ISSUE 08 - DEC 2012

Visit our Kickstarter
http://kck.st/TvkdvG
for printed MagPi!

The MagPi™

A Magazine for Raspberry Pi Users

Catch Santa using home automation

Win a 512MB Raspberry Pi

Merry Christmas From The MagPi

This Issue...

- Skutter
- Nanpy
- Pi Gauge
- Pibow
- CESIL Pi
- C++
- Ada
- MySQL
- Python Pit

ISSN 2051-9990

http://www.themagpi.com

Welcome to the eighth edition of The MagPi magazine,

It's Christmas! In this issue we hope to entice you into some festive projects to try after gorging yourself to the brim with Christmas pudding.

In this month's edition, we introduce you to a simple home automation project allowing you to control lights and appliances in your house using the power of the Pi! Just in time to catch Mr Claus! We get your Skutter project in motion with Morphy's article on adding wheels to your base. Gordon teaches us how to light up a Christmas tree, we have more on using the Pi to control an Arduino and Ben describes how to control servos attached to the Pi using the internet! If this isn't enough we have more of the old favourites plus an introduction to SQL.

As always, we have some great prizes for you to win in our monthly competition. The MagPi would like to say a big thank you yet again to PC Supplies Ltd who this month has outdone themselves by offering up for grabs a 512MB Raspberry Pi!

In addition to this we have some exciting news for you this month. As of December 1st, we at The MagPi are so excited to be able to offer our readers the possibility of a printed version of all eight issues of the magazine! This is something which gets constantly requested of us from our readers. All eight issues will be beautifully wrapped up in a limited edition MagPi binder making it a great gift to yourself or any of your loved ones of any age. For more information on this please visit http://www.kickstarter.com/projects/themagpi/the-magpi-magazine-from-virtual-to-reality.

On behalf of the whole team, thank you again for all your support. We hope you have a fantastic Christmas and we will see you in the New Year (1st of February).

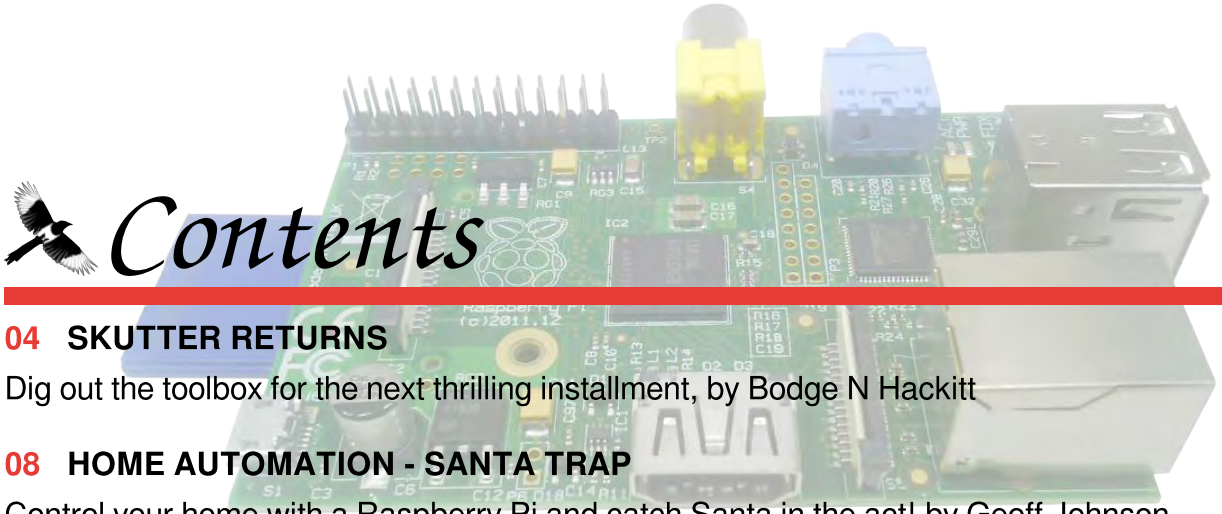
Ash Stone,
Chief Editor of The MagPi

MagPi team

Ash Stone - *Chief Editor / Administrator*
Chris 'tzj' Stagg - *Writer / Photographer / Page Designs*
Colin Deady - *Writer / Page Designs*
Jason 'Jaseman' Davies - *Website / Page Designs*
Matt '0the0judge0' - *Website / Administrator*
Tim 'Meltwater' Cox - *Writer / Page Designs / Admin*
Aaron Shaw - *Page Designs / Graphics*
Ian McAlpine - *Page Designs / Graphics*
Lix - *Page Designs / Graphics*
Sam Marshall - *Page Designs / Graphics*
W. H. Bell - *Page Designs*

Guest writers

Bodge N Hackitt - *Writer*
Geoff Johnson - *Writer*
Andrea Stagi - *Writer*
Ben Schaefer - *Writer*
Gordon Henderson - *Writer*
Alex Kerr - *Writer*
Luke Guest - *Writer*
Richard Wenner - *Writer*



Contents

Adding a motorised base Part 2

DIFFICULTY: ADVANCED

A simple, switching “H bridge”

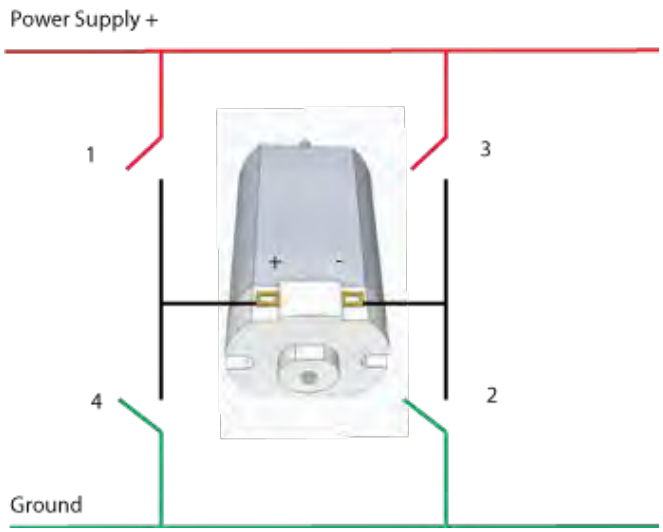
In the last article we looked at some physical means of adding motors to a robot and investigated adapting some motorised electronic toys as a potential source for robot bases.

In this article I will begin to explain how you can build your very own DC electronic motor driver module and write a basic control program for it.

We will start by re-examining the standard DC motor that was covered in the previous article.

To make the motor run forwards we apply a power source between the + and – terminals on the motor and to make it run in reverse we simply swap the power source terminals around.

The motor driver module we are going to create will need to be a circuit which is able to do this swapping around of the power supply terminals electronically. This can be accomplished using an “H bridge" circuit.



This diagram shows a simplified version of such a circuit. Closing switches 1 and 2 effectively connects the positive rail of the power supply to the + terminal on the motor and ground to the – terminal; causing the motor to run forwards. Alternatively, closing switches 3 and 4 connects the ground to the + terminal and the positive rail to the – terminal; causing the motor to run in reverse.

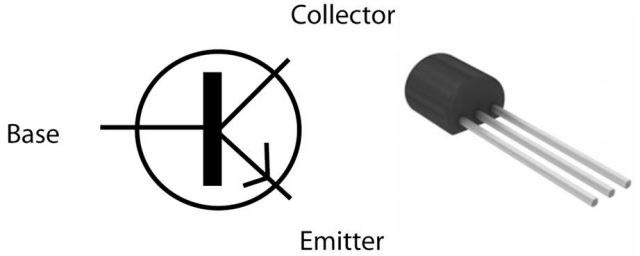
There is a potentially dangerous situation if switches 1 and 4 or 3 and 2 are closed. This would create a short circuit between Power Supply + and Ground which can be very problematic to say the least. Care must be taken when controlling this circuit to ensure that this situation can never happen.

In reality we can’t have four physical on / off switches like this as we need to control the circuit using the GPIO on the Raspberry Pi.

There are electronic solutions to this. One possibility is the use of electromagnetic relays to close these switches. However, the Raspberry Pi is not able to deliver enough power from the GPIO to directly activate such a relay without having something in-between such as a transistor. This leads us to the second possible solution which is to simply use some transistors as switches.

Transistors as switches

The transistor is arguably the most important electronic invention ever created. Its development is responsible for everything from portable music players to the processor used in the Raspberry Pi.



We will be looking at NPN type transistors. This device has three terminals called base, collector and emitter.

Connecting a power supply across the collector and emitter allows the transistor to be used as a switch. Without a connection to the base, the internal resistance of the transistor is extremely high and the switch is off.

If we apply a current to the transistor base then the internal resistance will drop by a corresponding amount and current will begin to flow from the collector to the emitter.

The transistor is able to vary its internal resistance very quickly, tens of thousands of times per second. (It's this feature that allows transistors to be used as amplifiers).

The amount that the current affects the internal resistance of the transistor is defined by a ratio known as the DC current gain and is referred to as h_{FE}.

In our case we want to supply a current to the base that will cause the internal resistance be near zero – just like a closed switch. This is called "transistor saturation" and there is an equation which tells us the current we need to apply to the base to make this happen,

$$I_B = I_C / h_{FE}$$

where I_C is the collector current and I_B is the base current. In order to find out what this current is it's necessary to measure the current that's drawn by the motor. This means an experiment is needed!

For this you will need your motorised base (in my case it's the modified Big Trak), a power supply (some batteries) and a multimeter.

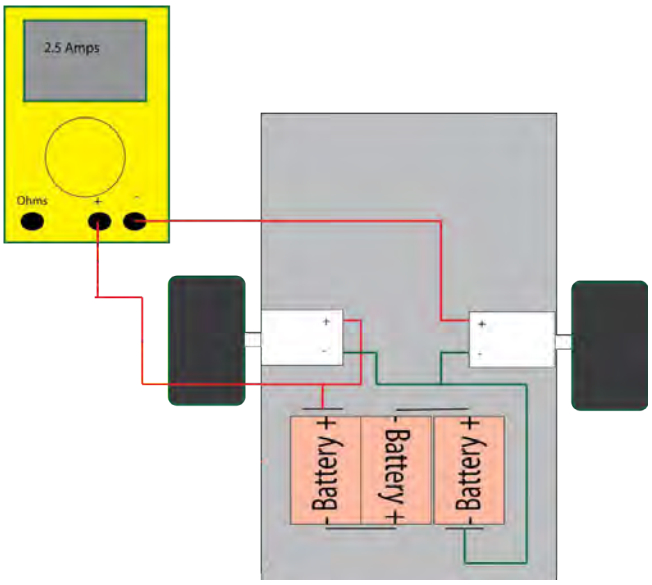
If you don't own a multimeter yet, they are an essential tool for anyone who is involved in electronics and allow you take a wide range of measurements including voltage, current, resistance, capacitance and h_{FE}.

