ISSUE 10 - MAR 2013

Get printed copies at themagpi.com

The MagPi

A Magazine for Raspberry Pi Users

Happy Birthday!

This Issue...

- WebIOPi Framework
- Expansion boards
- Backup SD cards
- Scratch fractals
- BASH basics
- Charm

Win a 512Mb Model B Raspberry Pi & case!

ISSN 2051-9990

The MagPi

http://www.themagpi.com

Welcome to isssue 10,

Thank you to all of those who have ordered volume one (issues 1-8). After some printing delays, the cordex binder for volume 1 has been submitted for rapid production. The final pdf files needed for printing each of the magazines are nearing completion. We expect to be releasing pdfs for printing this week. We are very grateful for your continued patience. Such delays are not expected with volume 2, since all of the articles are already in the correct format for printing.

In this month's issue there is a great selection of hardware and software projects. We are pleased to present part 2 of the WebIOPi and backup articles, new programming langauges and our regular Scratch and Python pages. For those looking for automation solutions, there is one article with a whole range of extension boards.

We are on the look out for willing volunteers to help with layout, testing and proof reading activities. If you can dedicate some time, please email editor@themagpi.com.

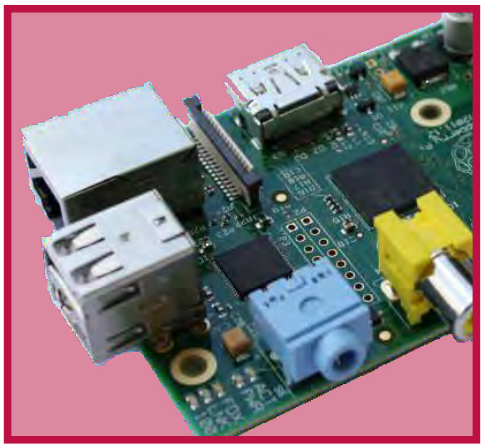
Chief Editor of The MagPi

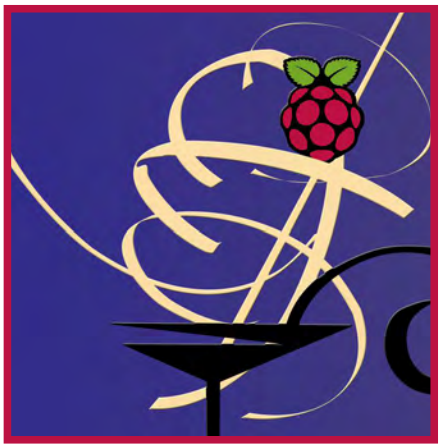
The MagPi Team

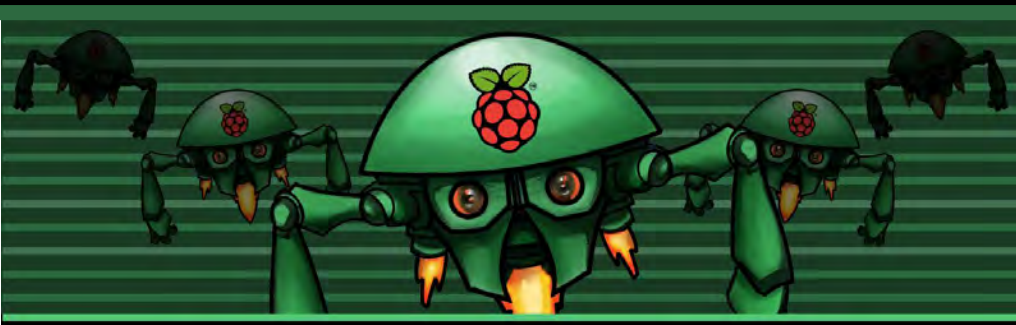
Ash Stone - Chief Editor / Administration
Chris 'tzj' Stagg - Administration
Matt '0ther0judge0' - Website / Administration
Tim 'meltwater' Cox - Administration
Lix - Administration
Aaron Shaw - Page Design / Graphics
Bryan Butler - Page Design & Theme / Graphics
Ian McAlpine - Page Design / Graphics
Isa McKenty - Page Design
Simon Johnson - Page Design
Steve Drew - Page Design
W.H. Bell - Page Design / Administration
Mark Robson - Proof Reading
Michael Beaucage - Graphics

Contents

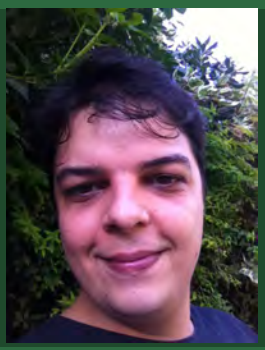
HAPPY BIRTHDAY RASPBERRY PI!







Remote Controlled Robot Cam, Part II



Eric PTAK

Guest Writer

DIFFICULTY : ADVANCED

Building the interface

Building your own interface is also easy, and is based on a HTML file embedding some Javascript. You only have to load the webiopi.js file from your HTML file to use the WebIOPi power. Create a new index.html file next to your Python script:

```
<html>
  <head>
    <title>CamBot</title>
    <script type="text/javascript"
src="/webiopi.js"></script>
    <script type="text/javascript">

      // Javascript code goes here

    </script>
  </head>
  <body>
    <div id=”box" align="center">
    </div>
  </body>
</html>
```

Take note of the starting slash when loading webiopi.js, to ensure it will be searched in the root of the server or it may be not found.
I added an empty script section; we will use the WebIOPi JS library here. There is also a div box, which will contain controls.

In the script section, we add an init function to build the interface using WebIOPi library. It contains many functions to ease creation of buttons that control GPIO. Here we use a basic button to call a different function on press and release. Each function calls a different macro on the server. Don’t forget to register the init function to WebIOPi. It will be called when everything is loaded and ready.

```
function init() {
  var button =
webiopi().createButton(
  "bt_up",    // id
  "/\\",      // label
  go_forward, // press
  stop);      // release

  $("#box").append(button);
}

function go_forward() {
  w().callMacro("go_forward");
}

function stop() {
  w().callMacro("stop");
}

webiopi().ready(init);
```

Be careful that webiopi() is a function and a reserved word that need brackets to return the WebIOPi object. You can use w() to short the webiopi() call.

It's now time to start the server and enjoy the interface. Open a terminal in the folder you created Python and HTML files and execute the script:

```
$ sudo python yourscript.py
```

Open a browser to the webiopi to control the chassis. Hold the button to go forward and release it to stop. The last piece missing is the webcam.

Add a webcam stream

There are many possibilities to stream a webcam, which may depend on the model you have. In my case, I have a recent webcam which outputs both RAW and MJPEG formats up to 1280x720@30fps.

First, check your webcam is installed with a terminal:

```
$ lsusb
[...]
Bus 001 Device 005: ID 046d:0825
Logitech, Inc. Webcam C270

$ ls /dev/video*
/dev/video0
```

Then, to check it's working, you can install uvccapture using apt-get or aptitude and take a single snapshot:

```
$ uvccapture -v
Using videodevice: /dev/video0
Saving images to: snap.jpg
Image size: 320x240
Taking snapshot every 0 seconds
Taking images using mmap
Resetting camera settings
Camera brightness level is 0
Camera contrast level is 255
Camera saturation level is 255
Camera gain level is 255
Saving image to: snap.jpg
```

If uvccapture returns without error, we can continue to stream the webcam.

I use MJPG-STREAMER, which is really easy to use. It gives me a 320x240@25fps pass-through MJPEG stream over HTTP. I tried FFMPEG but it takes the RAW output of the webcam to encode it in MJPEG with a framerate under 5fps.

You can download MJPG-STREAMER at http://sourceforge.net/projects/mjpg-streamer/

You will also need libjpeg8-dev you can install using aptitude/apt-get.

Uncompress and build MJPG-STREAMER using make command. Then execute it:

```
$ ./mjpg_streamer -i "./input_uvc.so
-r 320x240 -f 25" -o "./output_http.so
-n -p 8001" &
```

Back to HTML file, add a img tag with src set to http://raspberrypi:8001/?action=stream replacing raspberrypi by your Pi's IP. You can also directly try the URL in your browser.

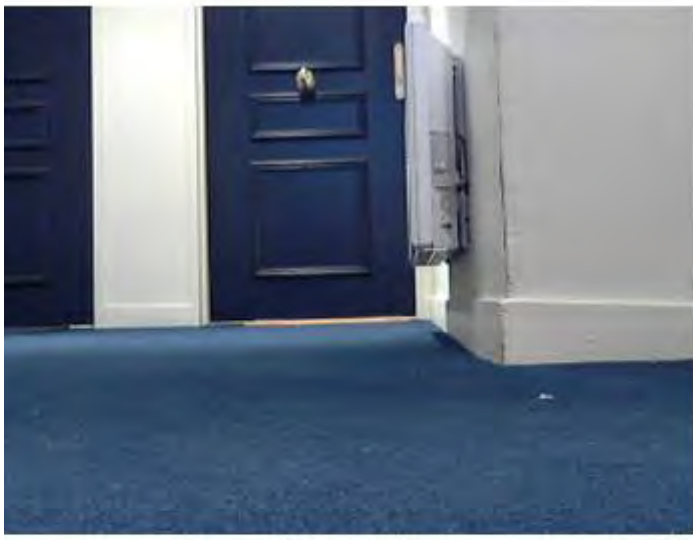
```
...
<img
src="http://raspberrypi:8001/?action=s
tream">
</body>
</html>
```

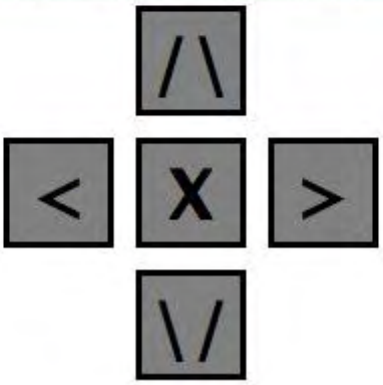
Conclusion

With this article, you learned how to install WebIOPi and how to use it in your own Python scripts to write macros you can call from the web.

The code is incomplete as it only allows to go forward and to stop. Just add left/right_backward, turn_left/right and go_backward functions to move the robot in all directions.
You can download the complete code at http://files.trouch.com/webiopi/cambot.zip. You will find more information on the project wiki and in the examples folder of WebIOPi archive.

