

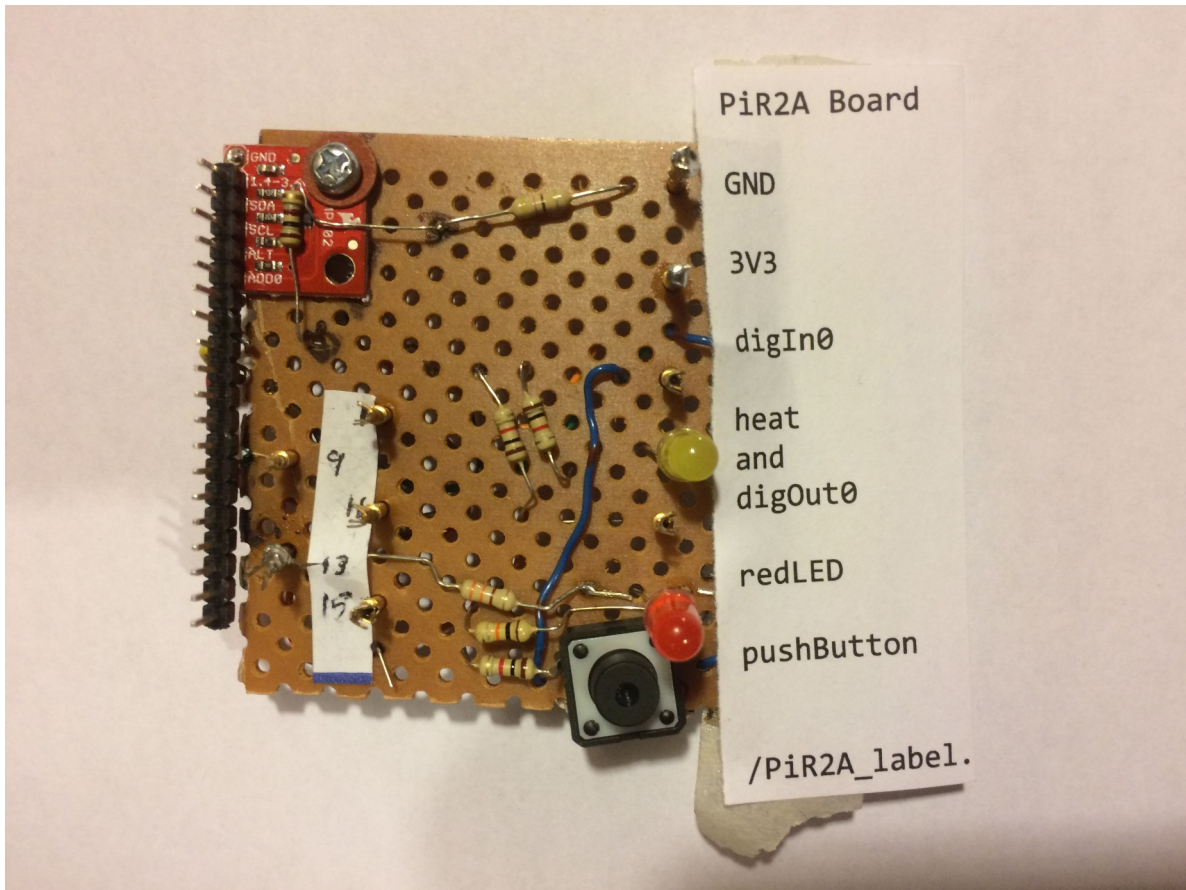
PiR2 Data Log System Definition V15

by David@ColeCanada.com

2022 B Feb 28

Table of Contents

1. Purpose
2. Introduction
3. Components
4. Record Definitions
5. Procedures/Methods
6. Related Documents
7. Sources



1. Overall Purpose

The purpose of the PiR2 system is to allow the owner of the PiR2 system to monitor and control each area of his home or business. This is done by installing a Raspberry Pi controller in each area. The owner can use his iPhone (or iPad, tablet etc) to oversee the monitoring of each area (for example, the kitchen temperature). The owner can also use his iPhone (or iPad, tablet etc) to control devices in each area (for example take a picture or turn on a lamp). A long-term log (i.e. a record) can be kept of the activity in each area of the house.

1.1 The More Detailed Purpose

The detailed purpose of the PiR2 data log is to define the data that moves within the PiR2 system. Most of the data manipulated by the PiR2 system is data that is acquired by the various Data Acquisition/Control (DA/C) io devices attached to each Raspberry Pi computer. Even without any additional controller electronics, the Pi computer itself can be used as a source of data. The internal temperature of the Pi processor can be easily read. So can the Serial Number and at least one voltage within the Pi processor. These three io devices are preprogrammed in the PiR2 system. They can be used to test the functionality of the PiR2 system software. The internal temperature of the Pi processor is of considerable interest because it tends to slightly fluctuate.

A Meh-In-Charge software license is required in order to store and access the acquired data measurements on the PiR2 web site. This license can be downloaded from the Meh-In-Charge website and is free for the next month of use. Upon expiry, for continued access, the license must be purchased.

Other external PiR2 io devices (e.g. sensors) are available for purchase. Or they can be built by the user. An owner of a Meh-In-Charge software license can access the data that is acquired by the sensors on his/her Raspberry Pi. Graphs of the data measurements can be prepared, analysed and/or printed anywhere by using any communications device such as an iPad, cell phone or computer. Of course the communications device must be attached to the internet. The owner can also control the Data Acquisition/Control (DA/C) devices on his/her Raspberry Pi remotely with such communication devices. What Pi devices can be controlled by the owner? A long list of such devices appears later in this document in Section 3.2.1. Fortunately most owners of a Raspberry Pi connect it to a television set via HDMI. With a Meh-In-Charge license, a remote user can cause recordings to be played on the speakers of the television set used by his/her Raspberry Pi. The recording can be any mp3 tune, any audio recording or even a phrase (wav) spoken by the owner.

