Setting up the Raspberry Pi based gasmeter

Louis Marais

Start: 2021-07-06
Last: 2021-07-09

Visit the github repository for more information: https://github.com/ELouisMarais/Gasmeter/

I use Raspberry Pi 2B, 3B or 3B+ for this application because I need a LAN interface and I think a Pi 4 is overkill but you are welcome to use any Pi you like. The 3D models for a case to house the gasmeter is designed for the Pi 2 / 3. The LAN port on the Pi 4 is located differently, so the case will not work.

The instructions in this document is for Raspberry Pi OS 'Buster' 32 bit, based on Debian 10.

1. Install the operating system, set up the Pi and construct the gasmeter.

Install the latest 'Lite' version of Raspberry Pi OS (Raspbian) and make sure the Pi is accessible using `SSH`. Use `raspi-config` with an attached keyboard and monitor or add an empty file called `ssh` to the `/boot` partition before installing the micro-SD card in the Pi. (Or add the file to the `/boot` partition before you install the micro-SD card in the Pi.)

Set up the Pi to your liking. I change the host name and password, set the time zone, set up access though our proxy (you will likely need to do this too if you are behind a firewall), update the Pi, disable Bluetooth and Wi-Fi and set up Chrony. (But, how do I do this? See the appendixes, they will help!)

Power down the Pi and disconnect everything from it.

Build the adapter board that sits on top of the Pi (can't call it a HAT, because it does not comply with the requirements for a HAT) and, if you like, print an enclosure for the system. Most of my systems sit on a piece of plywood because it is a reasonably safe way to protect the Pi, and makes modifying things easy. The board has a number of unused connectors on it, that you can use if you want, but do not populate these if you are not going to use them as this adds unnecessary cost to the project. TODO: Write a guide to building the board and assembling the system.

Assemble the system and connect it to a gasmeter sensor (or any switched type output – the adapater board has a pull-up and a low pass filter on the input which then goes to an inverting buffer that is used to trigger the Pi). The sensor that is used in our gasmeter is a Honeywell model IN-Z41. You can also add a power button (I usually add this to mine) and power LED (because of the LCD display, I usually do not add a power LED).

Once the system is assembled re-connect it to your network, power up and connect to it using `SSH`.

2. Set up the power button and power LED.

The power button is connected to pin 5 on the 40 pin expansion header (the `SCL` pin). Pulling this pin low while the Raspberry Pi has a power supply connected but is off powers up the Pi. So we only need to add the shutdown capability. Luckily this is easy because the good folks at the Raspberry Pi foundation has added this functionality in an overlay. Edit the `/boot/config.txt` file:

```
sudo nano /boot/config.txt
```

and add the following line at the end:

```
dtoverlay=gpio-shutdown
```

Then save using <Ctrl>-S and exit using <Ctrl>-X.

I usually do not connect the power LED for the gas meter application, as I believe having the LCD as a power on indicator is enough. But if you want to have a power LED, here are the instuctions.

The power LED is connected to pin 8 (TxD). This pin is set to high when the Pi boots if the serial port is enabled. (So this means you should not use the serial port and the power LED at the same time – the flickering LED may confuse users, so if that is your situation you will need to find a different solution.) To add the power LED functionality we will need to activate the serial port, but not as a serial terminal. Run `raspi-config`:

```
sudo raspi-config
```

Select `3 Interfacing Options` then <Tab> to <Select> and press <Enter>. Select `P6 Serial` and <Tab> to <Select> and press <Enter>. Answer <No> to the question "`Would you like a login shell to be accessible over serial?`" and press <Enter>. Answer <Yes> to the question "`Would you like the serial port hardware to be enabled?`" and press <Enter>. Acknowledge the information screen and then select <Finish> and press <Enter> when you are returned to the main `raspi-config` menu. Select <Yes> to the question "`Would you like to reboot now?`" and press <Enter>. Once the reboot is complete you can log back in and test the power button by using it to shut the system down and starting it again.