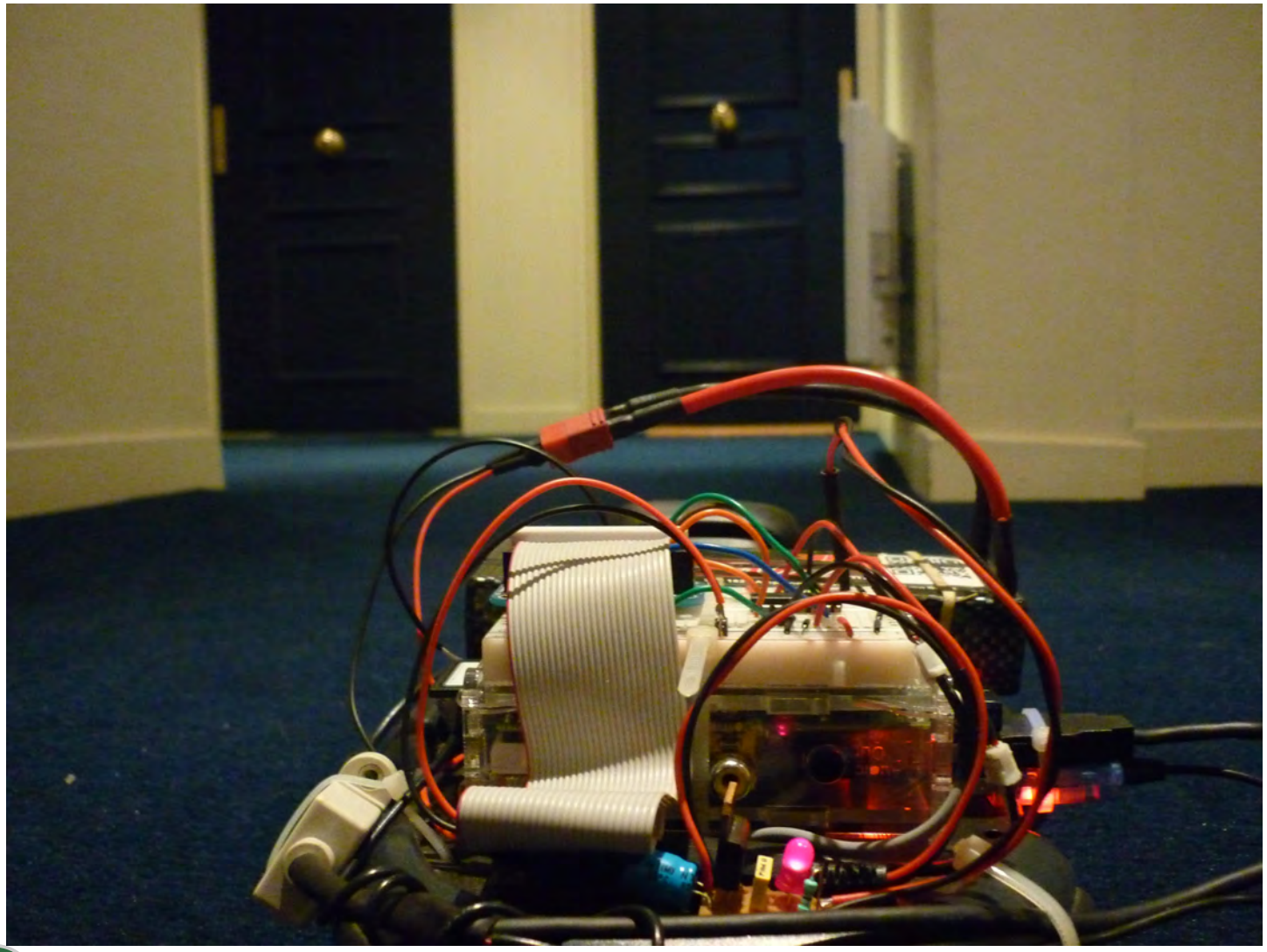
Eric PTAK, creator of WebIOPi
http://trouch.com
http://code.google.com/p/webiopi/





NOTE:

Part I of this tutorial appeared in the last issue of the MagPi. Please read Part I before attempting what is shown here. You can download Issue 9 at: www.themagpi.com



TRY A COCKTAIL OF PROJECTS

Lloyd Seaton
Guest Writer

This is the first of a series of articles intended to assist Raspberry Pi users to construct and use expanded I/O capabilities of their Pi.

DIFFICULTY : ADVANCED

Suitability

These are relatively advanced constructional projects and are not recommended for beginners. There are no kits of parts being offered. Instead, participants will need to purchase their own components from their preferred electronic component suppliers. However, lists of components and suggested suppliers will be provided.

"PCBs can be purchased more economically...if participants combine into groups of 2 or 3 (or more)"

The Power Of Groups

The economics of component purchasing are greatly influenced by the quantities involved. For these projects, participants should be able to achieve substantial savings if they can band together with like-minded individuals through school, Raspberry Jam or other organisations to combine their purchases.

Printed Circuit Boards (PCBs)

Most constructors shrink from the challenge of designing and producing a circuit board with any fineness of detail or complexity. That is the reason for this series of articles. By grouping together a "cocktail" of projects with a variety of PCB designs, the PCBs can be purchased more economically, especially if participants combine into groups of 2 or 3 (or more) to arrange their purchases.

PCB Manufacturing Strategy

The author became involved in the design and construction of small printed circuit assemblies (PCAs) late in 2011 when he decided to develop an electronic supervisory circuit to manage solar powered pumps on farms and elsewhere. It soon became apparent that the cost of PCBs needed to reduce and a decision was made to use the Mini Board Pro service of ExpressPCB and to design the project PCBs such that multiple (initially 2, later 3) project PCBs could fit on each of the 3 manufactured PCBs per order. The project PCBs are designed to be separated by hacksaw prior to assembly. This strategy achieved a unit price of about $15 Australian per

project PCB, seemingly a reasonable price. More recently, the supervisory circuit has been redesigned (with the help of a Raspberry Pi) to employ an ATtiny85 microcontroller instead of the previously pure analogue circuitry such that 6 project PCBs can now be fitted on each manufactured PCB, halving the PCB unit price again to about $7.50, a very satisfactory price.

The Projects

During the coming months it is intended that the following PCB projects will be covered:

1. Power I/O

2. Tiny I/O

3. MegaPower

4. Pi Bridge ICSP Interconnect

5. Battery Load Manager 85

6. BatteryLoadManager+

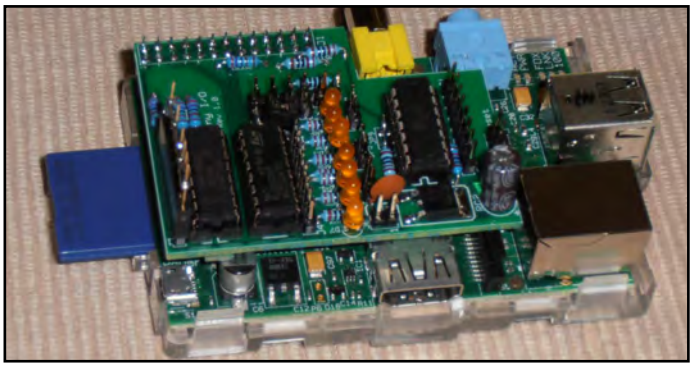
Brief descriptions of each project follow.

Power I/O

This PCB can include a hexadecimal rotary switch, 7 Darlington drivers with indicator LEDs and buffered access to the Pi's GPIO ports. A switching regulator circuit can optionally be included to allow the Pi to operate from a DC source of 7 to 20V.

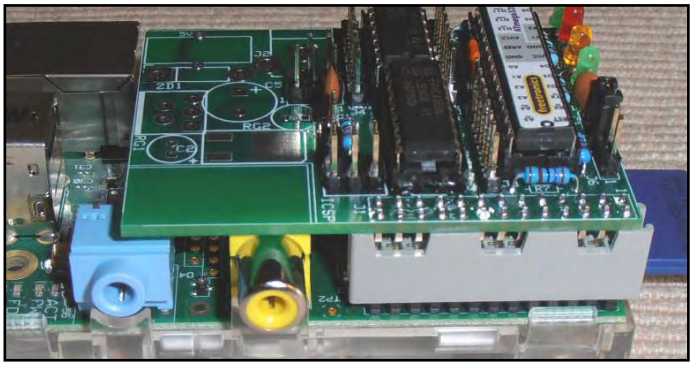
Tiny I/O

This PCB can include an ATtiny84 microcontroller, 7 Darlington drivers with indicator LEDs and buffered GPIO access.



MegaPower

This PCB is for people who are not interested in using the Raspberry Pi's GPIO, preferring instead to have an ATmega328 microcontroller with 4 indicator LEDs, 7 Darlington drivers and operating at 5V for broad interfacing capability. The board can serve as a close companion to a Raspberry Pi or operate on its own. When coupled with a Raspberry Pi, the MegaPower's on-board switching regulator circuit can power itself and the Pi from a 7 ~ 20V DC source. If not required, the switching power supply components can be omitted, leaving the ATmega328 to rely on the Pi's regulated power supplies.



Pi Bridge ICSP Interconnect

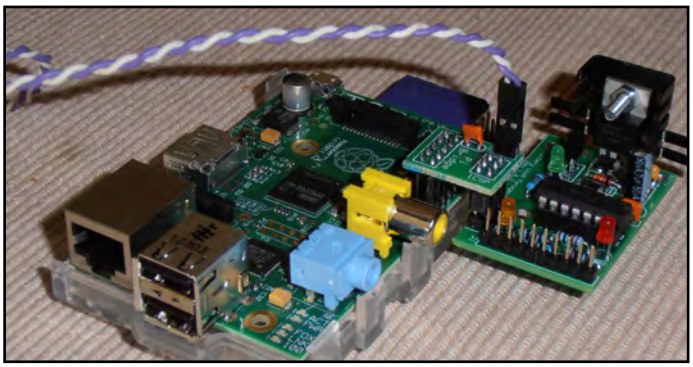
This little PCB (0.8" x 0.6") provides a convenient way to connect the Raspberry Pi for programming of the Battery Load Manager 85, BatteryLoadManager+ or some Arduino boards without the use of jumper wires.

Battery Load Manager 85

This PCB is for a stand-alone ATtiny85-based design (1.25" x 1.25") that can be programmed via the Pi Bridge ICSP Interconnect and a Raspberry Pi set up with Gordon Henderson's Arduino IDE procedure and extensions together with ATtiny support. Circuitry is included for monitoring of battery voltage (+ optionally another voltage) and switching of a high current load according to user policy programmed via an Arduino sketch. There are 3 indicator LEDs. An example Arduino sketch will be available for use as a solar pump supervisor.

BatteryLoadManager+

This PCB is for a stand-alone ATtiny84-based design (1.9" x 1.25") that is otherwise similar to the Battery Load Manager 85 but has additional capabilities. Surplus ATtiny84 I/O pins are brought out to a header for increased flexibility of application.



The Cocktails - What's In The Mix?