In future, such a person can dictate a phrase into his iPhone and cause the phrase to be “spoken” by the television set connected to his Raspberry Pi. In future, the Meh-In-Charge software will permit the operation of devices via Blue-tooth. Such devices must be located in the same room as the PiR2 computer. This is possible because Blue-tooth capability is already built into each Raspberry Pi.

A single PiR2 controller can be used to monitor any room. A person, far from the room, can discover what is happening in the room. The PiR2 controller in the room senses what is happening in the room. The data sensed by this controller is continually uploaded to the Meh-In-Charge website. At any time, the Meh-In-Charge user can see what is going on in the room. For example, the temperature of the room can be viewed in a graph on the user's iPad, even if the user is in a remote location.

One of the main ioDevice types supported by the PiR2 system is a USB recharging station. Each recharging station can be used to charge one of the following:

- iPod
- iPad
- iPhone
- cell phone
- tablet
- etc.

The PiR2 system is designed to do an enhanced job of recharging these devices. By measuring the current being furnished by each USB recharging station, the PiR2 system can know very much about the USB recharging situation such as:

- whether the recharging station is empty. (If so the user can be notified at bedtime)
- if the device being recharged is full or empty
- estimate how much time it will take to recharge the device
- etc.

In this way the user can control many devices by using the Meh-In-Charge system. The Meh-In-Charge name means that the user is in charge (of the recharging of his communication devices) as well as being in charge of the ioDevices connected to each PiR2 controller. Note that the formula “PiR squared” calculates the area of a circle. Each PiR2 device controls the area in which it is situated.

1.2 Previous Related Projects

The author has experience creating 2 related projects:

iGalri

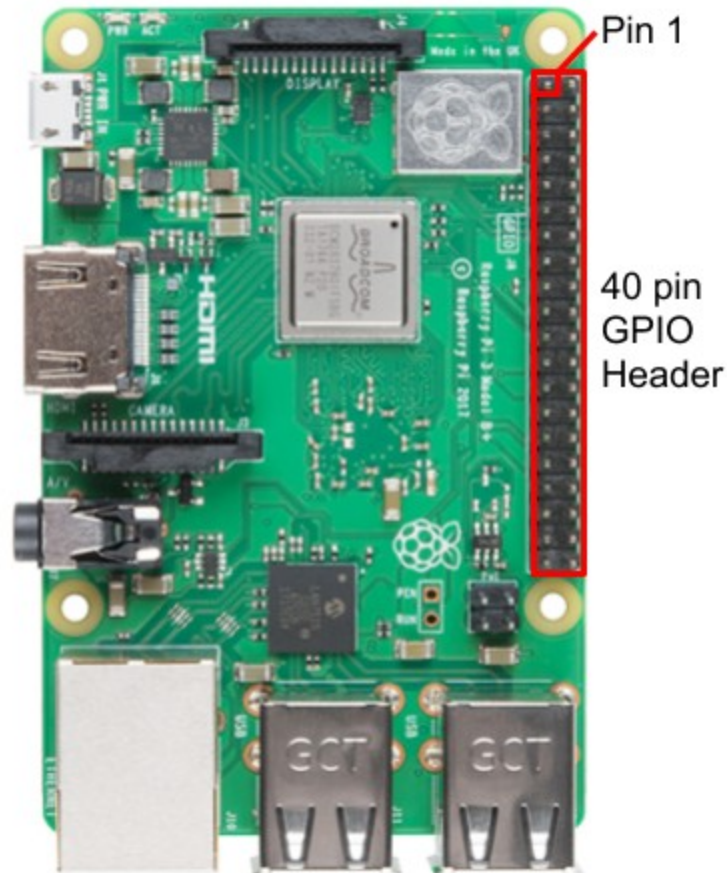
The iGalri project permits a user to upload a photo and its description to the iGalri.com website. All descriptions are stored in a database. The user can then choose any phrase that might appear as a subphrase in any of the descriptions. A suitable database SQL query is then done to search for the phrase. Each photo containing that phrase in its description will then be displayed (along with the full description of each photo).

Silver HOA

The Silver HOA project is a website for an HOA or condominium association (e.g. BurgundyUnitOne.com). Photos are displayed in various areas within the website. A small routine was created to permit authorised persons to upload captioned photos to the website. This website has been replaced by Burgundy1.org .

2. Introduction

The tiny \$35 Raspberry Pi computer Model 3 is shown below:



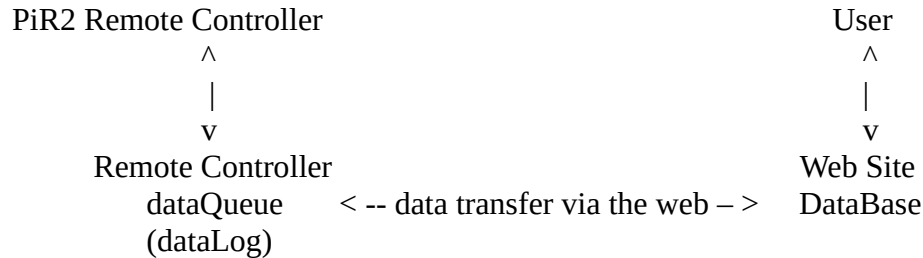
This computer is described in more detail in Source 8. It requires a power supply, HDMI monitor, keyboard and mouse along with the Raspbian Operating System and an Ethernet connection to the Internet. Setting them up is not described in this document.