3. Clone, compile and set up the software.

We need to execute some preparatory steps. We will need a `bin`, `data`, `etc` and a `logs` directory in the local user's home. We also need a `bin`, `etc` and `logs` directory for the `root` user:

```
cd
mkdir bin data etc logs src
```

Check that the required directories exist:

```
ls -l
```

The response should be similar to this:

```
total 20
drwxr-xr-x 2 pi pi 4096 Jul  6 14:21 bin
drwxr-xr-x 2 pi pi 4096 Jul  6 14:21 data
drwxr-xr-x 2 pi pi 4096 Jul  6 14:21 etc
drwxr-xr-x 2 pi pi 4096 Jul  6 14:21 logs
drwxr-xr-x 2 pi pi 4096 Jul  6 14:22 src
```

The software is hosted on github. We will use `git` to clone the distribution to the Pi, but we first need to install `git`. Make sure your Pi is up to date (see appendix) then execute:

```
sudo apt install git
```

Set up `git` to use a proxy if you have one:

```
git config --global http.proxy http://proxyUsername:proxyPassword@proxy.server.com:port
git config --global https.proxy https://proxyUsername:proxyPassword@proxy.server.com:port
```

We also need to install the `wiringpi` package:

```
sudo apt install wiringpi
```

A note about the `wiringpi` package: On 6 August 2019 the `wiringpi` package was deprecated (see http://wiringpi.com/news/). I have rewritten the `gasmeter` and `countpulses` software (used to be C, now Python 3), so these no longer use `wiringpi`. The `intensity` software still uses `wiringpi` as I need hardware level access to the pulse width modulation pin that drives the LCD backlight. At this time I could not find another simple solution, but I will keep looking. For now the `wiringpi` package for Raspberry Pi OS is still available and works fine.

Clone the distribution to the Pi. Perform these steps:

```
cd
mkdir src
cd src
git clone https://github.com/ELouisMarais/Gasmeter/
cd Gasmeter
```

The software is still under development (or it may just be that I'm lazy and have not moved the software to the `master` branch), so we need to switch to the `develop` branch:

```
git checkout develop
cd software
```

Now build the software:

```
./build
```

This will compile and copy all the required packages to the required locations.

We also need the `kickstart.pl` script from the OpenTTP distribution. Clone the distribution and install the required script and library manually. You can also install the complete OpenTTP distribution but that is a huge overkill and you will need to remove a lot of scripts and disable some of the services that get installed. Manually installing only the bits we need is much simpler. Perform the following steps:

```
cd ~/src
git clone https://github.com/openttp/openttp.git
cd openttp
git checkout develop
cd software/system/src
sudo cp kickstart.pl /usr/local/bin/
sudo mkdir /usr/local/lib/site_perl
sudo cp TFLibrary.pm /usr/local/lib/site_perl/
```

We now need to create some files that contains the information our system needs. These files need to be in the `~/etc` directory:

```
cd ~/etc
```

Create and populate the file used to specify the intensity of the LCD backlight:

```
echo {backlight level} > backlightlevel
```

`{backlight level}` is a value between 1 (off) and 1023 (maximum brightness).

Copy the current reading on your gas meter to the `~/etc/meterreading` file:

```
echo {meter reading} > meterreading
```

where `{meter reading}` is the reading in cubic meters displayed on your gas meter.

Create a configuration file for the gas meter and pulse counter sofware:

```
nano gasmeter.conf
```

and add these line to it. Replace the bits in curly brackets with sensible values:

```
[main]
# Max 10 characters for {room no} and {gas meter sn}
room no = {room no}
serial number = {gas meter sn}
# Default location for meter readings, do not change
meter reading file = etc/meterreading
# File used by kickstart.pl
lock file = logs/gasmeter.lock

[gasmeters]
# Define one or more targets, if more than one target, an additional target
# called combined is automatically added, and a section called
# [combined] must be added by the user.
# If a single target is defined the name of the target MUST be
# the same as the filename for [main][meter reading file] (meterreading)
# otherwise the values will not be written to a location recognised by
# the gasmeter.py software.
targets = meterreading
# Example of multiple targets:
#targets = meter1, meter2, meter3
meter reading path = etc/
# Lock file for the countpulses.py software
lock file = logs/countpulses.lock

[meterreading]
# input pin for pulsed; use BCM numbering (not physical pin numbers!)
input = 21
# volume per pulse in cubic meters
volume per pulse = 0.01
data path = data/
# file format can be one of YYYY (for a whole year's data in one file),
# or YYYY-MM (monthly files) or MJD (a day per file).
file format = YYYY-MM

# Example of sections required for multiple targets example above.

[meter1]
input = 5
volume per pulse = 0.01
data path = data/meter1/
file format = MJD
```

```
[meter2]
input = 6
volume per pulse = 0.01
data path = data/meter2/
file format = YYYY

[meter3]
input = 16
volume per pulse = 0.01
data path = data/meter3/
file format = YYYY-MM

[combined]
data path = data/
file format = MJD
```