It is possible to obtain a good multimeter for under ten pounds from a variety of retailers. Maplin sell one for £7.99 (CODE: N20AX).

DC motors draw different currents under different conditions. If a motor is free-wheeling then the motor will draw a comparatively small current.

Alternatively a stalled motor (a motor that is prevented from turning) will draw an extremely high current. The harder we make a motor work, the more current it will draw. In our case we want to measure the current the motors draw when our robotic base is trundling along the floor. One way to accurately obtain this measurement is to make the base move on the ground and measure the current that is being drawn. Here is the method I used with my Big Trak:

Connect the multimeter in series between the battery/power supply and one of the motors in the Big Trak.



The second motor must also be connected to the power supply and active otherwise only one motor will try to drive the whole Big Trak which will result in an inaccurate measurement. However, we only need to measure the current drawn by one of these two identical motors.

Add some weight to the Big Trak which approximates the expected overall weight of the finished robot. In the case of the Skutter this includes adding the robot arm.

Complete the circuit between the batteries and motor, including the multimeter in series as shown. As the Big Trak rolls along the floor, take a measurement of the current which is being drawn. Under the expected load for the Skutter using this method, one of the two Big Trak motors should draw a current of 2.5 Amps.

CAUTION: When motor stalling was tested the current drawn was approximately 20 Amps.

Continued over page...

Choosing the right transistor

Using the right transistor can make a massive difference to the design of your circuit. Often several factors need to be considered simultaneously. First, we need an idea of the minimum gain we need to provide. For this we will use the saturation equation given before. However, it is common to allow for 5 times the h_{FE} value to provide an operating margin (if you look closely at transistor data sheets this value can vary widely due to how hard it is driven). Here, I_C is the current we need for our motor at 2.5A and I_B is the current we can supply to switch the transistor (RPi GPIO pin can supply up to 15mA per pin, with a 51mA maximum across the GPIO in total).

$h_{FE(min)} > 5 \times (I_C/I_B)$
$= 5 \times (2.5/0.015) = 833$
(typical transistor gains are nearer 100…)

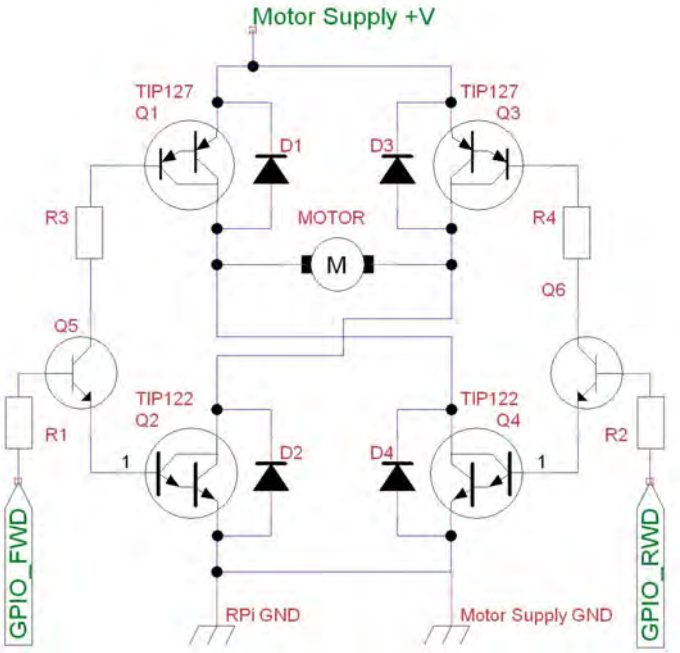
The next consideration is that the transistor will need to handle 2.5A through the collector and emitter. Therefore, $I_{CE(max)}$ needs to be high enough (and a $V_{CE(max)}$ high enough for your motor supply. Finally, the voltage drop $V_{CE(sat)}$ is also important, since not only does this determine how much of the supply voltage makes it through to the motor, but the amount of power wasted (as heat... too hot=melted) by the transistor (this is given by Power = I_C x $V_{CE(sat)}$).

The alarmingly high gain value can be solved by using more than one transistor in series, so the GPIO signal is amplified in stages. One way to do this is to use a special arrangement called a Darlington Pair; these are often available wrapped up in a single package. In fact TIP120, TIP121 and TIP122 are designed for such applications, offering h_{FE} values around 1000, $I_{C(max)}$ of 5 Amps and in a suitable form to attach a large heatsink if required.

However, another option is to make use of extra transistors to perform the switching of the H-Bridge and also provide isolation from the motor supply voltages (anything above 3.3V on the GPIO is bad news). The gain from these transistors can be used to reduce the required gain of the H-Bridge transistors; with some careful selection it may be possible to drive the motor without the Darlington Pairs (see http://goo.gl/ggHrq). This leads to the final consideration: whatever transistors you design for you'll need to source them or equivalent ones, hence why I've settled on the TIP devices. The trade-off is that the $V_{BE(sat)}$ can range from 0.7V – 4V, dropping a large chunk of the supply voltage, depending on the ratio of I_B and I_C.

Transistor motor driver circuit

Below is a "Weiss" H-Bridge circuit. You will see that our simple switch design is a little more complex than we thought, but each part of the circuit performs an important job.



There are four diodes (D1-D4) which are orientated in the opposite direction to the flow of current. This is because a DC motor can also generate an electric current when it turns faster than we are driving it. This is referred to as "fly-back current" and it can be high enough to damage the transistors. Having these reverse biased diodes allows any "fly-back" current to escape safely. If you have looked at the datasheets for the TIP devices you will notice that they already include the diodes internally, so you won't need to add extra ones (but it is important to remember if using other devices).

Also, you may note that the top transistors are slightly different to the bottom ones (the arrows point inwards). These are PNP type devices rather than NPN, which allows the driving voltage of the motor to be greater than the GPIO voltage. The PNP "twins" of the NPN devices mentioned are TIP125, TIP126 and TIP127. They function the same, except importantly they are active LOW, so the GPIO has to be turned OFF to switch the transistor ON.

You will notice that two extra transistors are used to switch on and off the TIP transistors. This allows the motor supply to be greater than the GPIO voltage without putting that voltage on the GPIO pin (with bad results for the Raspberry Pi). A nice side-effect is that not only does this make controlling the PNP

devices easy, but both the top and bottom "switches" can now be controlled with one GPIO pin for each direction.

Selection of this transistor is a little easier than the previous ones since the bulk of the work will be done by the TIP devices. A low cost general purpose transistor is suitable - the BC108 or 2N2222 should be fine.

An important aspect of this circuit is the common ground of the Raspberry Pi and the motor power supply. This allows the two separate power supplies to be tied to the same 0V allowing them to work in the same circuit (although we still have to take care to keep the motor supply +V away from the GPIO).

Resistor values

When you start to calculate the numbers, you will notice we have a lot of headroom, so we may as well aim for the ideal of only using a 5th of the gain. This means that the transistor is not only saturated, but not being pushed too hard (even if the motor voltage is raised or lowered, or more current is driven up to 5A). We will use a 9V motor supply for the calculations.

Q1, Q2 assume gain of 200 (1/5 of $h_{FE(min)}$),

$I_B = I_C/(h_{FE(min)}/5)$

$= 2.5A/200=12.5mA$

$R3 = (V_{motor}-Q1V_{CB}-Q2V_{BE})/I_B$

$= (9 - 1.4 - 1.4) / 0.0125 = 496$ ohm

(470 ohms gives 13 mA, gain 190)

Q5 assume gain of 6 (1/5 of $H_{FE(min)}$),

$I_B = 0.0125/6 = 2mA$

$R1 = (V_{GPIO}-Q5V_{BE}-Q2V_{BE})/I_B$

$= (3.3 - 0.7 - 1.4) / 0.002 = 600$ ohm

(680 ohms gives 1.76 mA, gain 7.5)

So we can use 680 ohms for R1 & R2 and 470 ohms for R3 & R4.

Control

With the extra control transistors it is easy to control the motor, turn on Q5 for forward and turn on Q6 to reverse. It should be obvious that you do not want to try forward and reverse at the same time!

You can check your program by connecting the GPIO pins to LEDs instead of the H-Bridge to confirm you will obtain the correct outputs.

Note: Pages 6/7 of this article have been updated to replace the original circuit. Other improved motor control solutions will be explored in later issues.

Article by Bodge N Hackitt

Updated by Meltwater

BasicGPIOHBridgeControl.py :

```
#! /usr/bin/python
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)

Q5=7; Q6=11 # Set GPIO Pins
#----------------------------IMPORTANT ------------------------------
#IF Q5 ON THEN Q6 must be OFF
# ELSE transistor short circuit !
#-------------------------------------------------------------------
#Set Starting State
GPIO.setup(Q5, GPIO.OUT); GPIO.setup(Q6, GPIO.OUT)
GPIO.output(Q5, False); GPIO.output(Q6, False)
print "Drive motor forwards for 3 seconds"
GPIO.output(Q5, True)
time.sleep(3)
print "Stop motor"
GPIO.output(Q5, False)
print "Drive motor in reverse for 3 seconds"
GPIO.output(Q6, True)
time.sleep(3)
print "Stop motor"
GPIO.output(Q5, False); GPIO.output(Q6, False)

GPIO.cleanup()
```

NOTE: This program would need another set of 2 GPIO pins to control a second H bridge if the robot is using two motors.

Automate your home with a Raspberry Pi... and catch Santa in the act!

This article covers an easy to build, cheap and above all, safe way to control mains powered devices with a Raspberry Pi. Nothing in this project involves going anywhere near any dangerous voltages. Soldering is limited to just a few joints and source code can be downloaded for the software part.

The Story

I bought some inexpensive remote controlled power sockets from http://www.amazon.co.uk (search Status remote control socket). *[Ed: I saw something very similar for $20 in Home Depot]*. I tried them out with a few random appliances, stuck them in the cupboard and forgot about them... but the Raspberry Pi inspired me to do something useful with them.

I decided to use the Raspberry Pi to replace the remote control and because it is programmable we can extend the capability. How about turning Christmas tree lights on and off at set times, using your smartphone to turn on the kettle just as you arrive home or flashing the bedside light when Santa walks on a pressure sensitive mat in front of the chimney on Christmas Eve!

Decoding the Remote Codes

The first step was to identify the signalling used by the sockets so I could try to mimic it using the Raspberry Pi. The communication between the remote and the sockets is radio based, allowing control without line of sight. This allows sockets in various rooms to be controlled from one location.