The Raspberry Pi computer is the main part of the PiR2 Controller. The formats of the data will now be defined:

The PiR2 data log system is comprised of 3 sections:

- a) the remote controller data queue and log
- b) the data being transferred across the web
- c) the DataBase housed at the PiR2 website

The data is accessed as shown in the following diagram:



The data records in the three sections highly resemble each other. The records in the remote controller are stored in an ascii queue in a sequential file format. The Python driver for each input/output (io) device can easily add a single record to an ascii queue (which is stored in a sequential ascii text file). The queue manager in each remote controller can easily work with a sequential file. Objects (such as photos in jpg format) can be in a second queue (an object queue) that is managed by the same software routine as the ascii queue. Note that the object queue transfers will be out of sync with the ascii queue.

The different data formats are chosen for simplicity in the remote controller's dataQueue (i.e. dataLog) where they will be appended using Python routines. Each ioDevice will have a matching Python routine that will add (or read) ascii records to/from the ascii queue. The more complex io devices will have more complex Python routines that also add/retrieve objects (e.g. a jpg photo or an audio wav) to the object queue. All objects will be stored in a common folder in the remote controller.

The data and objects are transferred bidirectionally between the remote controller and the DataBase on the website using HTML/PHP routines located on the website. These HTML/PHP routines are invoked using Mathematica routines on the remote controller. The Mathematica routines manage the queues and the HTML/PHP routines without human intervention. The HTML/PHP routines can transfer ascii records and objects in both directions across the web. The Mathematica routines can easily manage the ascii queues and the object queues. The data format of the data being transferred over the internet is designed to minimize the number of bytes being transferred so as to minimize data transfer times.

The data format in the DataBase at the website is designed to be accessed using SQL queries. The user accessing the data in the DataBase will solely use prewritten HTML scripts that invoke PHP routines that make use of SQL queries. The sole use of HTML scripts to access data in the DataBase means that the user can access the data from his/her remote ioDevices using any web-conversant device such as a computer, an iPad, an iPhone, a cellphone or a tablet.

2.1 The Ascii Queue

The records in the ascii queue can be of 4 formats: Basic, Normal, Max-Data and Objects. These formats are of increasing complexity. The 3 least complex formats do not include data objects. In the ascii queue, a typical data record in Basic format will include the following information:

Field Name	E.g.	Description
record type	D	indicates a Deleted data record
format code	B	code indicating the Basic data format
qid	000000010045	unchangeable io record identifier for each data record from this io device
dateTimePosted	0123431234	timestamp (number of seconds since the UNIX epoch which is Jan 01, 1970). This field is used for sorting the file when being accessed.
IoDevice	procTemp	a sensor or control device
timePosted	2019EMay13-17:19	DateAndTimePosted (16 bytes) NB not used for sorting the timePosted shows at a glance the age of the record
source	A	A : Acquired data (to be sent to the DataBase) C : Control data (to be sent to the remote controller)
st0	Q	initial state (does not change)
st	S	dynamic state in the queue (normally Q>S>A>L) Q added to the queue S sent from the queue A acknowledged receipt (transfer is complete) L has been logged to an external log file individual bad data records can be altered by the user D flagged for deletion M modified C data received from the database is Control Data which is in state C?? C states also become A states after being used???
ioDevice value	PiProcTemp 256^4	such altered records are treated like state Q The temperature of the Pi Processor 4 bytes of data x00000000 to xFFFFFFF this data is usually 1 bit (0 or 1) 1 byte (x00 to xFF) 4 bytes (0 to 4,294,967,295 which PiR2 considers to be -2 billion to +2 billion) this is a whole integer (not a real number). If it always needs to be multiplied by a factor, the factor is stored in the "multiplier" field of the header.

The length of a Basic PiR2 record is 62 bytes.

A header record is needed to more completely identify this data record. Its fields are:

Field Name	length (bytes)	E.g.	Description
Format	1	H	code identifying a header record
qid	4	0000000019991	unique record identifier
datiPosted	4		
timePosted	16		
PiSN	19	00000000a1b2c3d4Pi_	Serial Number of the Pi computer
ioDevice	15	procTemp	ioDevice name
ioType	8	temp	type of data measured
A/C	1	A	Acquisition / Control
room	16	kitchen	room name
ioDeviceName	32	Ambient Kitchen Temp	user's name for the ioDevice
units	32	degC,	measurement is in these units
DataFormat	1	B	data records format
ObjPath	64	C:/Pi/User/photos/PiR2	path housing objects on this Pi
unused	42?		future use
Total	256 bytes		

2.2 The PiR2 Log File

Each data queue is in a constant state of flux. The status fields in the initial (older) records eventually become “A”. The status of “A” means that the record has been sent to the DataBase and an acknowledgement has been received. The internet does a very good job of not “losing” records, but this system does not presume that all internet transfers are perfect. That is why the transfer of each record must be acknowledged. When the first 1000 records in the data queue all have a status of “A” it means that they are redundant and will be automatically logged to an external log file, which is usually a flash drive. After each data record has been moved to the external log file, its status is set to “L” which means “Logged”. After each external log file has been fully written and is closed, all the records with a status of “L” will be deleted from the data queue. This prevents each queue from filling the original (working) drive where it resides. All other activity in the data queue is blocked while the “L” records are being removed. Records in the data queue must always be accessed/updated by their “qid” identifier, not by their sequential position in the data queue. However, new records can always be appended to the sequential file (ie the data queue) without referring to their “qid” identifier.