When you generate your configuration file, you can keep it simple, just add the lines you need. This is the one I use for my set up:

```
[main]
room no = {Room no}
serial number = {Meter serial number}
meter reading file = etc/meterreading
lock file = logs/gasmeter.lock

[gasmeters]
targets = meterreading
meter reading path = etc/
lock file = logs/countpulses.lock

[meterreading]
input = 21
volume per pulse = 0.01
data path = data/
file format = YYYY-MM
```

We will use a cron job to make sure the programs that should be running are running. For this we use the `kickstart.pl` script. This script requires a configuration file:

```
nano kickstart.conf
```

Add the following lines to the file:

```
targets = gasmeter, countpulses, broadcast

[gasmeter]
# target is used to generate a unique name for the log file and 'check' file
target = gasmeter
# this is the command that will be executed
command = bin/gasmeter.py
# lock file that will be tested
lock file = logs/gasmeter.lock

[countpulses]
target = countpulses
command = bin/countpulses.py
lock file = logs/countpulses.lock

[broadcast]
```

```
target = broadcast
command = bin/broadcast.py
lock file = etc/broadcast.lock
```

then save the file (<Ctrl>-S) and exit (<Ctrl>-X).

We want the cron daemon to regularly check that everything is running, in this case every 2 minutes. Create the crontab file:

```
nano crontab
```

Add the following line to the file:

```
*/2 * * * * /usr/local/bin/kickstart.pl # See ~/etc/kickstart.conf
```

then save the file (<Ctrl>-S) and exit (<Ctrl>-X).

The intensity program is run as a service by the root user. The root user needs to run this as it uses the hardware pulse width modulation (PWM) functionality of the CPU, and only root can access that. For this we need to create a service file:

```
sudo nano /etc/systemd/system/intensity.service
```

and add these lines to it (you do not have to add the comments if you don't want to):

```
# This file should live in /etc/systemd/system/
#
# Install / enable with:
#   sudo systemctl enable intensity
# Start with:
#   sudo systemctl start intensity
# Stop with:
#   sudo systemctl stop intensity
# Uninstall / disable with:
#   sudo systemctl disable intensity


[Unit]
# Control the intensity of the backlight of an LCD display
# intensity is set by a value between 0 and 1024 in the
# /home/pi/etc/backlightlevel file.
Description=Control the intensity of the backlight of an LCD display

[Service]
# Command to execute when the service is started
ExecStart=/home/pi/bin/intensity
# Restart automatically on failure
Restart=on-failure

[Install]
# Start at boot time
WantedBy=default.target
```

then save the file (<Ctrl>-S) and exit (<Ctrl>-X).

4. Test the software

OK, here we go. To test the `intensity` software we can run it:

```
cd ~/bin
sudo ./intensity
```

Use `<Ctrl>-C` to quit the program.

If it all worked, we can install it as a service, then start it:

```
sudo systemctl enable intensity
sudo systemctl start intensity
```

Now change the `backlightlevel` value to make sure the intensity changes as expected. Valid values are between 0 and 1023:

```
cd ~/etc
echo {value} > backlightlevel
```

Once you are happy echo a value to the LCD backlight that fits the environment that the gas meter will be installed in.

OK, now let's test the other programs:

```
cd ~/bin
./gasmeter.py &
```

You can now adjust the LCD contract potentiometer if the LCD readout is not clear enough.

```
./countpulses.py &
```

You should see the gas meter reading in the lower right hand side of the display increment as the gasmeter runs. If you have the correct value in the `~/etc/meterreading` file the gasmeter reading and the reading on the LCD should correspond.

Once everything checks out properly we can install the crontab file we created earlier to make sure everything will run automatically.

