinkable tubing until you can just see the tip of the sensor, and follow the same procedure to heat-shrink the tubing. Figure 5 shows you the finished sensor; you can either plug the telephone connector directly into the dongle, or you can use an extension, if you have more than one sensor (Figure 3).

Finding the Sensor

For test purposes, I will now leave one sensor in the room, while routing the other through the window outdoors. The owfs project provides a generic Perl interface in the form of the *OW* module. The module in Listing 1 customizes the module to match the type of temperature sensor we will be using.

At first, we have no way of knowing how many devices are attached to the bus, which of them are temperature sensors, and what their unique IDs are. The *discover* method, which is called by the *new* constructor, finds this out for us by opening the *type* entries for all the devices attached to the bus and checking for a DS18S20. *OW::init('u')* then tells the module to contact the USB dongle, and the following call to the *temperatures()* method returns pairs of sensor IDs and measured temperature values. The destructor in line 49 calls *OW::finish()* to close the connection to the USB dongle.

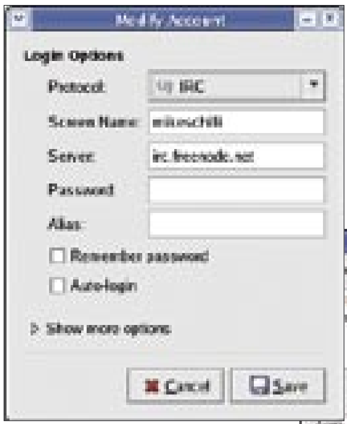
The script in Listing 2 shows a typical sensor application. It uses the *RRDTool::OO* module from CPAN to provide an object-oriented interface to Tobi Oetiker's *rrdtool*.

Without specifying any options, this script reads all the sensors and stores the current temperature values in a round-robin database. If the database does not exist, the *create* method in line 37 of the script creates the database with two data sources, "Inside" and "Outside," for the temperature inside the room and the temperature outside the window. Lines 23 and 25 map the sensor IDs to these intuitive names. The IDs are globally unique; any sensors you buy will have different IDs.

The *step* parameter in Line 38 sets the refresh interval to 300, that is, 300 seconds or 5 minutes; the database stores 5000 values before it starts to overwrite the older values. Reading and refreshing starts in Line 83 using the methods provided by *OWTemp*, and *update()* from *RRDTool::OO*.

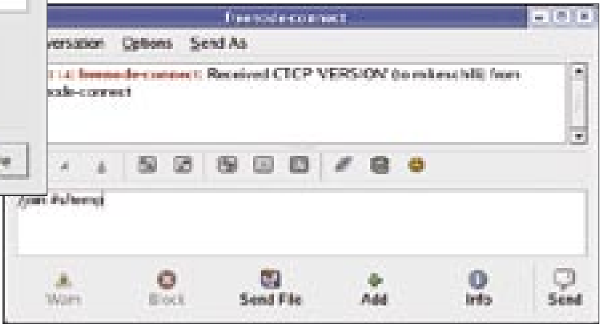
Lines 23 and 25 map the non-intuitive sensor IDs to human-readable values: "Outside" and "Inside." It is easy to find out which sensor has which ID by just attaching one sensor, and then viewing the directory structure, as shown in Figure 2.

If you call *rrdmon* with the *-g* parameter, it creates a graph of the temperature curve from the RRD data from both sensors and stores it in the */tmp/temperature.png* file (Figure 8). The inside sen-



Protocol: IRC
Screen Name: mikeschilli
Server: irc.freenode.net
Password:
Alias:
☐ Remember password
☐ Auto-login

Figure 6: The gaim IM client speaks the IRC protocol. Just create an account, click "Online," and ...



freenode-connect: Received CTCP VERSION (to mikeschilli) from freenode-connect

/join #sftemp

Figure 7: ... enter the command to enter the #sftemp channel.

sor is shown in red, and the outside sensor in blue.

Shouting Out

If you prefer remote, text-based access to the temperature output (say, you are on vacation and want to know whether you have left the oven on), you can write an IRC bot like the one shown in Listing 3. The bot connects to the IRC server at *irc.freenode.org* and opens a chatroom called *#sftemp*.

To dispel your worries, so that you can get on with your vacation, just launch an IRC client or the *gaim* IM client to visit the bot in its chatroom. Figure 6 shows the gaim configuration and Figure 7 gives you the command for entering the chatroom, where the bot is waiting for you to enter the "*temp*" keyword. The bot listens to the conversation in the chatroom, and if anyone says "*temp*", it extracts the last temperature values from the RRD archive and sends them back to the chatroom (Figure 9).

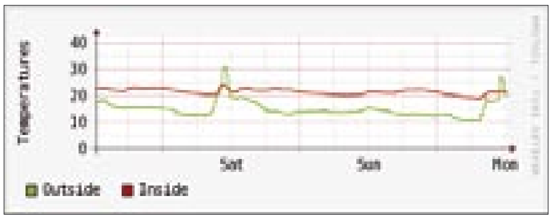


Figure 8: The graph drawn by RRDTool visualizes the temperature curve for the inside and outside sensors.