2.3 The PiR2 Record Set in the Log File

Each PiR2 log file is comprised of the following record set:

- H header record for records of each ioDevice
- C comment record (can be blank)
- D data record for each ioDevice
- additional such records
- E End of File record (may be absent)

3. Components

Each of the following components of the PiR2 system will now be described:

3.1 ascii data queue

3.2 object queue

3.3 raspberry Pi computer

3.4 routines: Python, Mathematica, HTML, PHP, SQL queries

3.1 io devices

Those highlighted in **Yellow** are fully described in this document.

3.5.1 List of ioDevices in the PiR2 Controller

PiR2 io Devices

DA/C	bit#	ioDevice	Description
A	0	pb01	Push Button # 1 (1:low:depressed)
A	1	IRin01	Infra-red detector (1:low:detected)
A	2	AC00amps	AC00 (110v mains) amps (1:low:amps are flowing)
A	3	ExtIn01	External Input Bit 01 (1:low:detected)
A	4	ExtIn02	External Input Bit 02 (1:low:detected)
A	5	USBamps01	USB charger amps 01 (1:low:amps are flowing)
A	6	USBamps02	USB charger amps 02 (1:low:amps are flowing)
A	7		unassigned
C	0	Led01	Inhibit LED (Red) on
C	1	IRout01	Infra-red source on
C	2	ACout01	AC output receptacle 01 enable
C	3	ExtOut01	External Output Bit 01 (low)
C	4	ExtOut02	External Output Bit 02 (low)
C	5	Led02	Signal LED (White) on
C	6	USBout01	USB01 charger enable
C	7	USBout02	USB02 charger enable
A	0	Temp00AnlgIn	Temp Sensed at location 00 (99.9)
A	1	USB01AnlgOut	USB01 Output Amps (16 bits analog)
A	2	USB02AnlgOut	USB02 Output Amps (16 bits analog)
A	3	AC01AnlgOut	AC01 Output Amps (16 bits analog)

ipAddr

Built-In ioDevices (in every Raspberry Pi)

DA/C bit#	ioDevice	Description	SW Name
A	piSN	readable pi Serial Number	?
A	PiProcTemp	Pi Processor temperature	?
A	PiVolt00	Pi voltage 00	core
A	PiVolt01	Pi voltage 01	sdram_c
A	PiVolt02	Pi voltage 02	sdram_i
A	PiVolt03	Pi voltage 03	sdram_p
A	AudioIn01	Pi Audio Input Jack (for microphone)	
C	AudioOut01	Pi Audio Output jack (for head phones)	

Internal non-hardware PiR2 ioDevices monitoring/controlling PiR2 activity

DA/C bit#	ioDevice	Description
A	PiR2dataQlen	Number of rows in the PiR2 dataQueue
A	PiR2objQlen	Number of rows in the PiR2 objectQueue
C	dtSampleInt	Time between PiR2 samples (default value) -1 no sampling (PiR2 is off) 1 (msec) between samples
	ethIP	Pi's ethernet IP address

External ioDevices (easily attached to a Raspberry Pi)

DA/C bit#	ioDevice	Description
C	HDMIAud01	HDMI TV audio speaker
A	PiPhoto01	photo taken by a Pi camera
A	PiVideo01	video taken by a Pi camera

3.2 ioType

There is a small number of different types of io:

ioType	Description
temp	Temperature degF,
amps	amps of electrical current
bit	lowVoltage = 1 = on
number	any integer from -2 billion to +2 billion
audio	any audio recording (mp3 or wav format)
voltsDC	electrical DC voltage
voltsAC	electrical AC voltage
text	any text string (never exceeds 256 bytes)
photo	any photo (jpg, tiff or png format)
video	any video

3.3 Meh-In-Charge Table Definitions

These tables are in the PiR2 DataBase at the Meh-In-Charge website.

To be added

4. Record Definitions

5. Procedures/Methods

5.1 Python routines

Each ioDevice on the Raspberry Pi's PiR2 controller will be controlled by a single Python routine (a function). The name of the function will include the ioDevice name. An example for the procTemp ioDevice follows:

```

fun io_procTemp(returns,DAC="A", ioDevice="PiProcTemp", simulate=globalSimulate,
initializeTo=None, deviceNameInMachine="sdram_p") :
    #Python 2 or 3 code
    if returns="DAC" result$=DAC
    if returns="ioDevice" result$=ioDevice
    if returns="deviceNameInMachine" result$= deviceNameInMachine
    if returns="simulate" result$=simulate
    if returns="initializeTo" result$=initializeTo
    if returns="dataValue" and "simulate"="False" result$ = Bash(vcgenCmd measure_temp)
    if returns="dataValue" and "simulate"="True" result$ = 57.0 +random()*2.0
    return result$
# end of definition of fun print_io_PiProcTemp()

```

#The following Python statement will display the (verbose) results

```

fun print_verbose_io_PiProcTemp() :
    #Python 3 code #this function will error out if run under Python 2
    if global_Python3="False" return "ERROR: This function (print_ioPiProcTemp) needs
    Python version 3"
    Print (io_PiProcTemp("iodevice"))           #device name
    Print (io_PiProcTemp("DAC"))                 #DAC A:Acquire, C:Control
    Print (io_PiProcTemp("deviceNameInMachine")) #if a software read is used
    Print (io_PiProcTemp("simulate"))            #is simulating being done?