Install the `crontab` file for the Pi user:

```
cd ~/etc
crontab crontab
```

This should be it. Your gasmeter is running, and the values are saved in the `~/data/` directory.

5. Set up broadcasting of the data and collecting it on another system.

You could set up data transfer to another system for reporting of the data, but if you want to do that is semi-real time it means a lot of data transfers and files being overwritten. I wrote the `broadcast.py` and `recvmsg.py` packages to facilitate broadcast and reception of the data over the network.

The first step is to set up broadcasting. This is done on the gas meter Pi. It requires a configuration file in the `~/etc/` directory. Create the file:

```
cd ~/etc
nano broadcast.conf
```

and add these lines to the file:

```
[main]
name = gasflow
targets = flow
path = etc
lockfile = broadcast.lock

[flow]
file = meterreading
```

Change value in the "`name =`" line to something that makes sense. This is the parameter that will be picked by `recvmsg.py` to identify the gas meter (I could have used IP addresses, but our network assigns them with DHCP, so you cannot be sure that the IP address will stay the same).

The software will now transmit the value in the `~/etc/meterreading` file every time the file modification time stamp changes. The logging software ensures that this happens at least once an hour. The data is transmitted using the UDP protocol so the message is repeated three times to ensure it is read by the receiving software.

The software for receiving the messages should be set up on a different machine on the same network.

Create a configuration file in the `~/etc/` directory (create it if it does not exist):

```
nano ~/etc/recvmsg.conf
```

and add the following lines:

```
[main]
targets = flow
lockfile = recvmsg.lock
lock path = tmp

[flow]
name = gasflow
description = Gas flow meter readings
data = meterreading
path = data/
file extension = gasflow
file type = MJD | YYYY-MM | YYYYMM | YYYY
```

The run the software to test it.

Back to the gas meter machine: Add an entry in the kick start configuration for `broadcast`. Modify the `targets` line:

```
targets = gasmeter, countpulses, broadcast
```

and add a section for `broadcast`:

```
[broadcast]
target = broadcast
command = bin/broadcast.py
lock file = etc/broadcast.lock
```

Next time kick start runs broadcasting will start. Check the `~/etc/` directory to see that the lock file gets updated.

Appendixes

Almost every one of these include a reboot step. If you want, you can do all of them and then reboot; it is not necessary to reboot after every one.

A. Change the password on the Pi

You could use the configuration utility (`raspi-config`), but it is easy doing it in a terminal window. Type the command:

```
passwd
```

then enter you current password and your new password (twice). Done.

B. Change the host name on the Pi

Run the configuration program:

```
sudo raspi-config
```

Select `1. System Options` then use the `<Tab>` key to highlight `<Select>` and press `<Enter>`.

Select `S4 Hostname` , `<Tab>` to `<Select>` and press `<Enter>`. Read the message then press `<Enter>`. Type your new host name then use `<Tab>` to select `<Ok>` and press `<Enter>`.

Once the changes are applied, use `<Tab>` to select `<Finish>` then press `<Enter>`. You need to reboot the Pi for the change to take effect.

C. Set the time zone on the Pi

Run the configuration program:

```
sudo raspi-config
```

Select `5. Localisation Options` then use the `<Tab>` key to highlight `<Select>` and press `<Enter>`.

Select `L2 Time Zone` , `<Tab>` to `<Select>` and press `<Enter>`. Select your time zone from the list and then use `<Tab>` to select `<Ok>` and press `<Enter>` (you may need to make more than one selection).

Once the changes are applied, use <Tab> to select <Finish> then press <Enter>.

Note that the date may not be correct, so check it by typing this in the terminal window:

```
date
```

If it is way off, some things may not work correctly. To set it type:

```
sudo date -s "6 Jul 2021 13:31:00"
```

substituting the date with the correct one (obviously).

All done!

D. Disable Bluetooth and Wi-Fi

Edit the /boot/config.txt file as the root user:

```
sudo nano /boot/config.txt
```

and add these lines* at the end of the file:

```
dtoverlay=disable-wifi
dtoverlay=disable-bt
```

Save the file using <Ctrl>-S and exit using <Ctrl>-X.