One size rarely fits all. To try to assist in the most cost-effective purchase of PCBs it is intended that there will be a choice of "cocktail files" available so that each constructor or group of constructors can choose to use the cocktail files that most closely match their interest in the various project PCBs. For example, if a constructor has little interest in stand-alone ATtiny PCBs, it would not be cost-effective to choose a cocktail file that commits significant space/cost to the Battery Load Manager projects.

Pictured below is an ExpressPCB Mini Board Pro delivery. Each of the 3 manufactured PCBs is 2.5" x 3.8" and is allowed to have no more than 350 holes. These manufactured PCBs will each yield a pair of different project PCBs which happen to be equal in size; one for the Tiny I/O project, the other for MegaPower. There are guide lines to assist with the hacksawing.

Procedure For Ordering PCBs

Before ordering PCBs it is necessary that you first install the ExpressPCB free CAD software on a PC with Internet access. The ExpressPCB.com software is a free download from the Internet and is intended for Windows machines but the author uses it with Wine on a Linux PC quite satisfactorily. You then need to choose your cocktail file, download the file and open it with the ExpressPCB software. Placing the order is then a straight-forward procedure whose full description can be found on the information blog.

Information Blog

An Internet blog has been established at **picocktails.blogspot.com** to provide an on-going information resource for constructors of these projects. Links to the design documentation, cocktail files, photo albums, test programs etc will be readily accessible from the blog.

MARCH COMPETITION

Once again The MagPi and PC Supplies Limited are proud to announce yet another chance to win some fantastic Raspberry Pi goodies!

This month there are three prizes!

The first prize winner will receive a new 512Mb Raspberry Pi Model B plus a PCSL Raspberry Pi case!
The second and third prize winners will receive a PCSL LCD VESA mount case.

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

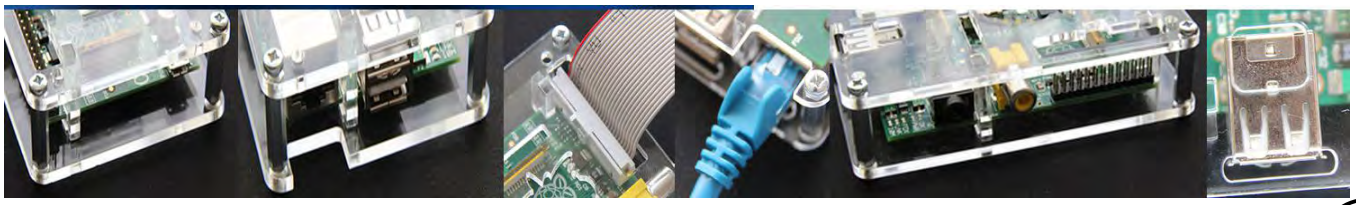
Closing date is 20th March 2013.
Winners will be notified in next month's magazine and by email. Good luck!

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

February's Winners!

The winner of the new pre-assembled Element 14 Gertboard is **Nicolas Penin (Mulhouse, France)**.
The 2nd and 3rd prize winners of the PCSL Raspberry Pi case are **Mervyn Burnett (Maryport, UK)** and **Beverley Skea (Liverpool, UK).**

Congratulations. We will be emailing you soon with details of how to claim your prizes!



Backing up - part 2

DIFFICULTY : MEDIUM

Norman Dunbar

Guest Writer

In part 1, I demonstrated how you could make a backup of your Raspberry Pi's SD card. In part 2, I shall demonstrate how you may check that the backup worked and also, I shall show you some of the "fun" that can be had using the backup image as a disc drive. You will see how to modify files within the backup image itself. When you subsequently write the image to an SD card, all your changes will be present.

Again, this is simple to do on a Linux computer, however, for Windows users it's not easy at all. Windows users should consider using Linux Live CD or running Linux in a VirtualBox VM (or similar) if you want to follow along.

Checking the Backup File

Making a backup is simple, normally, but how do you know that the backup has worked? It is not fun when you have a trashed Raspberry Pi SD card and you discover that your backup is not able to be restored. You have to begin again from scratch by building a new default distribution and reinstalling and configuring all your software. Of course, none your own files will be able to be restored. So you should always check your backups work.

The simplest method of checking is to write the image to a spare SD card and plug that in to the Rasberrp Pi and reboot it. This is also about the quickest method, and the part 1covered this in some detail.

This article looks at a method whereby you can take the backup image and pretend that it is a real device, an SD card if you like, and use it accordingly as if it was a real SD card.

Please note, all of the following assumes that the backup image is uncompressed and resides on a local hard disc. If you have a compressed image, then you may wish to uncompress it if you are following along.

The first step, as root – as ever – is to determine where the partitions begin. Because we have an image copy of the SD card, we can look at the file itself and see the partition table within. If you wish to run all the following commands as the Pi user, the please prefix each one with sudo. Otherwise, make life simple and type:

```
$ sudo sh
```

This command will start a root shell for you, and there will be no need to prefix the commands with sudo.

First list the backup image to see where the partitions begin and the sizes of the sectors. Fdisk reports sizes in sectors, but handily displays the sector size at the top.

The command to list the partition details is:

```
$ fdisk Rpi_8gb_backup.img
```

```
Welcome to fdisk (util-linux 2.21.2).
...
Command (m for help): p

Disk Rpi_8gb_backup.img: 7948 MB, 7948206080 bytes
255 heads, 63 sectors/track, 966 cylinders, total 15523840 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000dbfc6

Device                    Boot     Start        End  Blocks  Id  System
/BU/Rpi_8gb_backup.img1             8192     122879   57344   c  W95 FAT32 (LBA)
/BU/Rpi_8gb_backup.img2           122880   15523839 7700480  83  Linux
```

The output from this command is shown at the top of the following page.

When the utility has loaded, we use the 'p' command to print the current partition table. This is the command that shows details of the current partitions, where they are located and how big they are.

We can see from the line starting with ‘Units’ or ‘Sector size’ that a sector is 512 bytes. We can also see that our two partitions start at sectors 8,192 and 122,880. Multiplying these start sectors by 512 gives the start position in bytes. This works out at 4,194,304 and 62,914,560 bytes respectively, however, we don't really need these latter figures as Linux will calculate it for us.

On your Raspberry Pi, the first partition is normally mounted at /boot while the second is the root (/) mount point.

To check these, without needing access to the Pi, we need to create a pair of mount points, as follows:

```
$ mkdir /mnt/root /mnt/boot
$ chmod a=rwx /mnt/root /mnt/boot
```

The two commands above only requires to be executed once, the very first time we attempt this exercise. Next, and on any subsequent occasion when we do this, the following two commands should be typed - on one line each - to mount the backup image's two partitions as pseudo disc drives:

```
$ mount -t vfat -o loop,offset=$((8192 * 512))
/BU/Rpi_8gb_backup.img /mnt/boot
$ mount -t ext4 -o loop,offset=$((122880 *
512)) /BU/Rpi_8gb_backup.img /mnt/root
```

The above creates a couple of mount points (directories) and then mounts the first partition within the image file as a vfat file system on /mnt/boot and then mounts the second partition as an ext4 file system on /mnt/root.

The first two commands to create the mount points and set the permissions on them are only required once, the first time you carry out this exercise.

You can now see the files by opening a file manager and looking at the /mnt/boot and /mnt/root directories – you should see your various files as if you were looking on your Raspberry Pi.

At least you are now sure that your image file could be restored to an SD card, and that it is at least mountable - so it should be ok for future use if you ever require to restore a corrupted card. However, with the card image currently mounted as a pseudo drive on your Linux laptop, you can treat it exactly as if it was a real drive, and edit files, create new ones, delete ones you no longer require, and so on.

Edit a file

If, for example, you were about to restore this backup image to a second SD card, but for use in a Pi connected to a TV that has VGA input rather than HDMI, you could edit the config.txt file, in the /boot partition, inside the image before you write it to the new SD card.

As you have the image's first partition, the one normally mounted at /boot, mounted on your laptop as /mnt/boot, then all you have to do is edit /mnt/boot/config.txt using your favourite editor, and set hdmi_group and hdmi_mode to 1 and save the

file.

When you subsequently write this backup image to a new card, it will be ready to run on a Pi connected to your VGA TV.

Why would you want to do this? Well, You might have a couple of Pis running identical setups, but one is in the main computer room attached to an HDMI TV or monitor, and another in the Kids bedroom, connected to a VGA TV or monitor. This method will save you having to backup one of the devices, write the SD card for the other, boot it up attached to the "wrong" display, change the config and then boot it on the correct display.

Obviously, if you have configured both of the devices to have a static IP address when connected to your network, you will have to edit /etv/network/interfaces to suit the device which is having the SD card re-imaged, otherwise you will end up with two devices runing the same IP address, which can only result in problems.

Restore a single file

This method of mounting a disc image as a device is useful for times when you manage to delete a file, accidentally of course, on your Pi, but you know that you have a backup. If the Pi is on your network, you can simply mount the backup image as above, locate the file and check that it is the one you want, then use scp or sftp to copy the file to your Pi, as demonstrated below:

```
 $ mount -t ext4 -o loop,offset=$((122880 *
512)) /BU/Rpi_8gb_backup.img /mnt/root

 $ cd /mnt/root/home/pi
 $ scp Lost_file.txt  pi@raspberrypi:
```

You will be prompted for pi's password, and then the file will be copied over to the Pi user's home directory on your Raspberry Pi.

So there you have it, how to mount your backup file on the backup computer to check that it is ok, and as a bonus, how to extract a file (or files) for an individual recover. What else can we do?

Restore a full backup

This is as simple as initialising the SD card for the first time. You can restore a backup file to your SD card provided that the SD card is bigger or the same size as the image file. As ever, the following command must be executed as root.

```
 $ dd if=Rpi_8gb_backup.img of=/dev/mmcblk0
bs=2M
```

The command should be all on one line.

It takes a while, but it will restore in the end. All you have to do is wait for the copy to finish and then boot the Pi with the restored SD card in the slot. See part 1 for details on restoring compressed and/or split backup images.

Restore to a larger card

We can also use the smaller backup files to initialize a larger SD card, if we were perhaps upgrading. This is what I had to do when I restored a 4Gb backup to my 8Gb card. Once the restore completed I had a 4Gb image on an 8Gb SD card. In order to reclaim the missing 4Gb all I did was to boot the Pi with the restored 8Gb card in place, and login as the pi user as normal.

Once logged in, I executed the sudo raspi-config command, selected the option to "Expand root partition to fill SD card" and the system happily extended the 4Gb partition to fill up the remaining free space on the card. The actual resizing is carried out as part of the next reboot of the Pi - it doesn't happen immediately.

You can see that it worked by executing the df command, which does not need to be executed as root!