```

```

Print (io_PiProcTemp("initializeTo"))           #initial value
Print (io_PiProcTemp("dataValue"))           #Acquire and print the dataValue
# end of definition of fun print_io_PiProcTemp()

#For verbose output, code:
# print_verbose_io_procTemp()                 # read & print (verbose) the temperature
                                              # of this Pi Processor !

#To truly read the PiProcTemp, code:
# print(io_PiProcTemp(dataValue))            # read & print the temperature of this Pi Processor !
                                              If globalSimulate is False

```

A similar Python routine must be created for each ioDevice.

5.1.0 Various Commands for PiR2

5.1.0.1 Common Unix Printing System (cups)

The following is necessary to access printers

```
sudo apt-get install cups
```

5.1.0.2 Remote Desktop Communication

To install RDP (aka RDC)

```
sudo apt-get install xrdp
```

The following turns on/off RDC

```
xrdp
xrdp -kill
```

The Rpi's IP address is needed when starting RDC. To find the IP address, hover over the WiFi symbol in the top right corner of the RPi monitor or run ">\$ sh Ipaddr.sh" or ">\$ipconfig".

The RPi password is necessary when starting RDC on a Windows—10 PC. The mnemonic for this password is LydieArthur\$. To change the Pi password use ">\$ sudo passwd pi".

See the Rpi environment with ">\$ sh Environ.sh"

To start up the PiR2 system:

```
>$ sh pir2.sh
```

In 2022, a comprehensive shell command has been developed to display the complete environment on the Raspberry Pi. It is invoked by the command below:

5.1.0.3 Environ.sh command

```
>$ sh Environ.sh
```

5.1.0.4 Other Shell Commands

Other shell commands are:

```
>$ sh model.sh
>$ sh serial.sh
>$ sh Alarm.sh
>$ sh etc
```

5.1.1 Compiling and Running Python routines

The process to run a Python 3 routine on my Raspberry Pi is:

```
cd Python
```

Double-Click on the python source file eg Prog02C.py (see Document 1)

sh pir2.sh will run the pir2main.py program from the command prompt.

This system needs python version 3 to run. Some code won't run under python 2.

No known program exists to compile python programs.

5.1.2 Python routines for Built-In ioDevices (in every Raspberry Pi)

DA/C bit#	ioDevice	Description
A	piSN	readable pi Serial Number

The following routines provide the pi Serial Number from a reliable memory location. But a more complicated routine is needed to read the Serial Number directly from the Pi computer.

Routine a): >\$ echo "\$Serial"

Routine b): >\$ grep Serial /proc/cpuinfo

Routine c): >\$ grep Model proc/cpuinfo

A	procTemp	Pi Processor temperature
	Routine:	vcgenclmd measure_temp

A	PiVolt01	Pi voltage 00
	Routine:	vcgenclmd measure_volts core

A	PiVolt01	Pi voltage 01
	Routine:	vcgenclmd measure_volts sdram_c

A	PiVolt02	Pi voltage 02
	Routine:	vcgenclmd measure_volts sdram_i

A	PiVolt03	Pi voltage 03
	Routine:	vcgenclmd measure_volts sdram_p

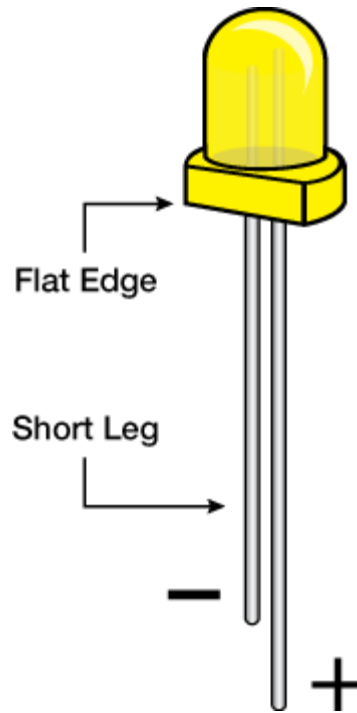
A	AudioIn01	Pi Audio Input Jack (microphone)
	Routine:	unknown

- C AudioOut01 Pi Audio Output jack (head phones)
Routine: unknown
- C AudioTV01 Pi Television speaker output
Routine: omxplayer Music/coupe.mp3
Also >\$ amixer cset numid=32 for audioTV
and > \$ amixer cset numid=31 for audioOut01
Also >\$uname -m gives aarah64???
Also >\$ omxplayer Music/Coupe.mp3
Also >\$ aplay Music/alarm.wav
- A piIP
>\$ ifconfig | grep inet >text/ip.txt
- C piArea routine g_piRoom = parameter
A osDistro
>\$ cat *etc*pi-issue shows the long Distro number
- A piSN
>\$ grep Serial *proccpuinfo*
- A osName
>\$ lsb release -a shows “buster”

5.1.3 led01 ioDevice LED

The control of the LED and pushButton are described in detail in Source 8.

This LED is connected to GPIO 12 which will provide a positive (+) voltage when ON (1). when OFF (0) the signal voltage coming out of GPIO 12 will be almost at Ground.



Note: It matters how you plug in your LED! Current can only flow in one direction through an LED, so pay careful attention to the leads. The short lead on the LED should be connected to the ground or through the 330 ohm resistor to the ground..