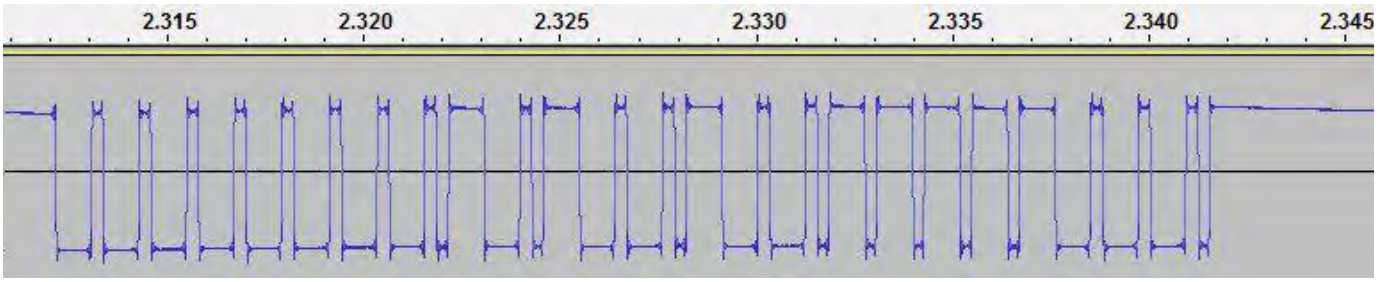
There's a sticker on the back of the sockets saying 433.92MHz, so I searched eBay for "433MHz receiver" to find a suitable receiver. This found me a transmitter and receiver for Arduino projects for only £1.60 ($1.99) including postage and packing from China!

I had to solder a 7" (17cm) antenna wire onto the receiver and transmitter. The antenna length represents a 1/4 wavelength of 433.92MHz. For this calculation you can use an online calculator such as http://www.csgnetwork.com/freqwavelengthcalc.html.

To read the code from the remote control I connected the receiver module to a computer microphone socket. I used a separate 5V power supply, but there's a 5V output on the

Raspberry Pi GPIO that would serve the purpose. The output from the receiver module is 5V digital, which isn't suitable to be connected straight into a computer audio socket. Therefore, I connected the output of the module via a 1M Ohm resistor to the microphone socket of my laptop. I assembled the circuit on breadboard, but you could use the other end of the floppy disk drive cable connector I mention in the Hardware section.

Audacity (http://audacity.sourceforge.com) is excellent freeware for examining signals like this. Once I was satisfied that the recording level was about right, I started recording and pressed one of the buttons on the remote. Having stopped the recording I was able to zoom in on the area where the signal from the remote was. The signal repeated over and over until the remote button was released. The narrow pulses seemed to be about 0.25ms in duration with the wide pulses 3 times as long.



From the waveform I created this binary string. Each bit represents 0.25ms with 1 = high pulse and 0 = low pulse. To aid readability each pulse has been separated with a dash.

```
11111-000-1-000-1-000-1-000-1-000-1-000-
1-000-1-000-1-0-111-000-1-0-111-000-1-
000-1-0-111-000-1-000-1-0-111-0-111-0-
111-0-111-0-111-000-1-000-1-000-1-0-
111111
```

This is make or break time for the project. If you have come this far and cannot find a repeating pattern, that appears when you press the remote control button, your sockets may not be using the simple AM signalling that this project relies on.

Sending a Signal

To be able to do anything with the captured data, I connected the transmitter to the Raspberry Pi. The transmitter is intended to stay connected to the Raspberry Pi, so it's powered from the +5V pin of the GPIO. I connected GPIO 7 of the Raspberry Pi to the data pin and the GND of the transmitter to the GND of the Raspberry Pi GPIO connector. The 3.3V signals are enough to drive the transmitter, though I only found this out by trying it.

Because Linux is a multi tasking OS and something else could use the CPU at just the wrong time, my program repeatedly sends the bit sequence 10 times with the expectation that at least one will be transmitted correctly.

When I ran my software on the Raspberry Pi for the first time, I captured the signal with Audacity. I could see that the waveform shape was correct, but upside down. Needless to say, the socket didn't do anything. I flipped all the bits in my output stream and re-ran the test code. This time the socket switched on! This just left me with the other buttons to transcribe.

Hardware

The connector for the GPIO and the cable I used is from an old PC floppy disk drive cable. These have a wider cable and plug than the Raspberry Pi GPIO connector, but you can fit the plug onto the GPIO pins with part of the plug going off the end. Of course, this won't work if your Raspberry Pi is in a case.

Only 3 wires are required from the GPIO; +5V, GND and GPIO 7 (CE1), the pin I use to control the transmitter.

On the other end of my ribbon cable is a very small piece of stripboard. Soldered to this board is a connector for the transmitter, made by cutting down an IC socket.

Continued over page...

Software

As the GPIO code in the example I followed from http://www.elinux.org/Rpi_Low-level_peripherals directly accessed memory it needed to be run by the root user. To make my life easy I developed and tested it while logged in as root. All commands given here assume you will be doing the same.

Everything is done from the command line so, unless your system is set to jump straight to the graphical front end, just stay at the command line.

To create a directory in which to keep the source code and the executable file while under development, type the following:

```
$ mkdir gpio
$ cd gpio
```

My source code can be downloaded from http://www.hoagieshouse.com. Just follow the links and save the file in the directory you just created. It takes 2 parameters, the channel and on or off. You'll need to edit the code to replace my remote control codes with your own. To edit the code type:

```
$ nano switch.cpp
```

Be carefull to retain the quote marks around the codes in the source. When you have changed the codes, quit by pressing <CTRL>-<X>, answer Y to the question about saving the file and accept the filename switch.cpp. To build the executable file, type:

```
$ g++ -o switch switch.cpp
```

Test it with your sockets by typing:

```
$ ./switch 1 on
```

If it works you will probably want to be able to run it as any user. Type the following commands to do this:

```
$ chmod +s switch
$ mv switch /usr/bin/
```

To schedule the sockets to come on and off at certain times, you can use something called cron jobs. Just type:

```
$ crontab –e
```

You will be able to edit a file that controls scheduled jobs. The format is described in the file, but to try something out add these lines to the bottom of the file:

```
0 *  * * * switch 1 on
10 *  * * * switch 1 off
```

This will turn socket one on for the first ten minutes of every hour.

So far, this is not very user friendly. A web based interface would be a lot nicer. My web interface allows the 4 channels to be switched on and off, but I've not gone as far as to add any scheduling to it. First, install mini-httpd to act as the web server. To do this type:

```
$ apt-get install mini-httpd
```

You can download the files for the web interface from http://www.hoagieshouse.com. In the zip file is the config file for mini-httpd and the var/www directory, which is where I put web page and cgi programs. These shouldn't need any modification, but will need to be copied to the correct locations. The HTML file uses a primitive AJAX request to run the CGI script with the channel and on/off parameters. The CGI script just pulls the parameters from the query and calls the switch program with them.

Conclusion

Now that you have the software foundation to remotely control multiple mains devices with your Raspberry Pi, I leave it as an exercise for you to imagine what other triggers you could connect to the GPIO to turn on and off devices. Oh, if you do catch Santa please tell him I would like the printed MagPi Christmas Pack from http://www.kickstarter.com/projects/themagpi/the-magpi-magazine-from-virtual-to-reality.

Maplin Parts List (Nov 2012)
VY48C - 433MHz TX/RX £9.99
M1M - 1M Ohm resistor £0.29
DG41U - Floppy disk cable £3.99

Article by Geoff Johnson

DECEMBER COMPETITION

The MagPi and PC Supplies Limited are very proud to announce a very special prize for the winner of this month's competition.

This month's prize is a new 512MB Raspberry Pi Model B plus a 1A 5V power supply and a PCSL Raspberry Pi case!

Both the 2nd and 3rd prize winners will each receive a PCSL Raspberry Pi case.

For a chance to take part in this month's competition visit:
http://www.pcslshop.com/info/magpi

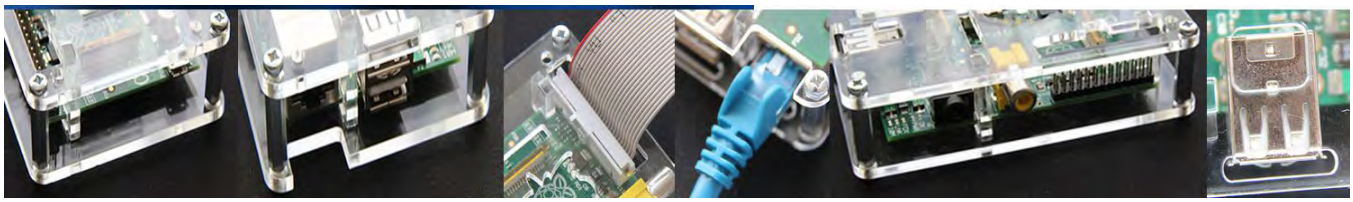
Closing date is 20th December 2012. Winners will be notified in the next issue of the magazine and by email. Good luck!

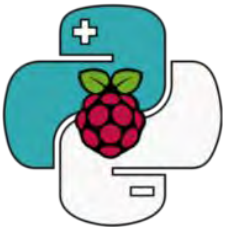
To see the large range of PCSL brand Raspberry Pi accessories visit
http://www.pcslshop.com

Last Month's Winners!

The 5 winners of the PCSL Raspberry Colour Case are **Dave Heneghan (Chorley, UK)**, **Dean Hutchison (Glasgow, UK)**, **Dave Carney (Hartlepool, UK)**, **Nigel Laudat (Liverpool, UK)** and **Peter Locastro (Derby, UK).**

Congratulations. PCSL will be emailing you soon with details of how to claim all of those fantastic goodies!





Control your Arduino board with Raspberry Pi and Python

The power of Raspberry Pi and the simplicity of Arduino using Python and a simple library: Nanpy.