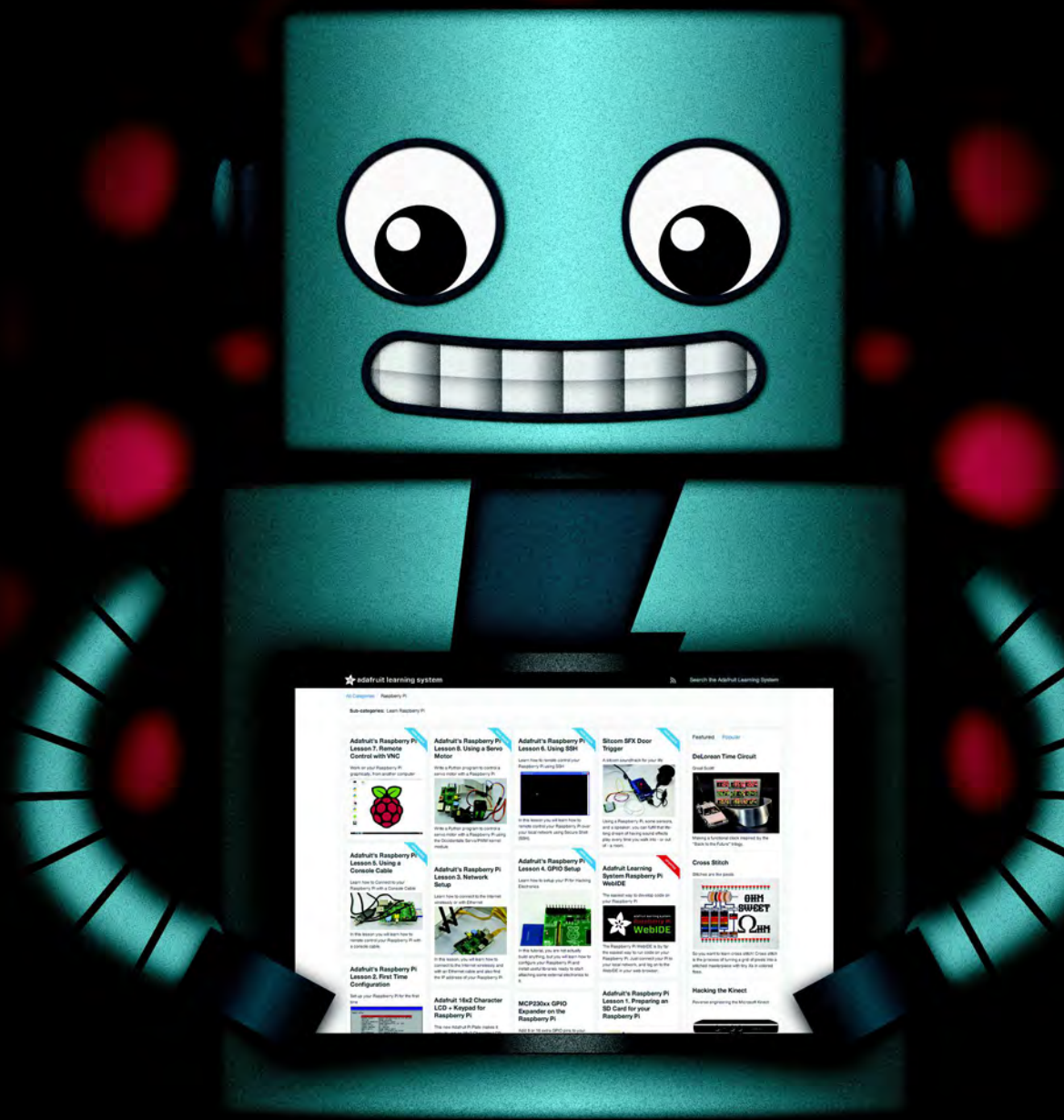
```
$ df -h /

Filesystem   Size  Used Avail Use% Mounted on
/dev/root    7.3G  1.6G  5.4G  23% /
```

The output above shows my root file system, mounted on /, is 7.3 Gb in size so I know it cannot possibly be the same size as it was when I restored from the 4Gb backup image.

adafruit learning system

Raspberry Pi WebIDE

BUILD AMAZING THINGS & LEARN HOW TO PROGRAM WITH THE RASPBERRY PI



adafruit

HTTP://LEARN.ADAFRUIT.COM/CATEGORY/RASPBERRY-PI

Bash Gaffer Tape

```
if [[ $var == 1 ]]; then
   cat /proc/cpu # Check the CPU type
elif [[ $var == 2 ]]; then
   cat /proc/meminfo # Memory information  else
   date -I # The current date
fi
```

1 - BASH basics

DIFFICULTY : BEGINNER

W. H. Bell

MagPi Writer

Bash (Bourne Again Shell) has been the default LINUX shell for several years. The aim of this series is to give an overview of the Bash shell, providing a description of the syntax and built in functions. Bash is great for lashing together several different programs with minimal overhead. For this reason, this of series of articles is called "Bash Gaffer Tape".

When reading these articles, it may prove useful to consult the Bash manual page by typing man bash. The search commands available within man and other commands can be found in the less manual page. Some Bash commands have already been discussed in the Command Line Clinic series in Issues 2 to 5.

Running Bash

When a terminal window is opened a shell interpreter is started. The default shell for the current user can be printed by typing,

```
echo $SHELL
```

The default shell is set for each user within /etc/passwd or via NIS or LDAP. For example,

```
grep pi /etc/passwd
```

returns

```
pi:x:1000:1000:,,,:/home/pi:/bin/bash
```

Scripts can be run by typing the commands directly into a terminal window or by using a shell script text file. A shell script file can be run in two ways: by sourcing the script

```
source script.sh
```

which is equivalent to

```
. script.sh
```

or by executing the script,

```
./script.sh
```

When a file is sourced, it is as if the file was typed into the current shell. Any variables which are declared in the script remain set when the script finishes. The script also has access to all of the variables declared in the current shell. In contrast, when a script is executed a new bash interpreter session is started. At the end of the bash session any local variables are cleaned up.

To execute a script, the path to the Bash interpreter should be given at the top of the file:

```
#!/bin/bash
```

Then the file should be made executable

```
chmod u+x script.sh
```

Finally, it is possible to type ./script.sh to execute the script.

Use nano (documented in the issue 3 C cave article) to create a hello.sh file containing:

```
#!/bin/bash
# A simple script to print a string.
echo "In the beginning.."
```

Then make the file executable and execute the script. The

echo command prints the string on the screen using the standard out. Strings starting with "#" are comments. Comments can be added on a separate line or at the end of a line.

Pipe operator

A series of commands can be chained together using the pipe "|" operator. A pipe has the effect of passing the standard out from one command to the standard in of another command. This is especially useful when handling strings,

```
# Print "Hello Joe", replace Joe with Fred.
echo "Hello Joe" | sed 's/Joe/Fred/g'

# Replace Hello with Goodbye too.
echo "Hello Joe" | sed 's/Joe/Fred/g' | sed
's/Hello/Goodbye/g'
```

In this example the sed command is used to replace a part of the string. The sed command (stream editor) is a program in its own right and has a separate manual page.

Redirection

The standard output from a program can be directed to a file or a device,

```
# Print a string to a file
echo "This is a file" > file.txt

# Print the contents of the file on the screen
cat file.txt
```

The operator ">" truncates the file and then appends the standard output to the file. To append to a file without truncation, the ">>" operator should be used.

If a command produces a lot of output which is not needed, the output can be sent to /dev/null instead:

```
# Run a command, but throw away the output
rm /tmp &> /dev/null # This command will fail.
```

A file can be used as the standard input of a command by using "<". This will be discussed later in the context of loops.

Variables

A variable is defined by assigning it a value,

```
myName="JohnDoe"
```

Bash is very sensitive to the use of white spaces. For the declaration to be interpreted correctly there must not be any spaces between the variable name and the equals sign or the equals sign and the value.

Once a variable has been defined, it is used by prepending the name with a dollar sign,

```
echo $myName
```

Variables which are defined in one shell are not available in a sub-shell unless they are exported,

```
export myName="JohnDoe"
```

where the variable can be exported when it is declared or afterwards.

if-else conditions

Logic conditions are inclosed in "[[]]" parentheses. The status of a variable can be tested using a logic condition,

```
#!/bin/bash
if [[ -z $myName ]]; then
  echo "myName is not defined"
else
  echo "myName is defined as \"$myName\""
fi
```

In this case the first condition is true if the variable is not set. At least one white space must separate the pieces of the logic condition. Save this script, change its permissions and execute it. Then try

```
export myName=$HOSTNAME
```

and run the program again. Then type

```
unset myName
```

and run the program again. The unset command removes the variable myName. Bash also provides else-if statements:

```
if [[ $var == 1 ]]; then
  cat /proc/cpu # Check the CPU type
elif [[ $var == 2 ]]; then
  cat /proc/meminfo # Memory information
else
  date -I # The current date
fi
```

A summary table of logic tests which can be applied to a variable are given below,

Syntax	Meaning
-z *string*	True if length of *string* is zero
-n *string*	True if the length of *string* is non-zero
var1 == *var2*	True if equal
var1 != *var2*	True if not equal

The logic comparisons of equal and not equal can also be used with wildcard syntax, to check if a sub-string is found in another string.

```
#!/bin/bash
str1="Raspberry"
str2="berry"
if [[ $str1 == "$str2"* ]]; then
  echo "$str1 begins with $str2"
fi
if [[ $str1 == *"$str2" ]]; then
  echo "$str1 ends with $str2"
fi
if [[ $str1 == *"$str2"* ]]; then
  echo "$str1 contains $str2"
fi
```

The 'for' loop

Bash provides many familiar loop structures. The for loop is most commonly used with input files or variables,

```
#!/bin/bash
# A string with values and spaces
list="a b c"
# print each character in the 'list' variable.
for l in $list; do
  echo $l
done
```

This can also be written on one line as

```
list="a b c"; for l in $list; do echo $l; done
```

where several of the newline characters in the script file are replaced with semi-colons. The variable list can be replaced with an input file,

```
#!/bin/bash
# Fill a file with some strings
echo "Apple Orange" > list # Truncate, append
echo "Pear" >> list # Append
# print each word in the input file.
for l in $(<list); do
  echo $l
done
```

This example uses a redirection from an input file to read each word. Bash separates the words using the space or the new line character.

For loops also support C-like iteration,

```
#!/bin/bash
# Print all numbers from 1 to 10
for (( i=1; i<=10; i++ )); do
  echo $i
done
```

Notice that the variable i is not prefixed by a dollar sign within the (()) parentheses of the for loop. This is an exception for this type of for loop.