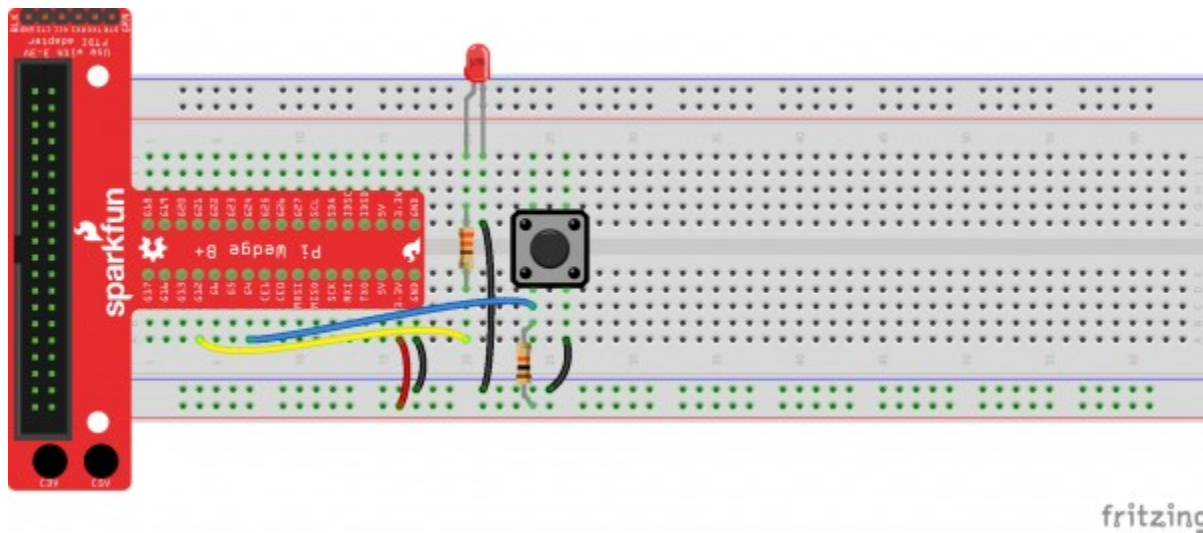
This LED (and the following Pushbutton) can be easily connected to the Raspberry Pi General Purpose Input / Output (GPIO) terminals (listed below) using the wedge (shown after that).

Raspberry Pi 3 GPIO Pinout

GPIO_GEN	Functions	GPIO	Pin	Pin	GPIO	Functions	GPIO_GEN
		3.3V	1		2	5V	
	SDA1 (I ² C)	GPIO2	3		4	5V	
	SCL1 (I ² C)	GPIO3	5		6	GND	
GCLK		GPIO4	7		8	GPIO14 TXD0 (UART)	
		GND	9		10	GPIO15 RXD0 (UART)	
GEN0		GPIO17	11		12	GPIO18 PWM0, CLK (PCM)	GEN1
GEN2		GPIO27	13		14	GND	
GEN3		GPIO22	15		16	GPIO23	GEN4
		3.3V	17		18	GPIO24	GEN5
	MOSI (SPI)	GPIO10	19		20	GND	
	MISO (SPI)	GPIO9	21		22	GPIO25	GEN6
	SCLK (SPI)	GPIO11	23		24	GPIO8 CE0 (SPI)	
		GND	25		26	GPIO7 CE1 (SPI)	
		ID_SD	27		28	ID_SC	
		GPIO5	29		30	GND	
		GPIO6	31		32	GPIO12 PWM0	
	PWM1	GPIO13	33		34	GND	
	FS (PCM), PWM1	GPIO19	35		36	GPIO16	
		GPIO26	37		38	GPIO20 DIN (PCM)	
		GND	39		40	GPIO21 DOUT (PCM)	

The diagram below shows the Wedge which is used to connect electronics to the Pi.

5.1.3.1 Wedge and Breadboard



5.1.3.2 Python routines for the LED device

The following Python3 software can be used to turn the LED on and off.
Copy the gpio software

```
pip install Python3.rpi.gpio
```

NB The above statement may have been deprecated, use the GUI to setup the GPIO pins

In a new file, copy the following code:

```
import time
import Python3.RPi.GPIO as GPIO

# Pin definitions
led_pin = 12

# Suppress warnings
GPIO.setwarnings(False)

# Use "GPIO" pin numbering
GPIO.setmode(GPIO.BCM)

# Set LED pin as output
GPIO.setup(led_pin, GPIO.OUT)

# Blink forever
while True:
    GPIO.output(led_pin, GPIO.HIGH) # Turn LED on
    time.sleep(1)                   # Delay for 1 second
    GPIO.output(led_pin, GPIO.LOW) # Turn LED off
    time.sleep(1)                   # Delay for 1 second
```

Save the file (I named my file blink.py). Run the code from the terminal by entering:

```
python blink.py
```

Running this program will cause the LED to blink on for 1 second and then off for 1 second. Click Ctrl-C to abort this program.

5.1.4 pb01 ioDevice (Pushbutton)

This pushbutton is connected to GPIO 4

Copy the following code:

```
import time
import RPi.GPIO as GPIO

# Pins definitions
btn_pin = 4    (connect it through the pushButton to 3v3.)
led_pin = 12   (connect it through 180 ohms to the long leg of the LED,
               and connect the long leg of the LED to GND.)

# Set up pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(btn_pin, GPIO.IN)
GPIO.setup(led_pin, GPIO.OUT)

# If button is pushed, light up LED
try:
    while True:
        if GPIO.input(btn_pin):
            GPIO.output(led_pin, GPIO.LOW)
        else:
            GPIO.output(led_pin, GPIO.HIGH)

# When you press ctrl+c, this will be called
finally:
    GPIO.cleanup()
```

Save the code as button.py and then run it.