Introduction

An Arduino board can communicate with the Raspberry Pi via a serial over USB connection. This creates a virtual serial interface, which it uses like a normal interface, reading and writing to the serial device file. To begin, attach your Arduino board and type:

```
$dmesg | tail
[..]usb 1-1.2: Manufacturer: Arduino[..]
[..]cdc_acm 1-1.2:1.0: ttyACM0: USB ACM
device[..]
```

My Arduino Uno board device is /dev/ttyACM0 and its driver is cdc_acm. Old arduino boards with a FTDI USB-Serial chip are accessed via /dev/ttyUSB*:

```
$ls -l /dev/ttyACM*
crw-rw---T 1 root dialout 166, 0 Nov  5
00:09 /dev/ttyACM0
```

Ok, now you should add your user to the 'dialout' group to give the required read/write access, then logout and login again for this to take effect:

```
$sudo usermod -a -G dialout YOURUSERNAME
```

This is important because Nanpy works using this device file. Nanpy is an open source project relased under the MIT license, and is composed of a server part (flashed to your Arduino which waits for commands on a serial) and a pure Python library. This library allows you to communicate with your Arduino connected via USB using classes and methods really similar to those in the Arduino framework. Behind the scenes when you create/delete an object or call methods with Python, Nanpy communicates via USB and asks the server part to create/delete the corresponding object. It also calls methods in Arduino for you: you can instantiate how many objects you want without worrying about deallocation and it's also possible to use in a multithreading context. Nanpy aims to make developers' lives easier; giving them a simple, clear and fast instrument to create prototypes and scripts interacting with Arduino, saving a lot of time. To install Nanpy read the README file. You need to install Arduino on your laptop or your Raspberry Pi in order to build the firmware:

```
$sudo apt-get install arduino
```

Nanpy is actually under heavy development and it's only been tested on the Uno board. You can get Nanpy from the Pypi page (http://pypi.python.org/pypi/nanpy) or Github (https://github.com/nanpy).
Let's see Nanpy in action and try to turn on a LED placed in the 13th pin of the Arduino:

```
from nanpy import Arduino
Arduino.pinMode(13, Arduino.OUTPUT)
Arduino.digitalWrite(13, Arduino.HIGH)
```

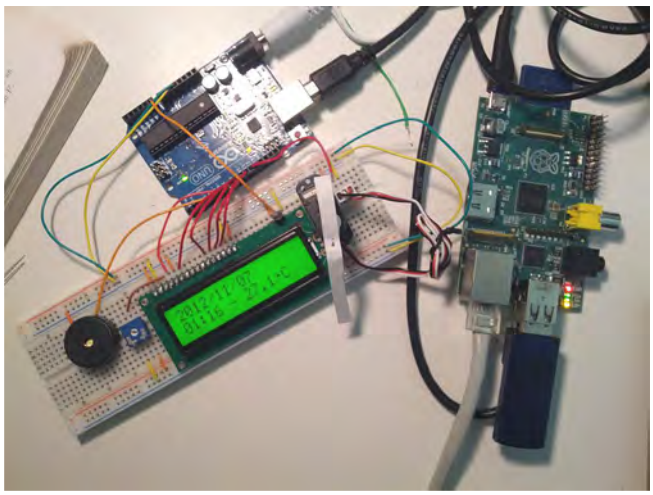
Arduino provides all of the main functions, delay, analog/digital write and read. No setup or loop functions, just objects and method calls. In fact, Nanpy supports all of the main Arduino methods - LCD, Tone, Stepper and other libraries. Now let's see how to use our 16x2 text-based LCD on pins 6, 7, 8, 9, 10 and 11, in order to write a better Hello World script:

```
from nanpy import Lcd
lcd = Lcd([7, 8, 9, 10, 11, 12],[16, 2])
lcd.printString("Hello World!")
```

Just a word of warning: Raspberry Pi may not provide enough power to drive your Arduino, so you might need to connect Arduino to an external power source.

The external world

Now I want to show you how to make Arduino communicate with the external world using the Raspberry Pi. To understand it we will build a modern clock, able to measure external temperature, with an alarm initialised via bluetooth (using an Android device in this case) and date and time updated via a NTP server...



You can find the project with instructions, an Android app and required components here: https://github.com/nanpy/eggsamples/tree/master/synclock. To show how Nanpy works in a multithreading context, this program creates a thread for every functionality, writing it all on the same LCD. In this article I show only the inner part of every "while True" cycle present in each "run" method, so I recommend you follow along with the source code. Let's start with the main thread, TimeThread, that reads the time from our ntp server every one second and stores it in a global variable, milltime:

```
response = ntplib.NTPClient().request(
      'europe.pool.ntp.org',
      version=3)
milltime = int(response.tx_time)
```

To show date and time on the LCD, create a second thread, ShowTimeThread:

```
...
self.servo = Servo(12)
...
dt = datetime.fromtimestamp(milltime)
lcd.printString(dt.strftime('%Y/%m/%d'),
      0, 0)
lcd.printString(dt.strftime('%H:%M'),
      0, 1)
self.servo.write(90 + (30 * self.c))
self.c *= -1
```

Every second we get the milltime global variable, trasform it to a readable format and then print the date and time onto the LCD. As you can see, printString can be called specifying the position (column, row) you wish the string to appear on the LCD. Then we move the servo motor like a pendulum every second. We can update the temperature in another thread. Reading the value of our temperature sensor from the analog pin 0 and printing it on the LCD, near the time, every 60 seconds:

```
temp = ((Arduino.analogRead(0) / 1024.0)
      * 5.0 - 0.5) * 100
lcd.printString("- %0.1f\xDFC" % temp,
      6, 1)
```

Ok, now let's see how to communicate with an Android phone that can set the alarm clock via bluetooth. I paired my device with the Raspberry Pi before start, follow this guide to do that: http://wiki.debian.org/BluetoothUser. Remember to install python-bluez too. We will use AlarmClock, a thread-safe class, to save on disk and get from it the alarm clock value (look at the code). Then we can start our bluetooth communication in another thread, AlarmClockThread:

```
...Bluetooth init and connection...
cli_sock, cli_info = srv_sock.accept()
cli_sock.send("%d:%d:%d", ck.getAlarm())
try:
    while True:
        data = cli_sock.recv(3)
        if len(data) == 0: break
        ck.setAlarm(ord(data[0]),
                    ord(data[1]),
                    ord(data[2]))
except IOError:
    pass
```

Our Raspberry Pi acts as a server, waiting for a bluetooth connection: once this happens, it sends the alarm clock to our device and waits for a new value to store. In the TimeThread we compare the actual time with the alarm value: if they match we can start another thread, PlayAlarmThread, playing a C note for 250ms, five times, using a Tone object through a speaker controlled via the 4th digital pin. It's time to wake up!

Start thinking about your own project with Nanpy, for example trying to bring your old RC car back to life: youtu.be/NI4PDfVMdgM

Article by Andrea Stagi

PiGauge

DIFFICULITY: Easy-Medium

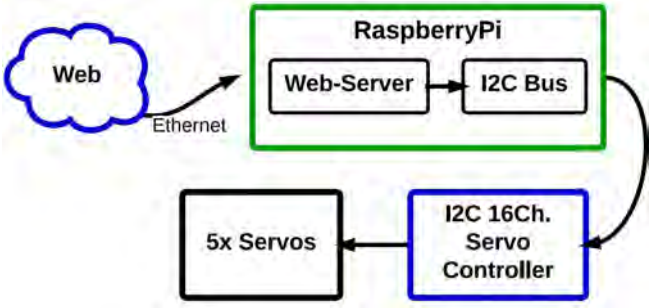
This fun project shows how to control servo motors over the internet using a Raspberry Pi.

Summary

Controlling hardware that is hooked up to the Pi is really fun. Controlling hardware from across town or another state that is hooked up to the Pi is awesome.

We are controlling five servos; each servo controls a needle on a chart that can show any data we choose through printable, modular backgrounds. We used PHP to create a webpage that is served up by the Pi. The PHP makes system calls through the command line that calls a Python script. In turn, the Python script controls the movement of the servos over an I²C bus; it also returns the positions of the servos by reading values out of a register that lives on the servo driver. The 16-channel servo driver is from Adafruit (http://www.adafruit.com/products/815); it comes with a nice library that takes care of low level operations. You need their tutorial for initial set up and library downloads. We have provided all our code and templates along with a help file in a Git repository. This project can be scaled to control up to 16 servos.

We used the newest Debian Wheezy distribution to develop the code on a Type B Rev1 Raspberry Pi. A Rev2 board can be used with some modifications to the library.



Bill of Materials

Here is a list of parts you will need to complete this project:

Bill of Materials		
Item	*Qty*	*Notes*
Servo Motor	5	180° Rotation
4-6V Power Supply	1	≈100mA per servo
16Ch. Servo Driver	1	I2C
Mounting Fixture	1	We made this in house

Adafruit servo driver datasheet:
http://www.adafruit.com/datasheets/PCA9685.pdf

Hooking Up Hardware

For safety, shutdown your Pi and remove power before making any connections.

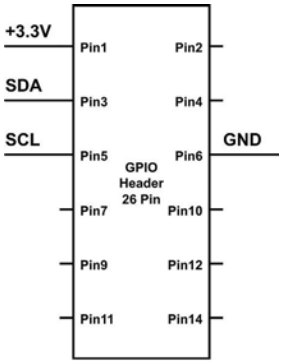
```
$ sudo shutdown -h now
```

First, connect to the servos. Most servos come with mating connectors pre-installed. Plug the connector into the servo driver, but make sure the colors match the silkscreen. We used Ch. 1-5

Black = Ground
Red = V+
Yellow = Signal

The Pi cannot source enough current to power the servos. Thus, you need an external power supply. We used a wall wart (AC adapter) from an old +5VDC cell phone charger that we had on hand. Use the terminal block on the servo driver to make the V+ and GND connections.

Lastly, connect the Pi to the servo controller. This requires four connections from the GPIO header on the Pi to the header on the servo driver: 3.3V, GND, SDA and SCL.



Double check ALL your connections BEFORE applying power.

Caution: The Vcc and V+ pins are adjacent to each other on the servo driver, don't mix them up like we did or you will have a stale Pi!

Plug in your wall wart and power up your Pi.

If you connected everything correctly you will not see or smell any magic smoke.

Download Software and Tools

Although not mandatory, it is a good idea to keep your Pi up to date; start with:

```
$ sudo apt-get update && sudo apt-get upgrade
```

Save the files in your home directory:

```
$ sudo apt-get install git
$ git clone https://github.com/Thebanjodude/PiGauge
```

Comment out all of the lines in this file:

```
$ sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

Add the I²C device to the kernel. Restart your Pi then add yourself to the I²C group:

```
$ sudo modprobe i2c-dev
$ sudo usermod -aG i2c yourusername
```

Install Apache and PHP

```
$ sudo apt-get install apache2 php5 libapache2-mod-php5
```

To find the IP of your Pi (i.e. 192.168.1.10):

```
$ ip addr
inet: ip.of.your.pi
```

Go to http://ip.of.your.pi and you should see the "It Works!" page.

Link the PiGauge Project to www root:

```
$ cd /var/www
$ sudo ln -s /home/pi/PiGauge
```

Add apache to the I²C group to allow it to access the I²C bus. Then restart apache:

```
$ sudo adduser www-data i2c
$ sudo /etc/init.d/apache2 restart
```

From your home directory:

```
$ sudo cp ./Adafruit-Raspberry-Pi-Python-Code/Adafruit_PWM_Servo_Driver/Adafruit_I2C.py /usr/local/lib/python2.7/site-packages/

$ sudo cp ./Adafruit-Raspberry-Pi-Python-Code/Adafruit_PWM_Servo_Driver/Adafruit_PWM_Servo_Driver.py /usr/local/lib/python2.7/site-packages/
```

You should be ready to go, head over to http://ip.of.your.pi/PiGauge/ and try it out!

Reading Servo Positions

In this code snippet we are adding two unsigned bytes from the I²C bus to get the position of a servo.

Continued over page...