Evaluating commands

A command can be evaluated by writing it within $(). For example,

```
dir_list=$(ls )
```

fills the variable dir_list with the text returned by the ls command. The syntax $() can be directly used as a variable. This can be useful within a for loop,

```
#!/bin/bash
for file in $(ls *.txt); do
  gzip $file
done
```

where this example gzips all of the text files in the present working directory.

Command evaluations can be nested,

```
touch /tmp/t1 # Create empty
$(basename $(ls /tmp/t1)) # File name only
```

and can include variables,

```
file=/tmp/t1
$(basename $(ls $file))
```

Each of the commands can include pipe operations,

```
files_to_gzip=$(ls * | grep -v .gz)
```

where this command excludes file names which include ".gz" from the variable. The pipe operator "|" passes the standard output from one command to the standard input of another command.

Challenge problem

Write a program to gzip all of the files in the present working directory. The program should not gzip files which have the .gz ending. The solution to the problem will be given in the next tutorial.

The MagPi What's On Guide

Want to keep up to date with all things Raspberry Pi in your area?
Then this section of the MagPi is for you! We aim to list Raspberry Jam events in your area, providing you with a RPi 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

CPC Raspberry Jamboree 2013

When: Saturday 9th March 2013 @ 10:30am
Where: Manchester Central Conference Centre, Manchester, UK

This major event will run from 10:30am until 4:00pm. If you cannot attend you can register for a webcast recording. Further information is available at http://raspberryjamboree.eventbrite.com

Bermuda Raspberry Jam

When: First Tuesday of each month @ 6:00pm
Where: Admiralty House, Pembroke, Bermuda

The meeting will run from 6:00pm until 8:00pm. Further information is available at http://bermudaraspberryusergroup.teamsnap.com or email rpi@sainib.com

Norwich Raspberry Jam

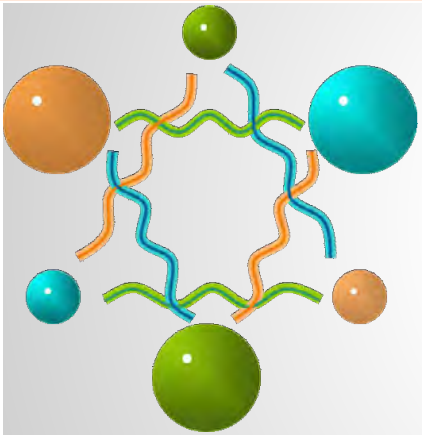
When: Saturday 9th March 2013 @ 12:00pm
Where: The Soup Lab, 5 St. Benedict's View, Grapes Hill, Norwich, NR2 4HH, UK

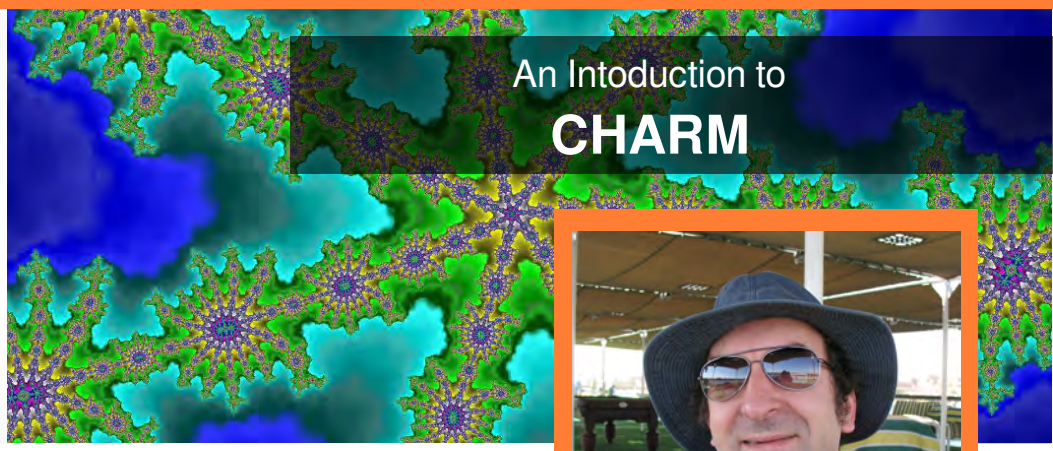
The meeting will run from 12:00pm onwards.
Further information is available at http://norwichrpi.org

Raspberry Pi Oxford Geek Night

When: Wednesday 13th March 2013 @ 7:30pm
Where: The Jericho Tavern, 56 Walton Street, Jericho, Oxford, OX2 6AE, UK

The meeting will run from 7:30pm with guest speakers featuring **Eben Upton**.
Further information is available at http://oxford.geeknights.net





Charm on the Raspberry Pi

Peter Nowosad

Guest Writer

DIFFICULTY : ADVANCED

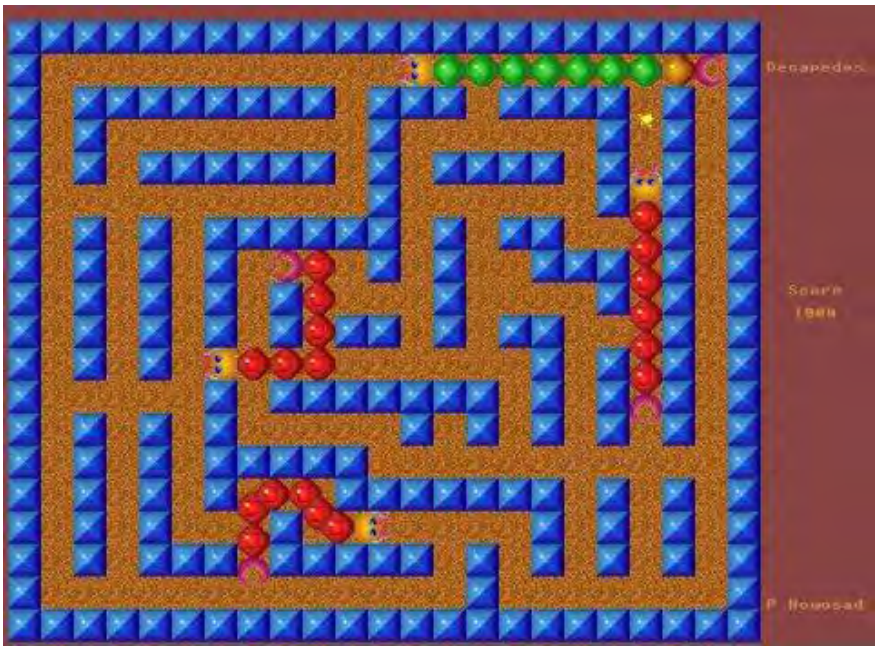
This and follow on articles are intended to promote interest in and understanding of the Charm language on the Raspberry Pi. As the author of the language, this is a goal I am keen to encourage, particularly among the younger generation of Raspberry Pi owners and users who are looking to learn a little about the mysterious world of programming.

The Charm tools are light on resource yet powerful and highly suited to an agile development environment. Applications of any complexity can be rapidly developed in small incremental changes without waiting around for the code to build; for instance the whole of the Charm distribution can be rebuilt in well under a minute!

Here I would like to cover the practical aspects of using Charm. I hope this will encourage people to take up writing Charm programs for themselves and join the over 1,200 people from around the world that have visited the Charm web site so far.

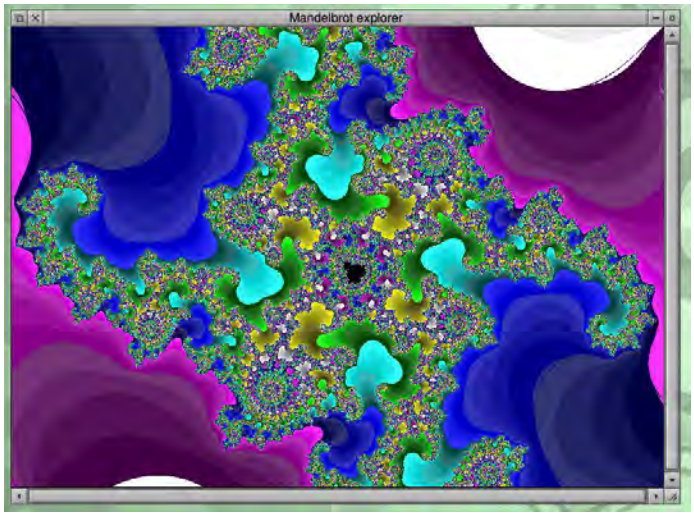
Installation

Charm version 2.6.1 is already bundled in the Programming folder of the RC6 release of Risc OS for the Raspberry Pi. I would however recommend updating to the latest version (currently 2.6.4) which is freely distributed under the Gnu Public License as a simple zip file from the Charm website charm.qu-bit.co.uk. There you can read more, get in touch with me through the Charm forum and view screenshots of the various Charm demos, of which the most fun is Decapedes, a shoot'em up somewhat akin to Pacman with some nice sound and graphics.



You may optionally wish to utilise the vector floating point

(VFP) capabilities of the ARM 11 chip which is off by default for people running Charm on emulators or older variants of the ARM chip. This will involve enabling the VFP option in the Charm shell and re-building the distribution as described on the web site. Doing this will for instance speed up the included Mandelbrot explorer program by an order of magnitude by replacing floating point emulator (FPE) instuctions with native VFP coprocessor instructions.



Tools

The set of Charm tools for Risc OS contains the following principal applications:

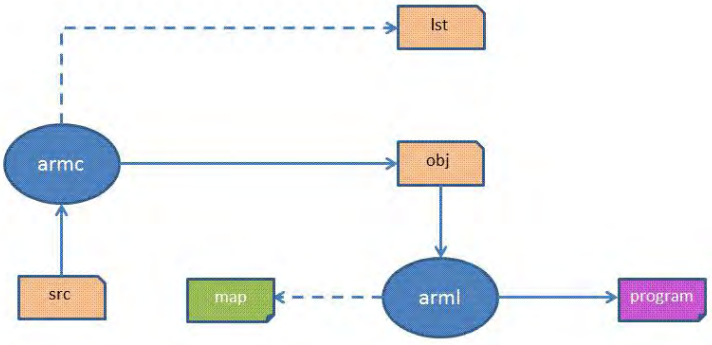
* **!Charm** - *A desktop shell application for running the tool set. The shell supports drag and drop for files and folders, command logging and error reporting.*
* **edit** - *A general purpose editor that is useful for writing and developing Charm source code (drag files or folders to edit on to the Charm shell icon with the shift key pressed).*
* **armc** - *A compiler that generates an object file in binary form from a source code file written in the Charm programming language.*
* **arma** - *An assembler that generates an object file in binary form from a source code file written in ARM assembly language.*
* **arml** - *A linker that combines Charm object files into an executable Risc OS application or module.*

New Projects

Creating new template projects in Charm is easy using the !NewProject utility application. Simply select New Project from the menu, name the project as you wish (default is MyProject) and drag the folder icon to the folder in which you wish it to reside. You can then build the project by dragging the project folder on to the Charm icon. You should then see the following in the Charm log:

Required project folders are created automatically, namely:

* **src** - *Charm language source files*
* **obj** - *object files created by the compiler or assembler*



In case you don't want to inline ARM assembler inside Charm source you can create the additional folder:

* **arm** - *Arm assembler source files*

Applications that live inside an application folder are usually linked directly to the correct !RunImage location from inside the project 'build file using the program command.