When the pushbutton is pressed, the LED will turn on. When it is released, the LED will turn off.

5.1.5 The Pi vcgenclmd options

```
vcgenclmd commands:
commands="vcos, ap_output_control, ap_output_post_processing,
          vchi_test_init, vchi_test_exit,
          pm_set_policy, pm_get_status, pm_show_stats, pm_start_logging,
          pm_stop_logging, version, commands,
```

```
set_vll_dir, led_control, set_backlight, set_logging, get_lcd_info,
set_bus_arbiter_mode,
cache_flush, otp_dump, codec_enabled, measure_clock, measure_volts,
measure_temp, get_config,
hdmi_ntsc_freqs, render_bar, disk_notify, inuse_notify, sus_suspend,
sus_status, sus_is_enabled,
sus_stop_test_thread, egl_platform_switch, mem_validate, mem_oom,
mem_reloc_stats, file,
vctest_memmap, vctest_start, vctest_stop, vctest_set, vctest_get"
```

5.1.6 tmp00AnlgIn (using the TMP102) on a Raspberry Pi

Wire up the TMP102 to the Raspberry Pi as follows:

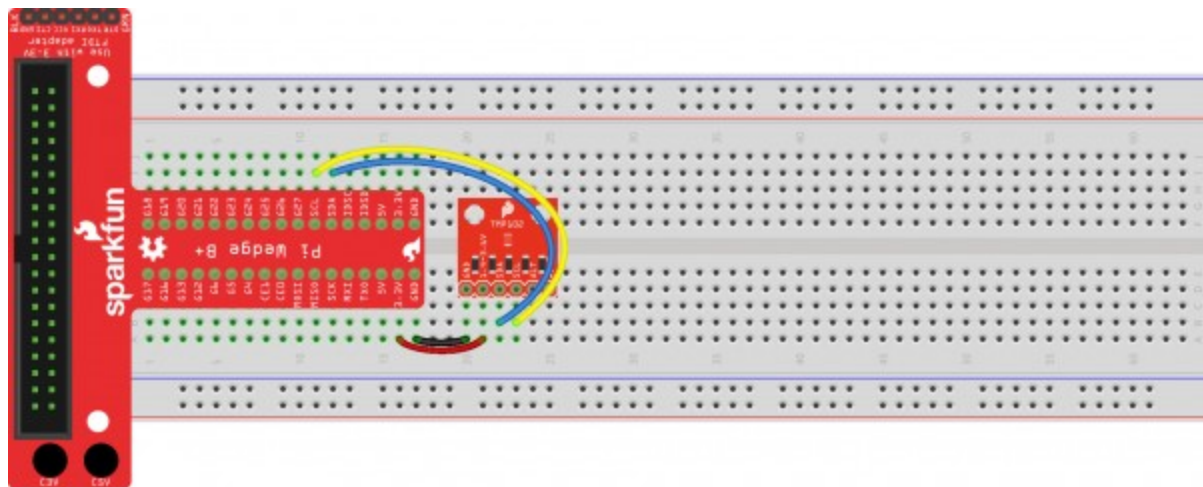
Connect **SDA1** (GPIO2, pin 3) to SDA on the TMP102

- Connect **SCL1** (GPIO3, pin 5) to SCL on the TMP102

- Connect power (3.3 V) to VCC on the TMP102

- Connect ground (GND) to GND on the TMP102

To do this, 4 tiny wires must be soldered to the TMP102 to be used as pins.



fritzing

install the python smb bus routines (Source 4 for more info):

```
sudo apt-get install python3-smbus
```