```
def print_position(chart_num):
  chart_pos = (chip.readU8(8 +    chart_num * 4)
+(chip.readU8(9 +          chart_num * 4) << 8))
  print chart_pos
```

From Table 3 in the PCA9685 datasheet you can see that positions are stored in every 4th and 5th register starting at register 12 and 13. To get the position of the servo you'll need to add the contents of the two registers together. However, adding them as is will get the wrong answer! Why? Two 8-bit registers are "glued" together to make a 16-bit register. This is called concatenation. This means that the first register contains bits 0-7 and the second contains 8-15. To properly add them together you'll need to shift all the bits in the second register to the left by 8-bits (<<8). Then you'll be able to add them together for a 16-bit number. The cool thing about registers is that the electronics don't care what is in them. It is completely up to you, the programmer, to interpret what is inside them.

Moving Servos

The whole project revolves around moving servo motors. The following lines of code are arguably the most critical. We defined a function called move_servos(). This function takes two arguments: which chart number you want to move and where you want to move it. pwm.setPWM() comes from the Adafruit library.

```
def move_servos(chart_num, chart_pos):
  pwm.setPWM(chart_num,0,chart_pos)
  time.sleep(0.1)
```

chart_pos is a number between 170 and 608, but will vary a little from servo to servo. These numbers relate to a pulse width time (look up servo control if you are interested). To make the software more intuitive we have scaled the numbers from 0-100 using a transfer function, then we took it one step further. Since servos are not exactly linear, we took some data points, named servo_data, and coded a linear regression (a fancy word for line of best fit) to make up for the non-linearities of the servos. The linear regression function returns the variables xfer_m and xfer_b that are used below.

```
def transfer(chart_percent):
  return int(xfer_m * chart_percent + xfer_b)

def inverse_transfer(chart_pos):
  return int(round((chart_pos - xfer_b) / xfer_m))
```

Software Testing

We are big believers in the Agile software development methodology and incremental progress. We didn't deploy the use of scrums or tracking in this little project but we did make some lightweight unit tests; they are available in the repository if you find yourself curious.

Acknowledgements

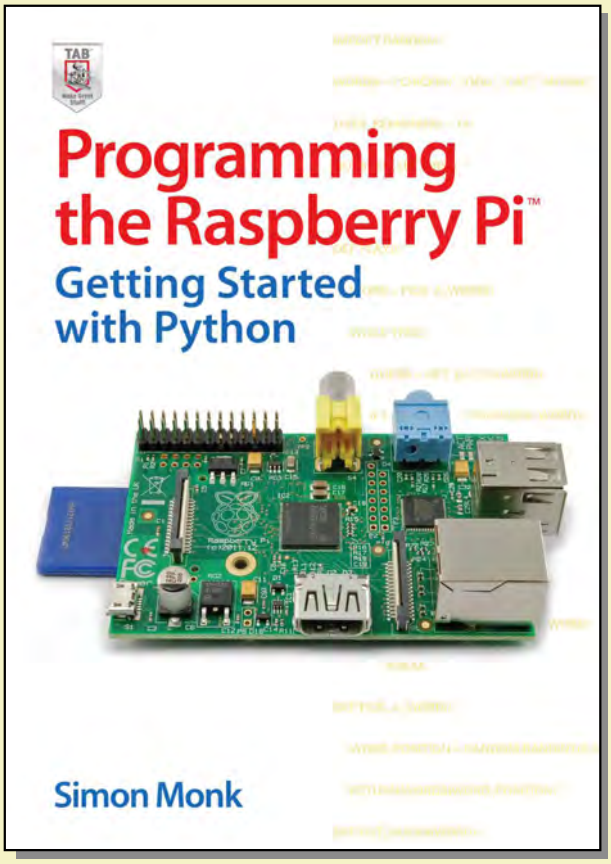
Special thanks to Scott Ehlers for patiently teaching me some new UNIX and PHP skills and to Tanda Headrick for building the mechanical display. A very special thanks to National Technical Systems (NTS) for sponsoring the project by giving us a bit of playtime to build a project status display. Follow the NTS links for more information on what we do when we aren't playing with a Raspberry Pi.

Article by Ben Schaefer

Programming the Raspberry Pi: Getting Started with Python

In a new book from www.raspberrypibook.com, Simon Monk covers basic Python to in depth GPIO usage.

Having bought a Raspberry Pi, chances are that you will be interested in learning how to program your new gadget. The book ***"Programming the Raspberry Pi: Getting Started with Python"*** **by Simon Monk**, guides the reader through the process of learning Python with the Raspberry Pi.

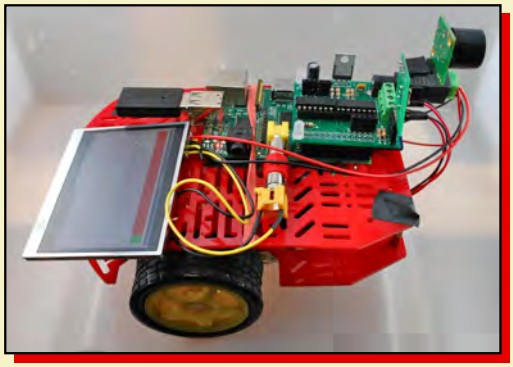


The book is accessible to newcomers to programming and leads the reader through the basics of Python, before moving on to more complex topics such as using the Tkinter and Pygame libraries as well as programming for the GPIO connector.

The approach is very much hands-on. Programming concepts are developed in example programs, which build from a simple start in the same way as you would when writing a program from scratch.

Three chapters of the book are devoted exclusively to programming and using the GPIO connector. Various techniques, tools and prototyping products are surveyed and explained including Gertboard, PiFace, Pi Cobbler and the RaspiRobotBoard.

Two of the hardware chapters are step-by-step instructions for building and programming hardware projects using the GPIO connector. The first project is a simple 7-segment LED display that shows the Raspberry Pi's system time. The second is a roving robot that uses the low cost Magician Chassis rover kit, along with the RaspiRobotBoard interface board.



All the source code from the book is available as a download from the book's website.

The book is available from most major book sellers from the end of November 2012 and further details can be found at the book's website (www.raspberrypibook.com).

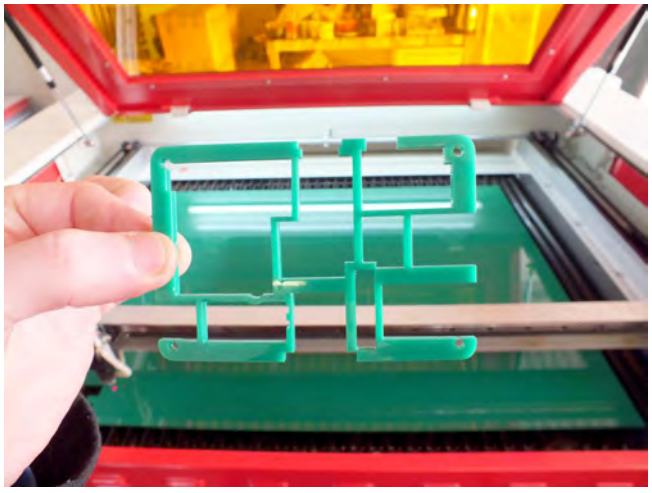
pibow interview

The Pibow guys give us an insight into the inner workings of their factory, how the Pibow has affected them, what plans they have instore for us and even tips on starting your own buisness.

Meeting the pibow guys was an interesting experience for me and as they were pretty close to me I thought it was logical to go visit them as well as doing an interview, and I'm glad I did!

Q: To what do you attribute the success of the Pibow and how has it affected you?

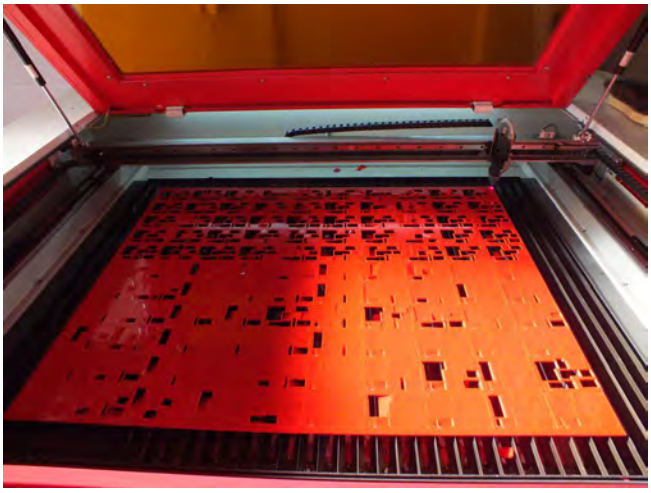
"Pibow was just so different to other cases which meant it really stood out. Obviously being featured on the RaspberryPi.org homepage was a massive coup for us and drove alot of the initial interest in Pibow. From that we just had to make sure we shipped as fast as we could. It was great sending out big batches of units; the next day there would always be loads of excited and happy tweets from people who's Pibow had just arrived."



An example laser cut part

Q: Is there anything you would have done differently?

"If we'd known how successful it would be we'd have bought the extra laser cutters sooner which would have made things alot easier and quicker. They are expensive though so we always wanted to be sure we'd need them in the longer term."



Q: What are your plans for the future; will there be any variation on the colours like a special edition 'PiWasp' (black and yellow) or 'Americana style' (red, white and blue), would there be cases with wheels?

"Right now we are focused on Picade (http://www.kickstarter.com/projects/pimoroni/picade-the-arcade-cabinet-kit-for-your-raspberry-p) which we revealed two weeks ago via Kickstarter. There are some Pibow things coming in the very near future too - including new colours!"

[O.K., so the cases with wheels suggestion was a little far-fetched but with how these guys are progressing, who knows!]

Jon and Paul from Pimoroni

Pibows waiting to be shipped

Q: Finally, what advice would you give to budding entrepreneurs from your experience?

"Know your stuff. The internet is full of quality infomation if you hunt for it. Then just go for it and accept as much help as friends and family will offer :-)

We've worked really really hard over the past four months to ship out all the Pibows people have ordered. We've run the lasers for around 16 hours a day, everyday, including all weekends during that time. We were prepared to put our lives on hold to ensure we could deliver as quickly as possible.

Friends and family have been amazing and offered help at the workshop and also provided some financial support to get us up and running. We can't thank them all enough for what they have done to help."

They look like some great cases. Hopefully, we will see the Picade kit in mass production soon too.

PICADE

Article by Chris Stagg

DID YOU KNOW?

Paul from Pimoroni is responsible for designing the Raspberry Pi logo!

The Pibow case is made from acrylic.

Pimoroni now have 3 laser cutters called Bert, Ernie and Cookie Monster! *[Ed: It can't be long before Oscar, Big Bird and The Count appear.]*

Learn how to create a 70s Christmas tree with CESIL and a Raspberry Pi.

CESIL - Standing for Computer Education in Schools Instructional Language was designed in the 1970s as an attempt to introduce young people in schools into the world of computer programming. Without computers in schools, pupils wrote programs on paper and sent them into their local computer centre. The results would come back in the post a week later!

CESIL is a very simplified assembly language with a very limited application base, however it is easy to learn and write simple programs. On its own CESIL is not terribly exciting, so I've written an interpreter for it in BASIC and added on a Christmas tree with programmable fairy lights! The tree has 4 rows of 8 lamps. Think of it as a grid 8 wide and 4 high.

A CESIL program is essentially three columns of text. The first column (which can be blank) is the label - it's a placeholder in the program which you can "jump" to from other parts of the program. The middle column is the operator - that's the instruction to execute. The final column is the operand - this is data for the instruction to use. This data may be the name of a label (if it's a jump instruction), it may be a number or it may refer to a named memory store or variable.

My extensions to the CESIL machine have included two more registers (three in total) to hold the row and column locations of the lamps and a colour instruction to set the lamp colour as well as a subroutine facility. The program can be up to 256 lines and contain up to 256 variables.

The best way to explain it may be to look at an actual program. This program reads in a number from the keyboard and prints a multiplication table:

```
# mtable:
# Multiplication table generator
  line
  print "Multiplication table generator"
  line
  print "What table"
  in
  store table
  load 1
  store index # Index times ....

loop:  load index
  out
  print " TIMES "
  load table
  out
  print  " = "
  mul index # Table was in accumulator
  out
  line
  load index # Add 1 to the index
  add  1
  store index
  sub  11 # Subtract 11 counting 1 to 10
  jineg  loop # If <0 then jump to loop
  halt
```

Blank lines are allowed and comments start with the # symbol. Most of this should be self-explanatory, but with just one accumulator, everything has to be transferred to and from memory - the "store table" instruction stores the accumulator in a variable called "table".

The standard CESIL instructions are:

`LOAD` - Transfer the number into the accumulator

`STORE` - Transfer the accumulator into a named variable

`JUMP` - Jump to the given label

JINEG - Jump if the accumulator is negative
JIZERO - Jump if the accumulator is zero
ADD - Add a value to the accumulator
SUB - Subtract from the accumulator
MUL - Multiply the accumulator with the value
DIV - Divide the accumulator with the value
HALT - End program
IN - Read a number from the keyboard
OUT - Outputs the accumulator as a number
PRINT - Prints a literal string (in "quotes")
LINE - Prints a new line

Extensions:

JSR - Jump to subroutine
RET - Return from subroutine (to the line after the last JSR instruction)

Christmas Tree extensions:

TREE - Build a new Christmas Tree
ROW - Transfer the accumulator into the Row register
COL - Transfer the accumulator into the Column register
COLOUR - Set the lamp indicated by the Row and column registers to colour value in the accumulator
WAIT - Delays for the given number of centi-seconds (100ths)

Note that you need to execute a WAIT instruction to actually reflect the colour changes in the lights. That means that you can set a lot of lights at once, then when you execute a WAIT (even a WAIT 0) instruction, all the lights will change at the same time.

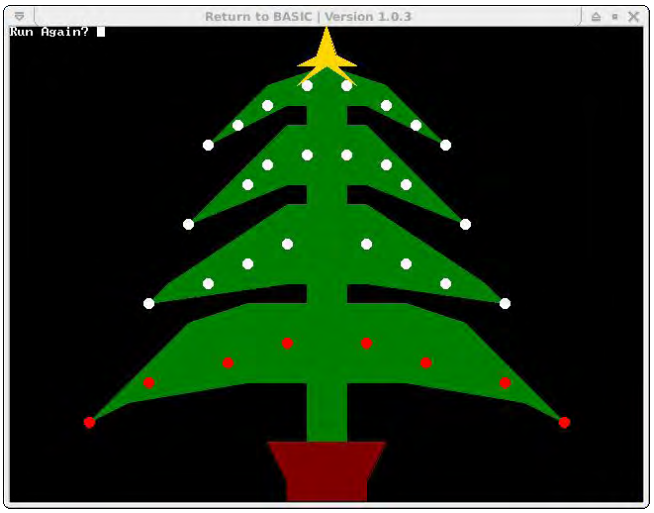
There are 16 standard colours:

0: Off, 1: Navy, 2: Green, 3: Teal,
4: Maroon, 5: Purple, 6: Olive, 7: Silver,
8: Grey, 9: Blue, 10: Lime, 11: Aqua,
12: Red, 13: Pink, 14: Yellow, 15: White

Our Christmas tree has 4 rows of 8 lamps; row 0 is at the bottom and column 0 is the left hand side.

The following program fragment will fill the bottom row with red lamps:

```
# Example program to light the bottom
#     row with RED lights
  tree  # Make a tree!
  load  0
  row  # Row 0 - Bottom
  load  7 # Count 7 to zero
loop:
  store  col-count
  col
  load  12  # Red
  colour
  load  col-count
  sub 1
  jineg  done  # Jump If Negative
  jump  loop
done:  wait 1     # Update the lights
  halt
```



First download the RTB BASIC interpreter from https://projects.drogon.net/return-to-basic/

Then you can install the CESIL interpreter and demos using:

```
  cd
  git clone git://git.drogon.net/cesil
  cd cesil
  rtb
  load cesil
  run
```

What I would like to see is people sharing examples, so please post them on forums, email them to me (projects@drogon.net) and what I'll do in January is have a look at the ones I've found and send a free Raspberry Ladder board to the one I think is the most original or clever...

Article by Gordon Henderson

Last time we looked at the very beginning of C++ and writing our first few programs. Today we'll carry on, as well as showing you how to run your programs and some more variable types.

Compiling and Running Programs

Once you've written your program (you may use any text editor, such as nano or Geany), save it as a .cpp file, then open a new terminal window and type the following:

```
g++ [name].cpp -o [program_name]
```

Replace `[name]` with the name of the file and `[program_name]` with what you'd like the actual program to be called.

If you get an error saying no such file or directory, you will have to use the `cd` command to find the directory you saved the .cpp file to, and then run the command again. If you get an error saying g++ is not found (this shouldn't be the case on newer images), type `sudo apt-get install build-essential` to install g++.

When compilation has finished (it might take a while for bigger programs), simply type:

```
./[program_name]
```

With `[program-name]` being the same as the one you used to compile. The program should then run, and when it is done you will be returned to the terminal as with any other command line program.

More Variable Types

Last time we looked at the `int` variable type, which lets us store whole numbers. Of course, we may need to store more than this, so there are different basic variable types. They are as follows:

Name	*What it stores*	*Example*
int	An integer – a whole number.	42
float	A decimal number correct to 6 decimal places.	3.141592
double	A decimal number correct to 10 decimal places.	3.1415926535
char	A single character.	c
string	A string of text.	Hello, how are you?
bool	True or false.	True

To make a variable, simply type the kind you want followed by the name you want to give it,

followed by a semicolon. Have a look at the code below:

```
#include <string>
#include <iostream>
using namespace std;

int main()
{
  // Make some number variables:
  int wholeNumber = 5;
  float decimalNumber = 5.5;

  /* Make some letter variables:
  ** Notice how we use single quotes
  ** for characters, and double quotes
  ** for strings. */
  string greeting = "Hello there";
  char punctuation = '!';
  // Make a boolean variable:
  bool isTrue = false;

  // Print our variables added together:
  cout << wholeNumber + decimalNumber << endl;
  cout << greeting + punctuation << endl;
  cout << isTrue << endl;

  return 0;
}
```

Notice what happens when we add the different kinds of variables. The number ones work as expected; 5 + 5.5 equals 10.5 and that's what we get out. When we add letter ones, the letters get added together. So we have our string, “Hello there”, and we add the exclamation mark, so we end up with “Hello there!”. This is why data types are important. If we had saved the numbers as strings, it would have joined 5.5 to the end of 5, and we'd end up with “55.5”. It is also a good idea to give variables sensible names. Variable names can have letters, numbers and underscores, but cannot begin with a number. There are also certain keywords you cannot use as they are reserved for the language.

Also notice we get 0 instead of “false” for `isTrue`. A boolean is basically a 0 for false or a 1 for true, just like binary, so that's how the program outputs it. Last time we used `cin` to let the user input a value and store that as a variable. Try doing that with the code above. We can also use the = sign to change what's stored inside a variable at any time (that's why they're called variables). The value of one variable can be assigned to another variable, for example:

```
int i = 4;
int j = i;
```

However this cannot be used to assign a value to a string:

```
int i = 4;
string s = i;
```

Since the string class has no member function to allow this assignment. The assignment of a number to a string can be achieved with a stringstream, which will be discussed in later tutorials.

Baby steps in a...

BIG WORLD

Ada, a language for everyone

By Luke A. Guest

Introduction

Following on from issue 6, we will continue to cover the basics of the Ada language.

Numeric types (continued)

Integer types can have negative numbers, such as -10, -55, etc. Natural types can only accept values starting from 0 (so no negative numbers) and positive types can only accept values starting from 1 (no negative values and no zero either).

All of these types can be used together in mathematical expressions. However, you must make sure you do not go outside of the range of the type you assign to, otherwise an error will occur.

For example, if you define two variables X : Natural := 1; and Y : Integer := 2; and then subtract Y from X and then assign back into X again (e.g. X := X – Y), this will cause an error as the result is -1 which is outside the range allowed for Natural types.

Boolean types

Boolean types have two values; either True or False. There is nothing else you can assign to a variable of Boolean type.

Simple decisions

All languages have the idea of boolean values as all conditions in programming rely on the idea of something being true or false.