Modular Programming

The concept of modules is key to an understanding of Charm (N.B. in this context Charm modules are different from Risc OS modules!). Each module is introduced with the keyword module, and each source file that Charm compiles contains a single module definition. It may however import exported declarations from any number of other modules on which it is dependent.

References are usually to other modules in the containing project and to run time library (RTL) modules that provide essential functions such as managing windows, files, the keyboard and screen. The latter are further documented on the Charm website. The order in which modules are compiled is determined by the project 'build file which also specifies the name and location of the linked application or module.

Getting Started

So now for the first few snippets of the Charm language. Each project must contain one and only one module in which a ~start procedure with one of two specific signatures is exported.

If no command line parameters are required, the ~start procedures is defined like this:

```
module Main
{
  export proc ~start() {...startup code }
}
```

Hence, the classic hello world program in Charm which uses the vdu stream of the Out run time library module can be coded in file src.hello as:

```
import lib.Out;
module Hello {
  export proc ~start () {
    Out.vdu.str ("Hello World!\n");
  }
}
```

though you will need the project 'build file to contain:

```
module hello
program hello
```

in order to build the program before you can run it (the linker will find the Out library automatically for you).

A Practical Project

If you are up for a challenge, I suggest replacing the default !NewProject MyProject module with the following code to output the first dozen factorial numbers:

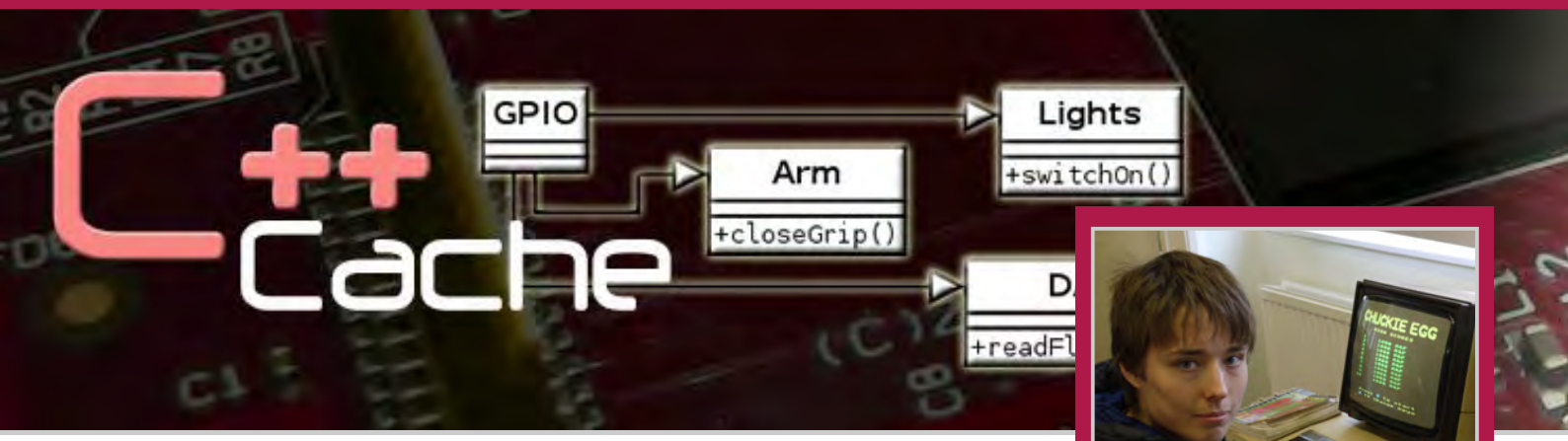
```
import lib.Out;

module MyProject
{
  proc factorial (int n) int
  {
    if n <= 1 return 1;
    return n * factorial (n - 1);
  }
  export proc start (ref array ref array char
argv)
  {
    for int i := 1 step inc (i) while i <= 12
    Out.vdu.num_fld (i, 2).str ("! = ").num
(factorial (i)).nl ();
  }
}
```

This code illustrates the use of recursion to calculate each value from the previous value via multiplication while utilising the axiom 1! = 1.

Finally as an exercise, try changing the program so that the first 20 factorials can be calculated without running into the 32-bit limitation on integer size (Hint: return a real from a factorial and use .float instead of .num to output it).

Next Time

Next time I intend to cover Charm data types, variables, strings and scoping.



Introducing streams

DIFFICULTY: MEDIUM

Alex Kerr
Guest Writer

We've covered some of the basics now, and you may be starting to notice some similarities between C++ and C. From this issue onwards, we will be covering what makes C++ and C different. For the basics like if statements and loops, have a read of The C Cave.

Compatibility with C

Several of the standard C library header files are accessible from C++. To use them, include them as you would any other header, but add a 'c' to the front of the name and take away the '.h'. For example, the C code:

```
#include <stdlib.h>
#include <stdio.h>
```

Becomes the following in C++:

```
#include <cstdlib>
#include <cstdio>
```

This will give you access to all the functions inside these libraries, such as printf(), and rand(), which allows C++ to borrow a lot of useful features from C.

I/O Streams

Not everything is the same though, as you will have noticed. For example, when we were outputting, we used cout instead of printf(). This is because C++ uses I/O (short for Input/Output) streams.

What this really means is that the things that control input and output are just fancy objects, instead of functions. Objects and classes will be covered in depth later on, as that is the fundamental difference between C and C++.

There are three main I/O stream headers. These are <iostream>, which controls output and input to and from the console, <fstream> which is used for files, and <sstream>, which is used for strings.

We already know how <iostream> works, so let us look at <fstream>.

Reading and Writing Files

Continued Over Page...

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

int main()
{
  // Create an input file stream named 'reader':
  ifstream reader("test.txt");
  while(reader.good())
  {
    string temp;
    getline(reader, temp);
    cout << temp << endl;
  }
  reader.close();
  return 0;
}
```

That code allows you to read in a file named 'test.txt' (in the same directory as the executable) and output its contents. You could use the '>>' symbol, as we do with cin, but you will find it only reads up to a space.

You get this issue with cin as well, so if you're trying to get something like a name which needs spaces, you could use:

```
getline(cin, variable);
```

And that will read the whole input and save it to 'variable', which needs to be a string.

For outputting, we can use it just like cout:

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

int main()
{
  // Creates an output file stream named 'writer':
  ofstream writer("test.txt");

  // Write some text to the file:
  writer << "Hello" << endl;
  writer << "This is a test file" << endl;
  writer.close();
  return 0;
}
```

This program creates a file named 'test.txt' – in the same directory as the executable – with the text you see. However, this will overwrite any files already there, so be careful!

To avoid this happening, and to do some other interesting things, there are various options we can use when making our file stream objects:

Option	Description	Notes
ios::app	Appends to an existing file, instead of overwriting it.	
ios::in	Gets input from a file.	Default when used with ifstream(filename)
ios::out	Outputs to a file	Default when used with ofstream(filename)
ios::binary	Reads the file in as binary, instead of text.	

These can be added as options when you first make the stream. For example:

```
// Makes a stream that can input and output.
fstream(filename, ios::in, ios::out);

// Makes a stream that appends to a file:
ofstream(filename, ios::app);
```

When a stream is created is it associated with a memory buffer. When information is written into the stream or read from the stream, information is read from or written to the memory. This means that if an output file is used, but the file is not closed some of the data may not be written to the output file. When a file is closed, the any information in the associated buffer is flushed to the file. While the fstream close function does flush the stream, some streams may require explicit flushing.

C++ streams are said to be "type safe", since they can be used to read an input value into a type without the need to necessarily use that type. In C, this is not the case and sscanf functions require projection.

Next time we will look at strings and <sstream>, allowing you to convert strings to different data types. Hopefully you can begin to see how all the streams are similar, but suited to their job. Try inputting and outputting from files, and see what interesting things you could do.

THE SCRATCH PATCH

Scratch Controlling GPIO 2 - Birthday Pi

This month's article is a continuation of the previous Scratch GPIO article in Issue 9. Here we will describe further GPIO functionality for Scratch, including use with the popular LEDborg RGB LED module.

Since March is the month of the Raspbery Pi's 1st Birthday, we thought that this would be the perfect opportunity to play with some cake, LEDs and programming. What better way to celebrate this awesome little computer?

To follow this article fully, you will need an LEDborg module from the PiBorg site at http://piborg.com/ledborg and also a cheap LED candle which we found on eBay. Total cost for both is ~£7. Alternatively you can just use a single LED as described in last month's Scratch GPIO article. Lastly, you also need to have completed last month's Scratch GPIO setup for this to work.

Last month we explored how to connect an LED to the GPIO using a breadboard and a resistor with the help of Simon Walters' Scratch GPIO program (http://wp.me/p2C0q1-27). This month we will explore some more sophisticated ways to control the GPIO with Scratch including adding LEDborg support. To start with you will need to download the updated Scratch GPIO handler by typing the following into an LX Terminal window:

```
$ sudo wget http://themagpi.com/files/issue10/scratch_gpio_handler.py
$ sudo mv scratch_gpio_handler.py ~/simplesi_scratch_handler
```

You will need to be connected to the internet for that to work. Once the above has completed, you are all set.

Advanced GPIO with Scratch

You will now be able to control seven GPIO pins as outputs (pins 11, 12, 13, 15, 16, 18 and 21) and treat another seven as simple inputs (pins 7, 8, 10, 19, 22, 24 and 26). You may have noticed from last month's blink11 script that you can simply use broadcast messages to turn pins on or off.

Broadcast messages are found in the "control" block of Scratch. With broadcast messages you can use commands such as **allon** or **alloff** to turn all the pins on or off at once. Alternatively you can change the state of single pins using commands like **pin11on** or **pin11off**, replacing the number with the pin number you are trying to alter. You can also combine messages together like so:

```
broadcast alloff pin11on pin13on
```

Birthday Pi

You will now need to have your LEDborg module plugged in (do not plug in when the Raspberry Pi is on, turn the Raspberry Pi off first).