In a new file, copy in the following code:

```
import time
import smbus

i2c_ch = 1
```

```

# TMP102 address on the I2C bus
i2c_address = 0x48

# Register addresses
reg_temp = 0x00
reg_config = 0x01

# Calculate the 2's complement of a number
def twos_comp(val, bits):
    if (val & (1 << (bits - 1))) != 0:
        val = val - (1 << bits)
    return val

# Read temperature registers and calculate Celsius
def read_temp():

    # Read temperature registers
    val = bus.read_i2c_block_data(i2c_address, reg_temp, 2)
    temp_c = (val[0] << 4) | (val[1] >> 5)

    # Convert to 2s complement (temperatures can be negative)
    temp_c = twos_comp(temp_c, 12)

    # Convert registers value to temperature (C)
    temp_c = temp_c * 0.0625

    return temp_c

# Initialize I2C (SMBus)
bus = smbus.SMBus(i2c_ch)

# Read the CONFIG register (2 bytes)
val = bus.read_i2c_block_data(i2c_address, reg_config, 2)
print("Old CONFIG:", val)

# Set to 4 Hz sampling (CR1, CR0 = 0b10)
val[1] = val[1] & 0b00111111
val[1] = val[1] | (0b10 << 6)

# Write 4 Hz sampling back to CONFIG
bus.write_i2c_block_data(i2c_address, reg_config, val)

# Read CONFIG to verify that we changed it
val = bus.read_i2c_block_data(i2c_address, reg_config, 2)
print("New CONFIG:", val)

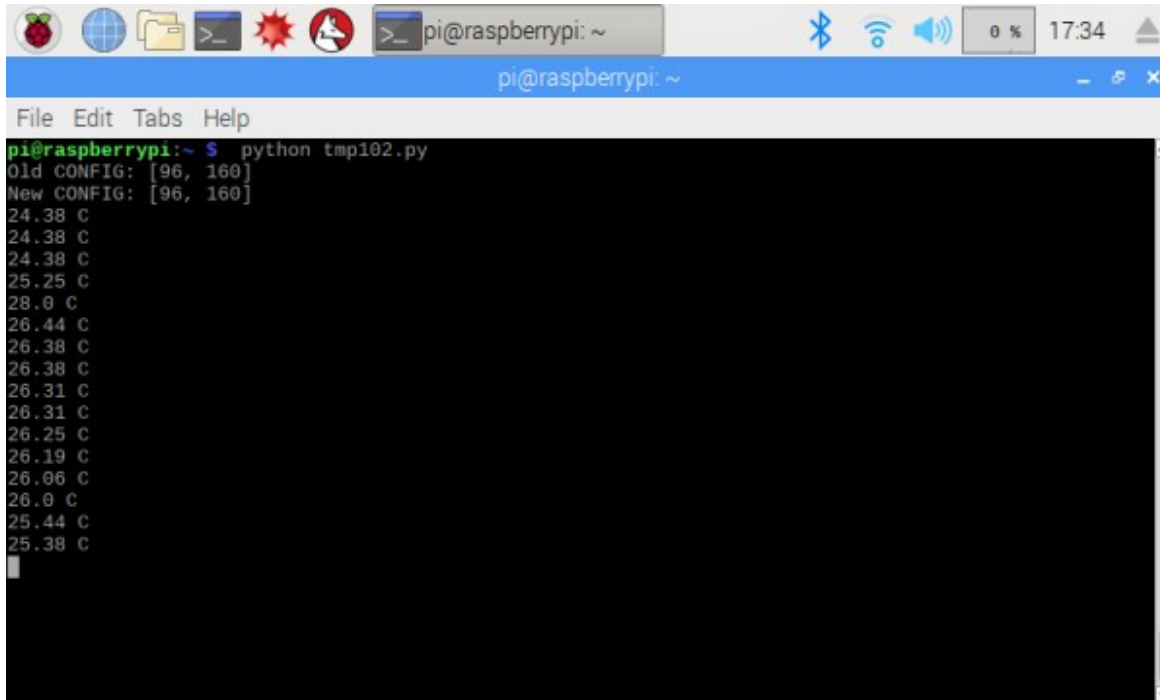
```

```
# Print out temperature every second
while True:
    temperature = read_temp()
    print(round(temperature, 2), "C")
    time.sleep(1)
```

Save the file (e.g. as tmp102.py), and run it with Python:

```
python tmp102.py
```

You should see the 2 bytes in the CONFIG register be updated and then the temperature is printed to the screen every second.



The screenshot shows a terminal window on a Raspberry Pi. The window title is "pi@raspberrypi: ~". The terminal output shows the execution of the Python script "python tmp102.py". The output displays the CONFIG register values and temperature readings every second. The CONFIG register values are [96, 160] for both old and new configurations. The temperature readings range from 24.38 C to 28.0 C.

```
pi@raspberrypi:~ $ python tmp102.py
Old CONFIG: [96, 160]
New CONFIG: [96, 160]
24.38 C
24.38 C
24.38 C
25.25 C
28.0 C
26.44 C
26.38 C
26.38 C
26.31 C
26.31 C
26.25 C
26.19 C
26.06 C
26.0 C
25.44 C
25.38 C
```

For more details about the TMP102 operation, refer to Source 5 below. The complete TI TMP102 datasheet is in Source 6 below. The TMP102 operates on the I2C serial bus which is described in Source 7 below.

5.2 Mathematica routines

It is possible that Mathematica routines can move files directly to the server. But probably only to the server at mathematica.com .

5.3 HTML routines

5.4 PHP routines

5.5 User HTML procedures

6. Related Documents

Document 1: /pi/home/pi/Python/Prog01C.py on my Raspberry Pi

7. Sources

Source 1. Raspberry Pi Python Functions: <https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/all>

Source 2: An Elaborate Pi Home Monitor:

<http://www.raspberry-pi-geek.com/Archive/2014/05/Automate-and-monitor-the-physical-systems-in-your-home>

Source 3: Python Underscore Syntax: <https://hackernoon.com/understanding-the-underscore-of-python-309d1a029edc?gi=572ccf06d3f8>

Source 4: python simbus software: <https://www.electronicwings.com/raspberry-pi/python-based-i2c-functions-for-raspberry-pi>

Source 5: TMP102 on Pi Python routines <https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/experiment-4-i2c-temperature-sensor>

Source 6: TI TMP102 Data Sheet

https://www.sparkfun.com/datasheets/Sensors/Temperature/tmp102.pdf?_ga=2.192862058.1956998499.1557989498-1748529165.1557989498

Source 7: I2C Explanation: <https://learn.sparkfun.com/tutorials/i2c>

Source 8: Spark Fun Experiment 1: <https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/experiment-1-digital-input-and-output>

Source 9: Python type syntax (esp. strings): <https://docs.python.org/3/library/stdtypes.html#str.split>

Source 10: Python type syntax (esp. strings): [139 Pi: PiR2A Area Controller Prototype \(139.html\)](#)

Source 11: Previous PiR2 Data Log System Definition V11 (pdf) : [PiR2 DataLog Definition11 pdf](#)

/w/PiR2_DataLog_Definition15.odt stored in mrmrLaptop C:...Documents/2022//DocsC/PiR2