Bot::BasicBot is a good example that shows you how to complete highly complex tasks with a CPAN module and just a bit of code. You just derive a class from

Listing 1: OWTemp.pm

```perl
################################
package OWTemp;
# Mike Schilli, 2005
# (m@perlmeister.com)
################################

use Log::Log4perl qw(:easy);
use OW;

################################
sub new {
################################
  my ($class, @options) = @_;

  my $self =
    { type => "DS18S20", };

  bless $self, $class;

  OW::init('u');

  $self->{devices} =
    [ $self->discover() ];

  return $self;
}

################################
sub temperatures {
################################
  my ($self) = @_;

  my @temperatures = ();

  for my $dev (
   @{ $self->{devices} }) {

   my ($val) = owread(
    "$dev/temperature");
   $val =~ s/\s//g;
   push @temperatures,
     [ $dev, $val ];
  }

  return @temperatures;
}

################################
sub DESTROY {
################################
  OW::finish();
}

################################
sub discover {
################################
  my ($self) = @_;

  my @found = ();

  for my $entry (owread("")) {
   DEBUG
   "Found top entry '$entry'";
   next if $entry !~ /^\d/;

   my ($type) =
     owread("$entry/type");

   DEBUG "Found type $type";
   next
     if defined $type
     and $type ne
     $self->{type};
   push @found, $entry;
  }
  return @found;
}

################################
sub owread {
################################
  my ($entry) = @_;

  my @found = ();

  my $result = OW::get($entry)
    or LOGDIE
    "Failed to read $entry";

  DEBUG
    "owread result='$result'";

  for my $entry (
    split /,/, $result) {
   $entry =~ s#/$##;
   push @found, $entry;
  }

  return @found;
}

1;
```

Bot::BasicBot, and define the *said()* method for the class; the method is called when someone says something in the chatroom. *said()* receives the message as a parameter, and can then check if the bot wants to respond, and either return a message or *undef*. If you have version 0.65 of *Bot::BasicBot*, you will see a message when you launch that says *Use of - > new() is deprecated, please use spawn()*, but you can just ignore the warning.

Installation

The *owfs* software distribution, which uses a USB interface to talk on the one-wire bus, is available at [2]. When this article went to print, only the latest version from the *owfs* project's CVS repository worked; enter *cvs -d:pserver: anonymous @cvs.sourceforge.net:/cvsroot/owfs co owfs* to get the version. There is also a tarball at [5] that has been shown to work with the scripts in this issue.

You need the latest version of SWIG ([7]) to install *owfs*; the developer version 1.3.27 worked fine. If you will be installing the *owfs* command line tool to access the one-wire bus via the command line (Figure 2), along with the Perl interface, you also require the FUSE user filesystem from [4], unless your distribution happens to provide it (test by entering *ls -l /usr/local/bin/fusermount*). Then enter the following:

```
./bootstrap
./configure
make
```

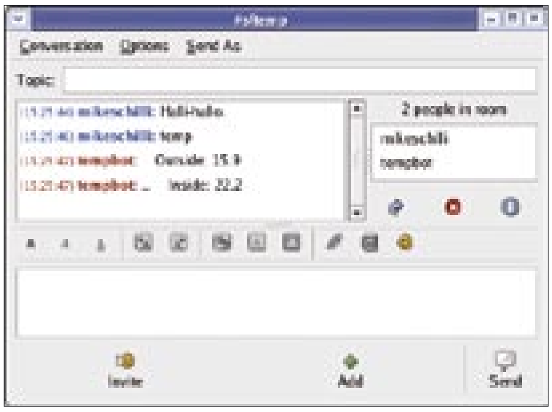
to start the build. Then run *make install* to install the command line tool. The following steps install the *OW* Perl module:

```
cd module/swig/perl5
perl Makefile.PL
make install
```