```
1   with Ada.Text_IO;
2   use Ada.Text_IO;
3
4   procedure Decisions is
5      Is_Defective : Boolean := False;
6   begin
7      if Is_Defective = True then
8         Put_Line ("Defective");
9      else
10        Put_Line ("Not defective");
11     end if;
12  end Decisions;
```

Listing 1: decisions.adb

Line 5: We define a Boolean variable, Is_Defective, and set it to be False.

Line 7: We test to see if Is_Defective = True. The = means whatever is on the left equals or is the same as whatever is on the right. The part between the if and the then is called an expression. We could also just have put **if Is_Defective then** to mean the same thing.

Also, **if Is_Defective = False then** and **if not Is_Defective then** mean the same thing.

Line 8: Anything between the **then** and the **else** keywords is run when the condition is true.

Line 10: Anything between the **else** and the **end if** keywords is run when the condition is false.

Line 11: All if statements should end with end if; even if there is no else part.

Type in the code in Listing 1 to see how we can make decisions in Ada using a Boolean type.

Literals

We have already seen some literals, even though you don't know what they are. A literal is any piece of data in source code such as the number 125 or the string “Hello, from Ada” in Listing 1 in Part 1; these are numeric and string literals, respectively.

Also in Ada there are character literals, such as 'C'. A string is made up from characters so we can add characters to strings using the & operator, like we did in Listing 2 in Part 1. For example “Hello “ & 'G' is the same as “Hello G”. You can create variables of type Character just as we did previously with Integer, Natural and Positive.

For the Boolean type, as noted above, the values of True and False are boolean literals.

More attributes

We have already come across the 'Image attribute, but now we will talk about three more attributes of integer types that are very useful. Every integer type has a range of values it can accept, this range is defined by the 'First and 'Last attributes. We can also get access to this range using the 'Range attribute - this really returns 'First .. 'Last, where the two dots (diaresis) say “this is a range of values”.

If you try to assign a value outside this range to a variable of a type, it will cause an error and your program will not run correctly.

Another use for 'First is to assign an initial value to a variable. Looking at Listing 2, the initial value to assign to a variable comes after the := symbol after the type name.

Cool features: Attributes

Many language entities provide attributes that can provide information to the programmer. You access attributes using the apostrophe ('). In our examples we have seen the 'Image (), 'First, 'Last and 'Range attributes of the 3 Integer types.

Exercises

1. Use the Integer'Image attribute to print the values returned by Integer'First and Integer'Last attributes.
2. Repeat exercise 1 but for Positive and Natural types.
3. Try replacing the initial values with the 'First attribute.

A bit of history

Back in the 1970s, the American Department of Defence (DoD) decided they wanted a programming language that could be used for all their projects. A number of teams around the world submitted proposals for a new language, each named after a colour: Red, Green, Blue and Yellow. The Green team, led by Jean Ichbiah from CII Honeywell Bull in France, won.

The first version of the Green language was passed onto programming teams within the DoD, who were told to use it for all subsequent projects.

The Green language was renamed Ada after Lord Byron's daughter, Augusta Ada (Countess of Lovelace), who worked alongside Charles Babbage and who is considered to be the world's first computer programmer due to her work on his Analytical Engine.

The first version of the Ada language came about in 1983, named Ada 83. The next version came along in the 1990s, initially called Ada9X and later renamed Ada95. The next version was Ada 2005 and another revision was released this year called Ada 2012.

The GNAT compiler is still being worked on to support the newer Ada 2012 features so unfortunately some of these won't work if you try them out. We can definitely use Ada95 and also most of the Ada 2005 features for our programs.

DATABASE BOOTCAMP

There seems to be a good deal of debate about the language most suitable for learning on computers and one rarely offered is SQL – 'Structured Query Language.'

It is the international standard used to control databases and I contend easier to learn than most code. It is what is known as a 'high level language' i.e. very close to a clipped form of English. SQL covers a huge range of applications worldwide - not least that of manipulating intelligent websites!

Once you start to play you will understand the rhythm, or the syntax, in speaking SQL to the point where you could almost guess a command.

Follow this sequence on your own Pi and I explain how to get MySQL working and produce a solid platform on which you can develop your database skills – beyond the scope of this article.

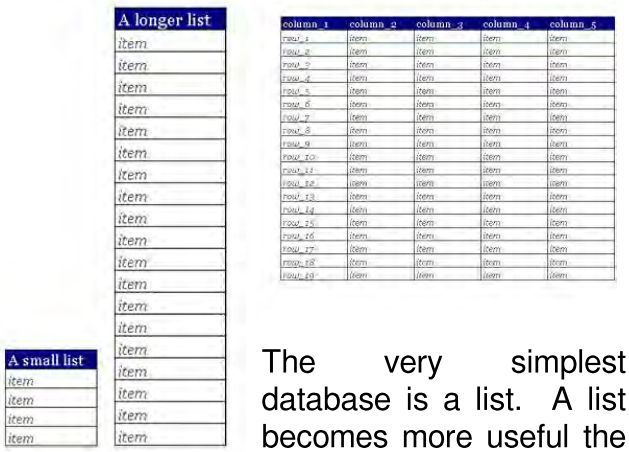
Installing MySQL

We are going to use an Open Source version of SQL called 'MySQL' but this is not included in the standard Raspbian distribution so needs to be located, downloaded and installed onto your Pi. Luckily, this complex process is controlled by one command and is run in a text screen or terminal window:-

```
pr0mpt > apt-get install mysql-server
```

Your Pi trundles off and begins the download. There may be periods in this process where nothing appears to happen but be patient. Eventually you will be presented with a blue screen and asked to set a password for MySQL – twice for confirmation.

Databases



The very simplest database is a list. A list becomes more useful the longer it gets but probably then requires more columns so you end up with a table.

A table has columns and rows – in database speak these are called fields and records. A large database can contain many tables – and so it goes.

There are three basic functions in maintaining any data in a table; add, delete and modify.

So in terms of the data we keep we can **add** a new record, **delete** an old record or **modify** the contents of an existing record – that is all! (DML - Data Manipulation Language).

It terms of how we keep the data we can **add**, **delete** and **modify** tables and **add**, **delete** and **modify** fields – that is all! Master these commands and we are well on our way to controlling databases. (DDL - Data Definition Language).

Once the data is in place it is useless unless we DO something with it and this too can be reduced to two basic functions **sort** and **search**.

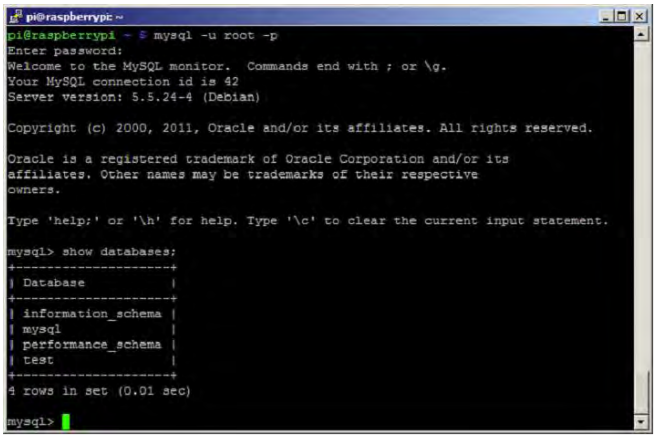
Data

We have managed to reduce elementary databases down into a few functions but before we proceed into MySQL we need to take one final look at data itself.

If ever you have needed to complete a form, paper based or on a web site, you would have come across one where the requirements are unclear or your response would not fit into the space available. You can scribble notes on a paper form or reduce the size of your handwriting to fit but that's not possible on computers as they are more highly defined. So it is with 'data-types'. MySQL recognizes only three types of data text, number and date. To store the data efficiently it needs to allocate the correct amount of computer storage space too.

Entering MySQL

Enter MySQL using the command `mysql -u root -p` From here on you have left Raspbian and now need to type SQL Structured Query Language. Although SQL is an international standard, local variations exist and MySQL is a particularly polite version. You have to say please at the end of every command otherwise nothing happens.



The short cut for please is the semi-colon. **Remember - no ; no action**.

Viewing the databases

To show the existing databases simply type:-

```
pr0mpt > show databases;
```

Should you forget your manners enter the please **;** on a separate line. You will display the four default databases. Before we start changing things lets become confident in our ability to look around, e.g:-

```
pr0mpt > show tables in
information_schema;
```

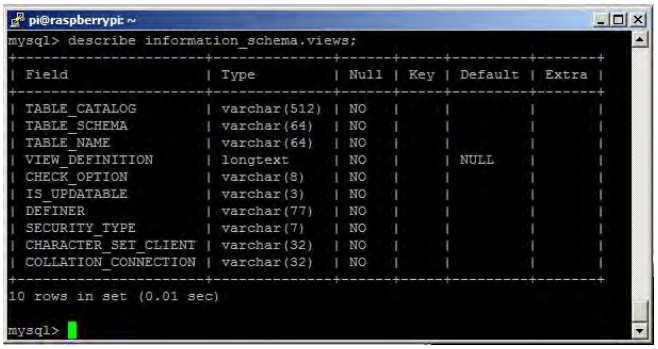
This displays the list of tables in the database information_schema. The columns in a table can be seen in three ways. All produce the same result:-

```
pr0mpt > show columns in
information_schema.views;
pr0mpt > show fields in
information_schema.views;
pr0mpt > describe
information_schema.views;
```

Note: The general format of the command describes the table in the database separated by a full stop i.e:-

```
pr0mpt> show fields in
<databasename>.<tablename>
```

Work through and try these examples and test your own options to gain the most from this article. A typical output is:-



Here you can see the columns – called fields – and other details that will be explained shortly.

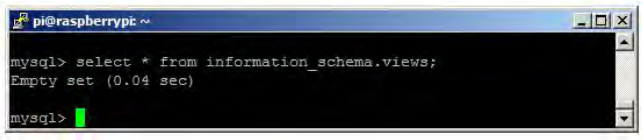
Viewing the data

We can go further by selecting to see all the data in any of the tables. The star * is known as a 'wild card' and means 'everything' so to select everything in a table enter:-

```
pr0mpt > select * from
information_schema.views;
```

Continued over page...

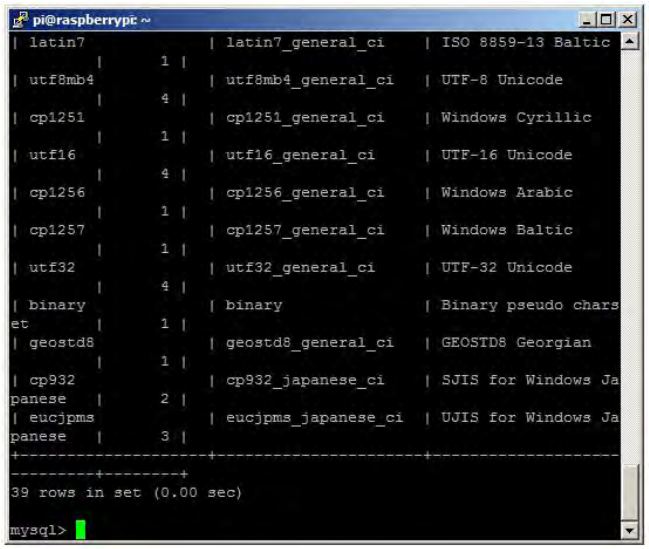
This produces one extreme of output – i.e. nothing, an 'Empty set'. Notice also the timing here – the Pi has taken four one hundredths of a second to produce this output.



The other option is represented by:-