Disable the service that initialises Bluetooth modems connected by UART by executing this command:

```
sudo systemctl disable hciuart
```

Then reboot the Pi for the changes to take effect:

```
sudo reboot
```

* For older operating systems (Debian 9, etc.) search the /boot/overlays directory with wifi and bt to find the names of the overlays to disable. For Raspian Stretch (Debian 9) the overlays are called pi3-disable-wifi and pi3-disable-bt for example.

E. Set up access through a proxy server

For general internet access edit (or create) the environment file:

```
sudo nano /etc/environment
```

Add the following lines to the file, replacing xxx.xxx.xxx.xxx with the IP address of your proxy server and yy with the port number:

```
http_proxy="http://xxx.xxx.xxx.xxx:yy/"
https_proxy="http://xxx.xxx.xxx.xxx:yy/"
# Optional - no proxy for local addresses
```

```
no_proxy="localhost,.localdomain,127.0.0.1"
```

and save the file. If your proxy requires user / password information modify the lines above to read:

```
http_proxy="http://username:password@xxx.xxx.xxx.xxx:yy/"
https_proxy="http://username:password@xxx.xxx.xxx.xxx:yy/"
```

Set up a proxy for apt. Create the file:

```
sudo nano /etc/apt/apt.conf.d/10proxy
```

and add the following to it:

```
Acquire::http::Proxy "http://xxx.xxx.xxx.xxx:yy/";
Acquire::https::Proxy "https://xxx.xxx.xxx.xxx:yy/";
```

then save the file. If your proxy requires user / password information modify the lines above to read:

```
Acquire::http::Proxy "http://username:password@xxx.xxx.xxx.xxx:yy/";
Acquire::https::Proxy "https://username:password@xxx.xxx.xxx.xxx:yy/";
```

Reboot the Pi:

```
sudo reboot
```

and when it is done, log back on.

F. Update the Pi

Update the Pi using:

```
sudo apt update
```

and

```
sudo apt upgrade
```

to ensure you have the latest software. You should reboot the Pi after doing the initial update.

G. Set up a Network Time service

I used to use `ntp` as my preferred service, but recently changed to `chrony`. It was time.

Update the Pi as described in section 1. Then run:

```
sudo apt install chrony
```

to install the software. The default configuration will be used and `chrony` will be started after it is installed. The default configuration will most likely not be ideal so stop `chrony`:

```
sudo systemctl stop chrony
```

and edit the configuration file:

```
sudo nano /etc/chrony/chrony.conf
```

Remove or comment out the `pool` entries and add the local servers on your network and / or local pool servers (have a look on the internet to see how to go about this) then save the configuration using `<Ctrl>-S` and exit using `<Ctrl>-X`.

Restart `chrony`:

```
sudo systemctl start chrony
```

You can check if it is running using either:

```
systemctl status chrony
```

or

```
chronyc sources
```

The `chronyc` command can be used to provide lots of information about your network time server. There are lots of information (sometimes a bit confusing) for `chrony` on the Internet. Happy hunting!

WARNING: Network time servers are frequently hacked and used for questionable purposes. Make sure you lock down your server if your Pi is exposed to the Internet.

H. Set up Pi CPU temperature logging

This is completely optional, but can be useful.

If you want to you can set up logging of the Pi CPU temperature. The required software is part of the OpenTTP distribution. We already cloned the OpenTTP distribution to get the kick start subsystem, so we just need to get the required script from the distribution.

Copy the Pi CPU temperature log script:

```
cp ~/src/openttp/software/gpscv/common/bin/logpicputemp.pl ~/bin/
```

From the user root (`/home/pi/`) create the temporary directory required by the software, and a data directory for the log files:

```
cd
mkdir status data/cputemp
```

Create the required configuration file:

```
cd ~/etc
nano gpscv.conf
```

Add the following lines to the file, then save it:

```
[paths]
cputemp data = data/cputemp

[cputemp]
lock file = logs/logpicputemp.lock
```

Test the software:

```
cd ~/bin
./logpicputemp.pl -d
```

You should see something similar to this, if not, carefully check the configuration and fix any issues:

```
20:25:08 Script name: logpicputemp.pl
20:25:08 Authors: Louis Marais
20:25:08 Version: 1.0
20:25:08 Home directory: /home/pi
20:25:08 Log path: /home/pi/logs
20:25:08 Configuration file: /home/pi/etc/gpscv.conf
20:25:08 $lockFile: /home/pi/logs/logpicputemp.lock
20:25:08 /home/pi/data/cputemp/
20:25:08 Opening /home/pi/data/cputemp/59169.temp
20:25:08 temp=52.1'C

20:25:08 52.1
saveData subroutine
$mjd: 59169
$temp: 52.1
```

Use `<Ctrl>-C` to stop the script.