For this next part, we will be using a more advanced method called variables. To set up a variable, you need to go to the "variables" block and click on Make a Variable. You will then need to name your variable according to what you want to do. As before you can create variables for certain pins, such as pin11 and you can also create a variable called allpins to change them all at once. For this example we will be using three variables called ledborgr, ledborgg and ledborgb - these can be seen in the dark orange colour variable boxes to the left.

Once you have created the variables, you can then change the brightness of each of the red, green and blue channels of the LEDborg by altering the value of each of the corresponding variables.

The script shown to the left starts by setting all the variable values to 0. It then goes on to ramp up the blue part of the LEDborg until it is at its full brightness (value 100). It then waits for half a second and ramps the brightness down to 0 again and then waits another half second before it is then repeated for the green and red channels. Eventually it starts again at the beginning. This will repeat itself until the red stop button is pressed.

The front cover of this issue shows the Pi and LEDborg in action! In the next Scratch GPIO edition, we will discuss some simple inputs to complement the outputs discussed so far.

Article by Aaron Shaw

THE SCRATCH PATCH

The Julia Set

Over the next two articles, we are going to use Scratch to draw some fractal patterns.

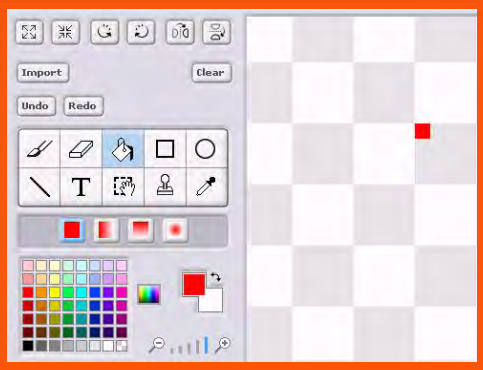
Gaston Julia, the French mathematician, did the early work on fractals which is why the patterns we will be creating are known as "Julia Sets". He was badly injured in World War One, which is why he wore a patch over the centre of his face.

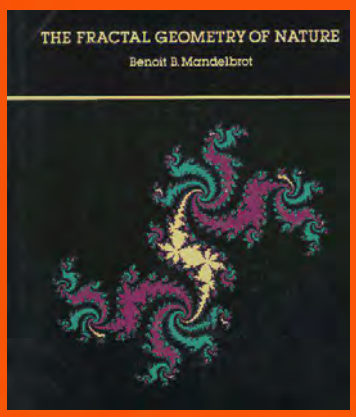
The program to draw these patterns is actually quite simple! Let's start by deleting the cat sprite and making a new sprite with two tiny costumes.

Gaston Julia (1893 - 1978)

Click on the paint new sprite button and then zoom in as far as possible. Erase what's there already and draw the smallest possible square (with the rectangle tool) and fill it in with red.

Call this costume "colour" and then make another one the same, except black, and call that "black".

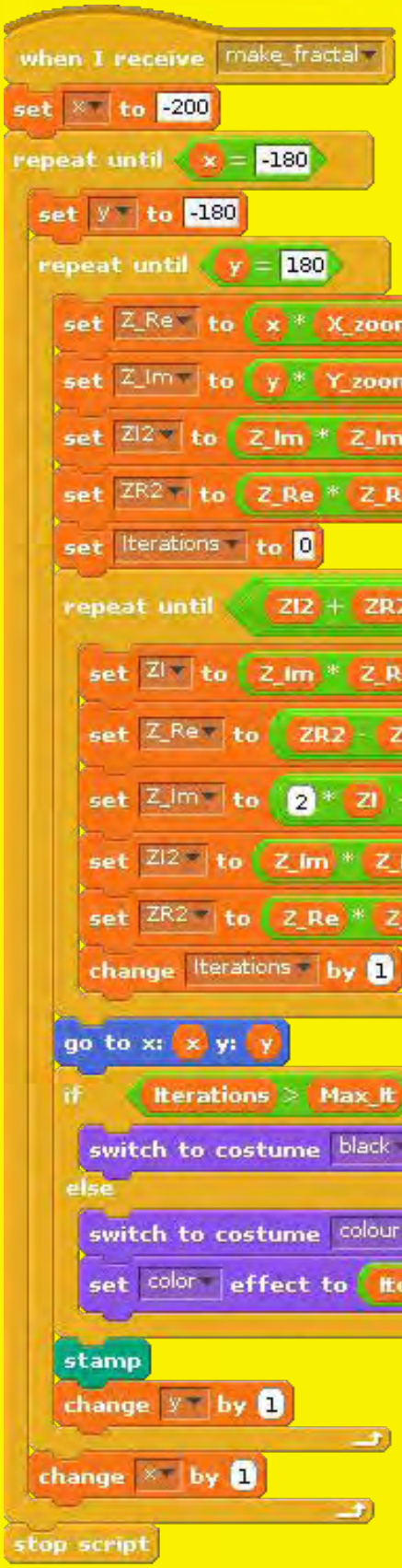




Mandelbrot continues Julia's work on Fractals

In the 1970s, the Polish mathematician, Benoit B. Mandelbrot, began to use computers to create images based on fractal mathematics. His book, "The Fractal Geometry of Nature", was very popular and introduced fractals to a much wider audience.

The Main Drawing Script



This is the main script that draws the fractal image. It's crucial to create the variables correctly.

These variables MUST be **"For this sprite only":** iterations, ZI, ZI2, ZR, ZR2, Z_Im, Z_Re, x, y.

These must be **"For all sprites"**: C_Im, C_Real, Col_offset, Max_It, X_zoom, Y_zoom.

In the first few lines of the script, we set up which area of the screen this sprite will draw on.

This one does a strip 20 pixels wide: from x = -200 to x = -180.

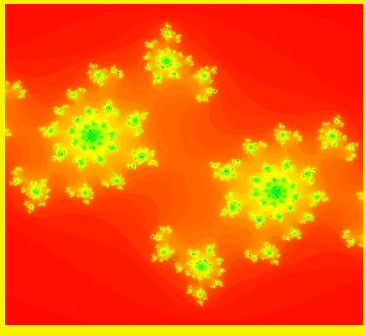
When you have made this script, you need to "duplicate" it 20 times.

Then change the values for x so that each sprite does another 20 pixels.

The last sprite should be in charge of x = 180 to x = 200.

This allows the program to draw to several parts of the screen at the same time, which will speed things up a lot. I've gone with 20 "strips", you could try experimenting with other arrangements.

Here's an example of the type of images you will be able to create.



c = -0.7467 + 0.3515*i*

Scratch On!

i - the Imaginary Number

The imaginary number, ***i***, is the square root of -1.

The difficulty with ***i*** is that it is impossible to find a "normal" number that, when multiplied by itself, becomes -1**.**

Descartes did not like ***i*** and he gave it the name "imaginary number" as a kind of insult. However, after the work of Euler and Gauss, ***i*** quickly became an accepted part of mathematics.

Complex Numbers

When we work with ***i***, we often use it as part of a "complex number". A complex number has a real part and an imaginary part.

$$C = a + bi$$

In this equation, C is a complex number and it is made by adding the real part, a, to the imaginary part, b, multiplied by ***i*.**

The mathematics of the Julia set requires us to use complex numbers. When we make our fractal images, you should picture the x axis of the screen as representing the real part of the complex number (a) and the y axis as showing us the value of the imaginary part (b).

Generating a Julia Set

As you will see on the next page, we give the program the values for the real and imaginary parts of C. Then the sprites visit every "pixel" in the region of the screen we are drawing on, treating it as a graph of the complex plane (this is what we call the visualisation of complex numbers that I explained above).

The current values for x and y are taken as the initial real and imaginary parts of Z (Z_Real and Z_Im), Then we iterate over a function:

$$z_1 = z^2 + c$$

If the value of the sum of the squares of the real and imaginary parts of z is more than 4, we consider that this location has "escaped" (if we carried on iterating, the number would get bigger and bigger) and we use the number of iterations completed so far to set the colour of the pixel.

If the values stay lower than 4 within the maximum iterations we have chosen, the location is considered to be part of the Julia set and the pixel is coloured black.

The wikipedia article on the Julia set is useful, if this (rather skimpy) explanation has not made the maths clear enough.

The Green Flag Script

This script is run when the green flag is clicked. You can give it to any of the sprites you've made (I'd suggest giving it to the first.)

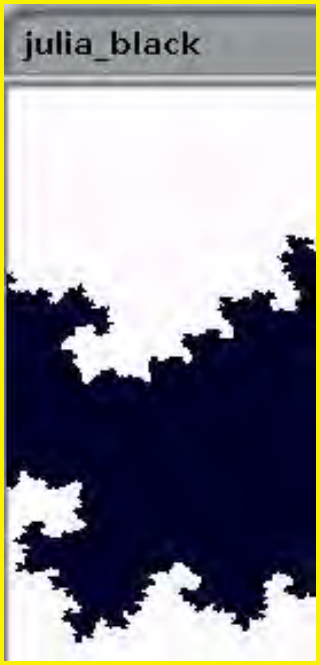
The most important thing is to enter the values or the real and imaginary part of "C".

The colour image lower down on this page is known as "Douady's Rabbit" and has the following values for C:

Real: -0.123
Imaginary: 0.746

Next, you can specify the maximum number of iterations. I'd suggest you try about 20 at first. If you choose more, you'll get a smoother range of shades in the picture, but it will take longer to draw.

Setting the "zoom" allows you to focus in on a smaller area. A zoom of 1 is "normal size". You could see what happens when you enter 2 or 3 instead.



A monochrome image from an earlier version of the program.