from the *owfs* distribution. A cronjob which runs every five minutes fills the

Listing 2: rrdmon

```
#!/usr/bin/perl -w
###############################
# rrdmon -Temp sensor monitor
# Mike Schilli, 2005
# (m@perlmeister.com)
###############################
use strict;
use Getopt::Std;
use Log::Log4perl qw(:easy);
use Sysadm::Install qw(:all);
use RRDTool::OO;
use OWTemp;

Log::Log4perl->easy_init(
                   $DEBUG);

my $RRDDB =
  "/tmp/temperature.rrd";
my $GRAPH =
  "/tmp/temperature.png";

my %sensors = (
 "10.E0E3C7000800" =>
   "Outside",
 "10.B2A7C7000800" =>
   "Inside",
);

getopts("g", \my %o);

# Constructor
my $rrd =
  RRDTool::OO->new(
 file => $RRDDB);

# Create a round-robin DB
$rrd->create(
 step        => 300,
 data_source => {
  name => "Outside",
  type => "GAUGE"
 },
 data_source => {
  name => "Inside",
  type => "GAUGE"
 },
 archive => { rows => 5000 }
  )
  unless -f $RRDDB;

if ($o{g}) {

 # Draw graph in a PNG image
 $rrd->graph(
  start => time() - 24 *
    3600 * 3,
  image       => $GRAPH,
  vertical_label =>
    'Temperatures',
  draw => {
   color  => '00FF00',
   type   => "line",
   dsname => 'Outside',
   legend => 'Outside',
  },
  draw => {
   type   => "line",
   color  => 'FF0000',
   dsname => 'Inside',
   legend => 'Inside',
  },
  width       => 300,
  height      => 75,
  lower_limit => 0,
 );

} else {

 my $ow = OWTemp->new();

 my %values = ();

 for my $station (
  $ow->temperatures())
 {
  my ($dev, $temp) =
    @$station;
  $values{ $sensors{$dev} } =
    $temp;
 }

 $rrd->update(
  time   => time(),
  values => \%values
 );
}
```



(15:21:40) mikeschilli: Hallo-hallo
(15:21:40) mikeschilli: temp
(15:21:47) tempbot: Outside: 15.9
(15:21:47) tempbot: Inside: 22.2

mikeschilli
tempbot

Figure 9: The IRC bot responds with the current temperatures.

RRD archive: */5 * * * * cd /path; ./rrdmon; ./rrdmon -g;.

You need both *rrdmon* and the *OWTemp.pm* Perl module in your */path*. The files created by *rrdmon* are stored in */tmp*, but you can change the path variables in *rrdmon* (lines 18/20). As each sensor has a unique ID, you need to modify lines 23 and 25 to reflect your environment. Use the approach shown in Figure 2 to discover the sensor IDs.

The remaining modules, *Sysadm::Install* and *Log::Log4perl*, are available from CPAN. RRDTool::OO requires either a working *rrdtool* installation, or it will try to download a working installation.

For the bot, you need *Bot::BasicBot*, which automatically installs the *POE* distribution.

Be Secure

When you plug the USB dongle into your computer, the hot-plug mechanism creates a USB device, something like *root -rw-r--r-- /proc/bus/usb/003/008*. As owfs needs write access to the dongle, root privileges are required to read the temperature values. An executable hot-plug script in ***/etc/hotplug/usb/ds2940*** gives you a workaround to avoid running the scripts as root:

```
#!/bin/bash
# /etc/hotplug/usb/ds2940
chmod a+rwx "${DEVICE}"
```

To allow the hotplugger to run the script when the dongle is inserted, append the following code to */etc/hotplug/usb.usermap*:

```
# /etc/hotplug/usb.usermap
# DS2940 one-wire USB device
ds2940 0x0003 0x4fa 0x2490 ➘
0x0000 0x0000 0x00 0x00 ➘
0x00 0x00 0x00 0x00➘
0x00000000
```

This allows all our scripts to run with non-privileged user IDs, which keeps the host security people from jumping on your back. ■

INFO

[1] Listings for this article: *http://www.linux-magazine.com/Magazine/Downloads/65/Perl*

[2] The one-wire file system project: *http://owfs.sourceforge.net*

[3] DS18S20 temperature sensor: *http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2815*

[4] The Fuse project website: *http://fuse.sourceforge.net*

[5] CVS snapshot for owfs: *http://perlmeister.com/devel/owfs-2.2p0RC-cvssnap.tgz*

[6] DS9490R one-wire USB dongle datasheet: *http://pdfserv.maxim-ic.com/en/ds/DS9490-DS9490R.pdf*

[7] SWIG developer version : *http://prdownloads.sourceforge.net/swig/swig-1.3.27.tar.gz*

[8] One-wire bus: *http://en.wikipedia.org/wiki/1-Wire*

Listing 3: tempbot

```
#!/usr/bin/perl -w
################################
# tempbot-Temp sensor IRC bot
# Mike Schilli, 2005
# (m@perlmeister.com)
################################
use strict;
use Bot::BasicBot;

package TempBot;
use base qw( Bot::BasicBot );
use Log::Log4perl qw(:easy);
use RRDTool::OO;

################################
sub said {
################################
  my ($self, $mesg) = @_;

  return
    unless $mesg->{body} eq
    "temp";

  my $rrd =
    RRDTool::OO->new(file =>
     "/tmp/temperature.rrd");

  my $dsnames =
    $rrd->meta_data(
      "dsnames");

  $rrd->fetch_start(
   start => time() - 5 * 60,
   end   => time()
  );

  my $string;

  while (my ($time, @values) =
   $rrd->fetch_next()) {

    for (my $i=0;
         $i < @$dsnames; $i++){
     $string .=
       sprintf "%10s: %.1f\n",
       $dsnames->[$i],
       $values[$i];
    }
    return $string;
  }
}

$^W = undef;

TempBot->new(
 server =>
   'irc.freenode.net',
 channels => ['#sftemp'],
 nick     => 'tempbot',
)->run();
```