```
pr0mpt > select * from
information_schema.CHARACTER_SETS;
```

This produces what appears to be a mess, and it is, but it should look like this:



All the | + and – characters are actually border lines in a crude table that looks like this:-

col_1	col_2	col_3	col_4	col_5	col_6
data	data	data	data	data	data
data	data	data	data	data	data

Creating a database

Now the hard bit. MySQL can become very complex. Let's create a database called magpi type:-

```
pr0mpt > create database magpi;
```

Prove the database has been created:-

```
pr0mpt > show databases;
```

You now want to tell the system to use your new magpi database so type:-

```
pr0mpt > use magpi;
```

Creating a table

A database can contain many tables, as we have seen. Tables only need to be set up once but need to specify the data-type and size of the data it is expected to hold.
A generic command looks like this:-

```
pr0mpt > create table <tablename>
(
<col1><col1_data-type>,
<col2><col2_data-type>,
<col3><col3_data-type>,
……,
);
```

A more useful example:-

```
pr0mpt > create table birdtable
(
id int NOT NULL AUTO_INCREMENT,
first_name char(25),
town char(30) default "Cardiff",
dob date,
birds int,
record timestamp default now(),
primary key (id)
);
```

Now you can use `describe birdtable;` to reveal the by now familiar description:-

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
first_name	varchar(25)	YES		NULL	
town	char(30)	YES		Cardiff	
dob	date	YES		NULL	
birds	int	YES		NULL	
record	timestamp	NO		CURRENT_TIMESTAMP	

6 rows in set (0.01 sec)

Next time we will be looking at inserting data into pre-populated tables and viewing the data using sorting, searching and logical expressions.

Article by Richard Wenner
This article is based on http://pr0mpt.me
- a free video resource.

The MagPi What's On Guide

Want to keep up to date with all things Raspberry Pi in your area?

Then this new section of The MagPi is for you! We aim to list Raspberry Jam events in your area, providing you with a Raspberry Pi calendar for the month ahead.

Are you in charge of running a Raspberry Pi event? Want to publicise it?
Email us at: editor@themagpi.com

Bloominglabs Raspberry Pi Meetup

When: First Tuesday of Every Month @ 7:00pm
Where: Bloomington, Indiana, USA

Meetings are the first Tuesday of every month starting at 7:00pm until 9:00pm, everyone is welcome. Further information is available at http://bloominglabs.org

Manchester Raspberry Jamboree

When: Saturday 9th March 2013 @ 10:00am
Where: Manchester Central Conference Venue, M2 3GX, UK

The meeting will run from 10:00am until 4:00pm and there are a limited number of places. Tickets and further information are available at http://raspberryjamboree.eventbrite.com

Norwich Raspberry Pi User & DEV Group

When: Saturday 8th December 2012 @ 12:00pm
Where: House Cafe, 52 St. Benedicts Street, Norwich, UK

The meeting will run from 12:00pm until 5:00pm. Further information is available at http://norwichrpi.org

CERN Tarte au Framboise

When: Saturday 19th January 2013 @ 9:30am
Where: CERN Microcosm, Geneva 1211, Switzerland

This event takes place at the world-famous CERN. Doors open at 9:30am and the meeting runs until 4:30pm. Further information is available at http://cern-raspberrypi.eventbrite.fr

the python pit

Python can launch subprocesses that function separately. Using this approach it is possible to create any number of desktop widgets.

We will create two widgets: a simple RSS news reader and an image downloader for Astronomy Picture of the Day.

Run `magpi_widgets.py`. This will start the widgets `widget_image.py` and `widget_rss.py`. A single CTRL+C in the terminal will send a kill command to each subprocess. This code requires some additional Python modules which can be installed from the terminal:

```
sudo apt-get install python-setuptools
sudo easy_install-2.7 pip
sudo pip install feedparser # parses RSS
sudo pip install beautifulsoup4 # parses HTML
sudo apt-get install python-imaging-tk # provides image manipulation
```

magpi_widgets.py :

```
# Python Widgets using pygame and subprocess
# By ColinD - 02 November 2012

import subprocess, os, signal, Tkinter, time

# run the widget subprocesses - feel free to add more here!
pImg = subprocess.Popen(["python","widget_image.py"],stdin=subprocess.PIPE)
pRss = subprocess.Popen(["python","widget_rss.py"],stdin=subprocess.PIPE)

# send the screen width to the sub processes
r = Tkinter.Tk()
width = r.winfo_screenwidth()
pImg.stdin.write(str(width)+"\n")
pRss.stdin.write(str(width)+"\n")

# Run until subprocesses killed with a single CTRL-C
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    os.kill(pImg.pid, signal.SIGKILL)
    os.kill(pRss.pid, signal.SIGKILL)
```

widget_image.py :

```
import urllib, Image, pygame, os, sys, time
from bs4 import BeautifulSoup

# read stdin from parent process and calculate widget screen position
baseXPos = int(sys.stdin.readline()) - 200 - 10
os.environ['SDL_VIDEO_WINDOW_POS'] = str(baseXPos) + "," + str(30)

# display a borderless window to contain  the resized image
windowSurface = pygame.display.set_mode((200,200), pygame.NOFRAME)

while True:
    try:
        soup = BeautifulSoup(
                urllib.urlopen('http://apod.nasa.gov/apod/astropix.html'))
        # APOD has one img tag so we can use find instead of findAll
        imgTag = soup.find('img')
        imgUrl = imgTag['src']
```

PYTHON VERSION: 2.7.3rc2
PYGAME VERSION: 1.9.2a0
O.S.: Debian 7

TESTED!

```
            imgName = os.path.basename(imgUrl)

            # if the image already exists then do not redownload
            if not os.path.exists(imgName):
                urllib.urlretrieve("http://apod.nasa.gov/apod/"+imgUrl,imgName)

            # download, resize and save the image for the widget
            imgOriginal = Image.open(imgName)
            imgResized = imgOriginal.resize((200, 200), Image.NEAREST)
            imgResized.save(imgName)

            imgLoad = pygame.image.load(imgName)
            windowSurface.blit(imgLoad, (0,0))
            pygame.display.update()
        # if an exception occurs skip the download on this loop
        except (IOError, TypeError, RuntimeError):
            print "Error downloading, will try again later"

        # sleep for 8 hours as we do not want to spam the server!
        time.sleep(28800)
```

widget_rss.py :

```
import pygame, os, sys, feedparser, time
pygame.init()

# read stdin from parent process and calculate widget screen position
baseXPos = int(sys.stdin.readline()) - 300 - 10
os.environ['SDL_VIDEO_WINDOW_POS'] = str(baseXPos) + "," + str(430)

# create the Pygame window and fill the background with colour blocks
screen = pygame.display.set_mode((300,150))
pygame.draw.rect(screen,(80,140,80),(0,0,300,50))
pygame.draw.rect(screen,(80,80,80),(0,50,300,50))
pygame.draw.rect(screen,(160,160,160),(0,100,300,50))

# define the font to output the RSS text
font = pygame.font.SysFont('dejavuserif', 10, True)

while True:
    myFeed = feedparser.parse('http://www.raspberrypi.org/feed')

    # set the window title to be the name of the blog
    pygame.display.set_caption(myFeed['feed']['title']+" RSS")

    # get the articles from the RSS and output the text
    for i in range(0, 3):
        layerText = pygame.Surface(screen.get_size())
        outputText = (myFeed['items'][i].title, myFeed['items'][i].updated,
                      myFeed['items'][i].link, myFeed['items'][i].description)
        # clear the surface each loop by filling with transparent pixels
        layerText.set_colorkey((0,0,0))
        layerText.fill((0,0,0))

        j = 0
        for line in outputText:
            j = layerText.get_rect().y + j + 5
            text = font.render(line.rstrip('\n'), 0, (255,255,255))
            textpos = text.get_rect()
            textpos.x = layerText.get_rect().x + 5
            textpos.y = textpos.y+j
            layerText.blit(text, textpos)
            screen.blit(layerText, (0, 50*i))
            pygame.display.flip()
            j = j +5
    # sleep for an hour, do not spam the server!
    time.sleep(3600)
```

Ideas for improvement

The widgets display will only refresh when checking for new content to download, causing blank output if another window is dragged on top. Try using Python's datetime to determine when to download new content while separately updating the screen output every second.

The Year of The MagPi

I first proposed the idea of The MagPi magazine back in March on the Raspberry Pi forums after reading numerous threads by those new to programming who wanted to learn but were unsure of how to start or whether they had enough experience to operate the Raspberry Pi.

At the time, there was no documentation how to use this clever little computer and most tutorials and projects were aimed at those with some experience in using Linux or with a background in programming. I felt this might put a lot of beginners off learning how to program or prevent them from realising the vast variety of projects that would be possible with limited effort with the Pi.

To target the more experienced programmer, one thing I felt was missing from the Raspberry Pi community was a central hub, aside from the forums, which would allow those to explain in detail to others how to replicate their project, show case pictures and videos and answer any questions from the community on the topic to help others in a similar situation.

Coming from a medical background with experience in publishing papers in journals, I felt that the best way the community could share their experience was to create a centralised place where everyone could access and share information. The media of an online peer reviewed journal seemed to tick all the boxes and comply with the educational ethos of the Raspberry Pi Foundation.

Over the last 8 months, both The MagPi magazine and the team has evolved. We are now a limited company and work very closely with the Raspberry Pi Foundation.

We encourage users to send their projects and we upload them to the draft of the issue. Readers get an early glimpse of the next magazine and they can provide suggestions to correct or clarify what is written and any tips or tricks which may be of benefit.

We currently produce a 32 page monthly magazine. Our content covers articles on a variety of Raspberry Pi related themes including coding, robotics, home automation, electronics and practical techniques (to name a few).

We are thankful for all the support given to us by both the Raspberry Pi Foundation and all our readers. We have been overwhelmed by how we have been positively received, with constant feedback to reflect this and some great articles featuring the magazine, including those by the Wall Street Journal and the BBC's Rory Cellan-Jones!

We have grown in size as a team, with editors and contributors from all over the world, from all age groups and all professions; from educationalists to medicine, technologists to students. To reflect the global interest in The MagPi magazine, it has been translated to French and German with Chinese and Spanish translations also being worked on.

We're super excited by our plans for The MagPi in 2013, including our Kickstarter project allowing readers the chance to get all eight issues in a limited edition binder. We hope you will continue to support us in the New Year.

Ash Stone
Chief Editor of The MagPi

editor@themagpi.com