Complex Numbers to try

If you search on-line for Julia Set fractals, you will find plenty of values for C to use in this program. Here are a few that I have used while testing this program:

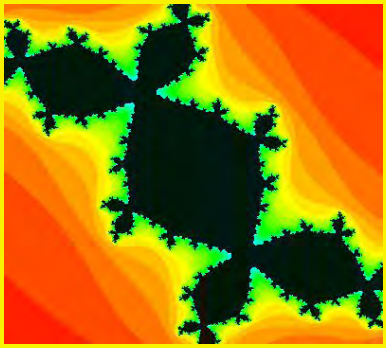
Real:	-0.4	Real:	-0.8
Imaginary:	0.6	Imaginary:	0.156

Real:	-0.1
Imaginary:	0.651

Real:	-0.7467
Imaginary:	0.3515

Download

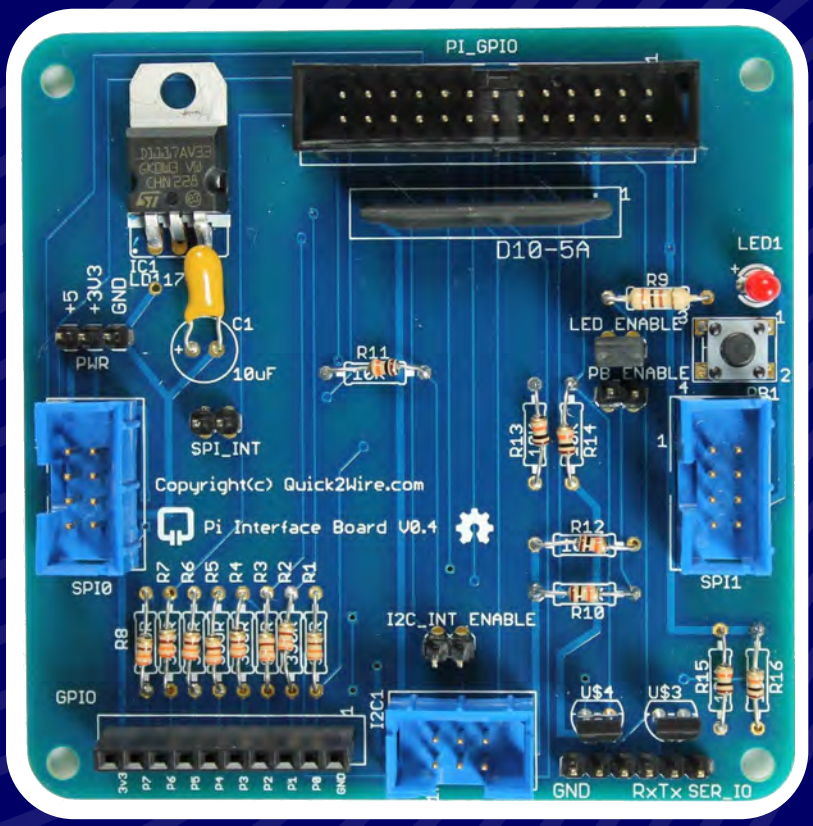
If you get stuck, you can get the code here:
http://tinyurl.com/gastonjulia/



Douady's Rabbit

Quick2Wire

SAFE AND SIMPLE CONNECTION TO YOUR RASPBERRY PI



HARDWARE

- Interface board
- I²C Port Extender
- Analogue board

SOFTWARE

- For GPIO, I²C, SPI
- Device Libraries
- Examples

SUPPORT

- Articles
- Tutorials
- Forum

Interface board: £13.86 | Port Extender: £9.80 | Combo: £22.66 (save £1.00)

Prices include UK VAT but exclude Postage and Packaging: from £2.70

Find out more at quick2wire.com

the python pit™

Parallel calculations part 1

DIFFICULTY : ADVANCED

W. H. Bell

TheMagPi

Previous Python examples have discussed web server access to programs. However, sometimes other client server relationships can be useful. This month's article is the first part of a demonstration of a basic client server application, which shows how other applications might be written.

The Raspberry Pi can be deployed with solar panels or batteries and connected to a network using a Wifi dongle. In this manner, it can be used for remote monitoring or robotics. Alternatively, the Raspberry Pi could be used as the control centre of a computational system. In the first example, the Raspberry Pi could be a client or a server. In the section example, the Raspberry Pi is likely to be a server.

Servers are processes which listen for clients to connect to them. When a server receives a request from a client, it is common for a thread to be allocated to the client connection. A thread is the smallest sequence of programmed instructions which can be managed independently by an operating system. Often the server listening process runs as one thread and gives the clients each a thread from a limited pool. When the client is finished, the thread in the server should be released for a new connection. If the server created a new thread for each client connection it would quickly run out of memory.

Sometimes calculations require more than one computer to find a result quickly. When a physics or engineering problem is described by an equation with many variables, finding a global minimum for the equation can be impossible on paper and take too long with one computer. To solve this problem, a network of computers can be connected together to calculate many points and numerically solve the equation much more quickly. While the Raspberry Pi does not have the fastest CPU, it can be used to demonstrate this principle. For the client-server parts of this problem another Raspberry Pi or another computer will be needed. If you have many other computers to play with or can invite many friends around, all the better.

Classes and function evaluation

Create a file called `FunctionCalculator.py` and add to it,

```
# A class to calculate the value of a function string
class FunctionCalculator:
  def evaluate(self, cmd):
    y = 0.
    print "exec \"%s\"" % cmd
    exec cmd
    print "y = %e" % y
    return y

# A class to calculate many the result of many equations at once.
class SynchronousCalculator:
```

```
  def __init__(self):
    self.calculator =  FunctionCalculator()

  def evaluate(self, cmds):
    results=[]
    for cmd in cmds:
      results.append(self.calculator.evaluate(cmd))
    return results
```

The FunctionCalculator is a simple class which has one member function that executes the value of a string as a python command. The SynchronousCalculator includes an instance of the FunctionCalculator class to evaluate many commands one after another. In the case of the FunctionCalculator, the evaluate method takes one string whereas the SynchronousCalculator function takes a list of strings.

To test FunctionCalculator.py open a python shell by typing python. Then type:

```
from FunctionCalculator import FunctionCalculator
f = FunctionCalculator()
f.evaluate("y=5*100")
```

This evaluate function prints the command and the result of evaluating the command. Many points of a mathematical function can be evaluated using the SynchronousCalculator. For example, ten points on the curve y=x^2 can be calculated by typing,

```
from FunctionCalculator import *
import math
f_sync = SynchronousCalculator()
cmds = []
for x in xrange(11):
  cmds.append("import math; y = math.pow(%f,2)" % (x))

f_sync.evaluate(cmds)
```

Notice that the FunctionCalculator class, which is used by the SynchronousCalculator class, does not need to import the math functions until they are needed during the command evaluation. This becomes very powerful when used with many computers, where the function to be evaluated remotely might not be known before runtime.

The next step needed to improve the speed of calculations is to use more than one computer. To do this, one Raspberry Pi will be needed as a server and another Raspberry Pi or other computer will be needed as a client. Connect both Raspberry Pis or the Raspberry Pi and other computer to the network. Then find the IP addresses for the two machines. On LINUX or OSX type ifconfig, or on Windows type ipconfig.

To avoid any difficulties with network address translation (NAT), make sure that the two computers are on the same network. Then test the network path using ping from one machine to the other. For example,

```
ping -c 5 192.168.1.12
```

If this is successful, it will return the time the ping took five times. Use ping from both machines to be absolutely sure the network path is as hoped. Now create a SimpleServer.py file containing

```
import socket
import threading

class SimpleServer:
  def __init__(self,host,port):
    self.host = host
    self.port = port
    self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.sock.settimeout(None)
    self.client_sockets = []
```

```
  def initialise(self):
    try:
      self.sock.bind((self.host, self.port))
    except socket.error:
      return

    self.sock.listen(5)
    self.server_thread = threading.Thread(target=self.serve_forever)
    self.server_thread.setDaemon(True)
    self.server_thread.start()
    print "Server running on %s and listening on %d" % (self.host, self.port)

  def serve_forever(self):
    try:
      request, client_address = self.sock.accept()
    except socket.error:
      return

    self.client_sockets.append(request)
    print "Received connection from ", client_address
```

To start up the simple server, open a python shell and type:

```
from SimpleServer import SimpleServer
import socket
server = SimpleServer("192.168.1.3", 20000)
server.initialise()
```

Then go to the other Raspberry Pi and open a python shell and type:

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(("192.168.1.3", 20000)) # The IP address of the server machine
sock.close()
```

Now look on the server machine. The server machine reports that the client machine has connected to it.

In the example program, the server address is explicitly used in the server startup, since the host name of a Raspberry Pi using DHCP resolves to the local address `127.0.0.1`. If the server's host name is associated with an address on the local network, then the command `socket.getfqdn()` can be used instead of `"192.168.1.3"`.

The `SimpleServer` class contains several member variables which are initialised in the constructor `__init__`. These member variables contain the values for the host name, port number, socket and client connections. The port number has to be high enough to be accessible to a non-root user. Each server binds to a given port number. Once the port number is in use, it cannot be used by another server. Therefore, the server has to be shutdown before it can re-bind to the same port. The instantiation `socket.socket(socket.AF_INET, socket.SOCK_STREAM)` creates a socket. The `AF_INET` refers to the address family, which in this case is an Internet Protocol address. There other types of socket for bluetooth. The `SOCK_STREAM` is the socket type, which is a reliable two-way connection-based byte stream. The available address family names and the socket types are also found within standard LINUX C libraries. The `client_sockets` variable is a list for holding each client connection.

The `SimpleServer` class contains two other member functions `initialise` and `serve_forever`. The initialise function tries to bind to the allocated socket. If this is successful, it configures the socket as a listening socket. Then a thread is created which executes the `server_forever` member function. The thread is configured as a daemon and started. The `serve_forever` member function is the listening process which accepts connections from clients. The client socket associated with a connection is then added to the list of client sockets.

The next article will show how to send and receive commands from clients, using threads for each associated connection.

Tested with Python 2.7.3 (Raspbian) & 2.6.1 (OSX 10.6.8)

Feedback & Question Time

I am new to RPi. It is great. Thanks for your mag. I download it and then transfer to my Kindle where it is easy to read.

- Dr David C Cusworth

Very well done for everything Raspberry Pi that you are doing. I am finding it very interesting and informative.

- Terry Hardy.

Thanks you for the great project regarding the pi magazine. When I was a child my father used to buy these kind of magazines and I miss them, not only for me, but for the new children who think the only OS in existence is that thing called windows, and when they look at linux just don't know what to do, even when they are computer related students. Don't ask when they are confronted with the command line.

- AlvaroMendoza

I love you!

- Petr Masopust

Im a big fan of the raspberry and also appreciate all the work you put in the MagPi magazine. Its a pleasure to read the articles and feel the energy and love your team of volunteers put into that. Thank you very much

- Dominik
from Munich, Germany

I have read your current issues of the mag and am delighted with it. It has been a while since this sort of publication was available, and I have been wanting to get into programming for years, even up to the point of attending college. But with this I can study at my own pace and start my own projects. I dont have a pie as of yet, but intend to buy the starter pack from Maplin as soon as I can. Again, thanks.

- Ryan Hunter

I was wondering if I could write for The Mag Pi magazine, I love the magazine, and I of course love technology and writing. I was also one of the first Raspberry Pi users. Thank you.

- Arman Bhalla

I just wanted you to know that I love your magazine. It is well laid out and highly informative. It is the perfect introduction for students, and also hobbyists to the wonderful world of Raspberry Pi. Believe me when I say, you are helping create the next generation of Engineers, makers, hackers, and artists. Keep up the great work.

- David Sanders