Add Pi CPU logging to kickstart:

Edit the `kickstart.conf` file:

```
cd ~/etc
nano kickstart.conf
```

Add `cputemp` to the `targets` list and add a section for `cputemp`:

```
...
targets = gasmeter, countpulses, broadcast, cputemp
...

[cputemp]
target = cputemp
command = bin/logpicputemp.pl
lock file = logs/logpicputemp.lock
```

And that should be it. Wait a few minutes then check that the script is running and data is being saved.

I. Creating a temporary disk for storing of often written files

We use a temporary disk with the OpenTTP to create a directory in user space where often written files are stored. This prevents SD card burnout. It is much more of a problem with the OpenTTP due to the large number of files that are overwritten regularly, but it may be useful here to extend the ife of the SD card.

The directory is created by modifying the `fstab` as the `root` user:

```
sudo -i
nano /etc/fstab
```

Add the following line to the file (this creates a 100 MB particion) then save the file (note that this is all ONE line):

```
tmpfs /home/pi/lockStatusCheck tmpfs
defaults,noatime,nosuid,uid=pi,gid=pi,mode=0755,size=100M 0 0
```

Reboot the Pi for the change to take effect. Check that the expected direcotry shows up in the user space. As the `pi` user:

```
ls -l ~/

...
drwxr-xr-x 2 pi pi  160 Nov 17 15:38 lockStatusCheck
...
```

The kick start subsystem will automatically use this new directory. It is written to use this location for its check files that gets written every time it is called by the `cron` process.

If it is installed the Pi CPU temperature logger will also automatically use this location for its status file, called `cputemp`, that holds the last CPU temperature value.

For additional benefits change the configuration files to make use of this handy new space and sleep soundly knowing that you have added some life to your SD card. This also prevents stale lock files surviving a reboot, as this directory is recreated at boot. The required changes are highlighted below:

```
~/etc/gasmeter.conf

[main]
...
lock file = lockStatusCheck/gasmeter.lock
...
[gasmeters]
...
lock file = lockStatusCheck/countpulses.lock

~/etc/broadcast.conf

[main]
...
lockfile = lockStatusCheck/broadcast.lock

~/etc/gpscv.conf

...
[cputemp]
```

```
lock file = lockStatusCheck/logpicputemp.lock

~/etc/kickstart.conf

...
[gasmeter]
...
lock file = lockStatusCheck/gasmeter.lock

[countpulses]
...
lock file = lockStatusCheck/countpulses.lock

[broadcast]
...
lock file = lockStatusCheck/broadcast.lock

[cputemp]
...
lock file = lockStatusCheck/logpicputemp.lock
```

J. Useful utilities

Here are some utilities I usually add to my systems.

`locate` – this does what it says, helps you to locate files. The database that holds the content of you disks are refreshed once a day, so if you want to locate something recently added you need to update the database before using `locate`. Install using:

```
sudo apt install mlocate
```

Then update the datebase:

```
sudo updatedb
```

To use just type:

```
locate {file}
```

where `{file}` is the filename (or partial filename, so you can guess, but be prepared for a large number of results!)

Watch this space for more!

References

These were useful during writing of this document (last accessed on 2021-07-09):

https://howchoo.com/g/njnlzjmyyjg/how-to-set-the-timezone-on-your-raspberry-pi

https://blog.sleeplessbeastie.eu/2018/12/31/how-to-disable-onboard-wifi-and-bluetooth-on-raspberry-pi-3/

https://www.raspberrypi.org/documentation/configuration/use-a-proxy.md

https://www.instructables.com/id/Adding-local-internet-proxy-settings-to-Raspberry-/

https://pinout.xyz/

https://raspberrypi.stackexchange.com/questions/77905/raspberry-pi-3-model-b-dtoverlay-gpio-shutdown