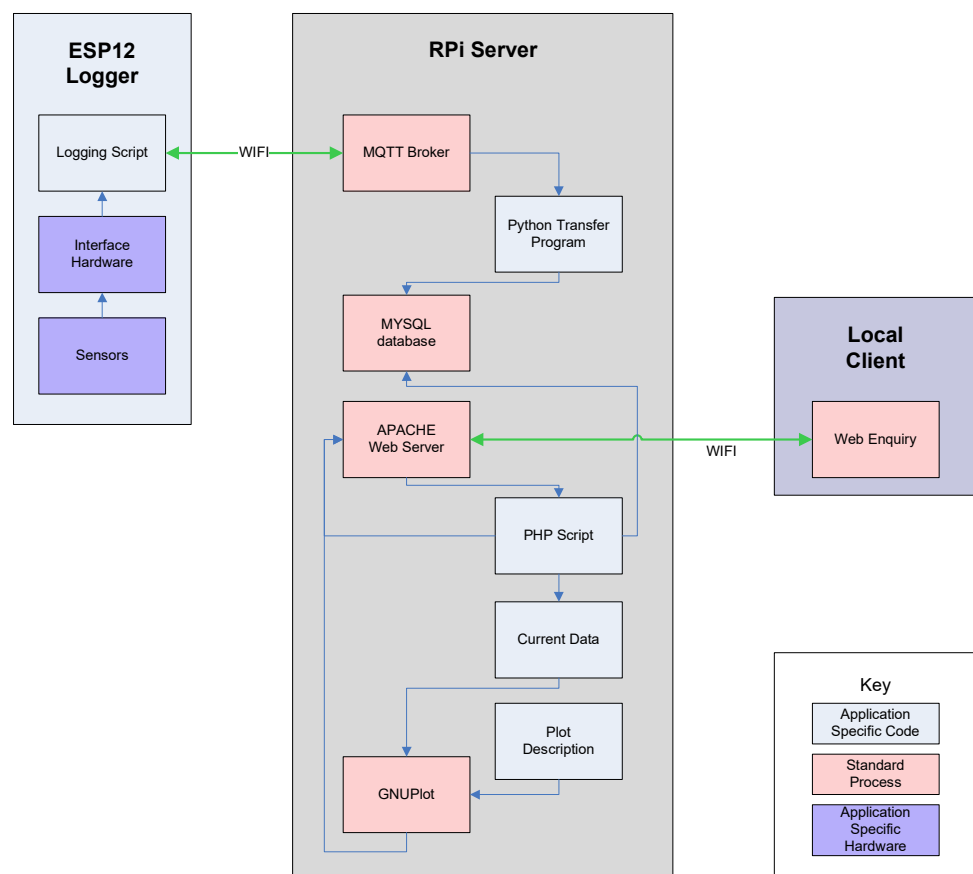


## IOT Home Automation Framework for ESP8266 Devices

The Internet of Things (IOT) revolution has started. The ultra-cheap hardware has arrived with the ESP8266 and the Raspberry Pi. The open source community has responded with well proven, core software modules. Putting it all together in an expandable system is still slightly daunting, especially if you want to keep control of your data by avoiding cloud services.

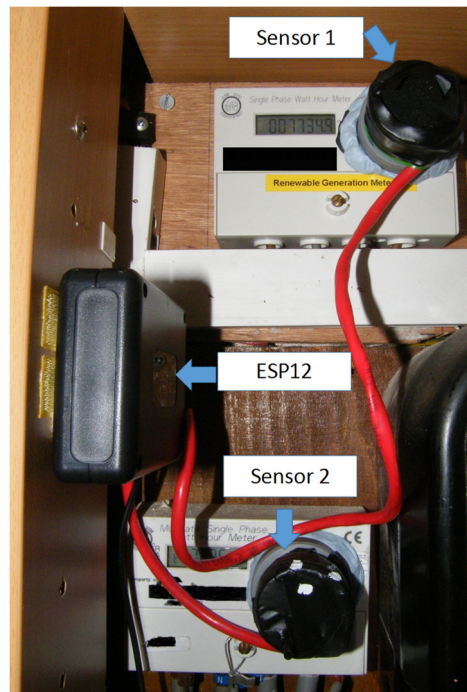
This project demonstrates transmitting data from a wifi-enabled, microcontroller based on the ESP8266, store it and display it in a web page on a local network. By using well proven core modules, glued together by simple, custom interface scripts, a robust system is produced, which, with straightforward modification, can be adapted to a wide range of applications. The general organisation is shown below –



One Raspberry Pi based, data and local web server can easily handle a number of IOT units supplying other data by just changing or adding a few scripts. My own plan is to extend the system to include temperature, sunshine, lightning activity, background radiation and heating oil level in the near future. These will all be built on ESP8266 devices, most likely the ESP12. All the data will be under local control, securely behind a firewall.

## Dual Pulse Count Logger Example

The specific project to demonstrate the approach is to log the electricity generation from a PV system and imported electricity from the grid as measured by two electricity meters.



The hardware for the logger uses phototransistors to sense the led pulses from electricity meters. The electricity meter provides one pulse per Wh passing through the unit. No electrical connection to the meter is required. The pulses are counted in the ESP12 and the count transmitted to the Raspberry Pi server every minute over the wifi.

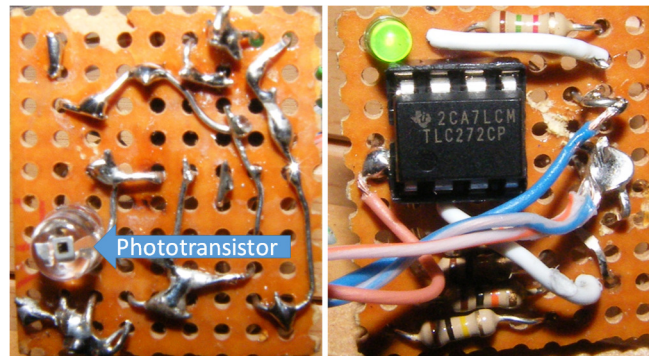
## Circuit

The circuit is shown below-



## Building the hardware

The unit was constructed in three parts, a sensor unit for each electricity meter and a central unit with the ESP and power supply. Putting the phototransistor and its comparator close together reduces the likelihood of electrical interference giving false counts. The sensor modules are double sided with the phototransistor on its own so it can be easily aligned with the meter led.

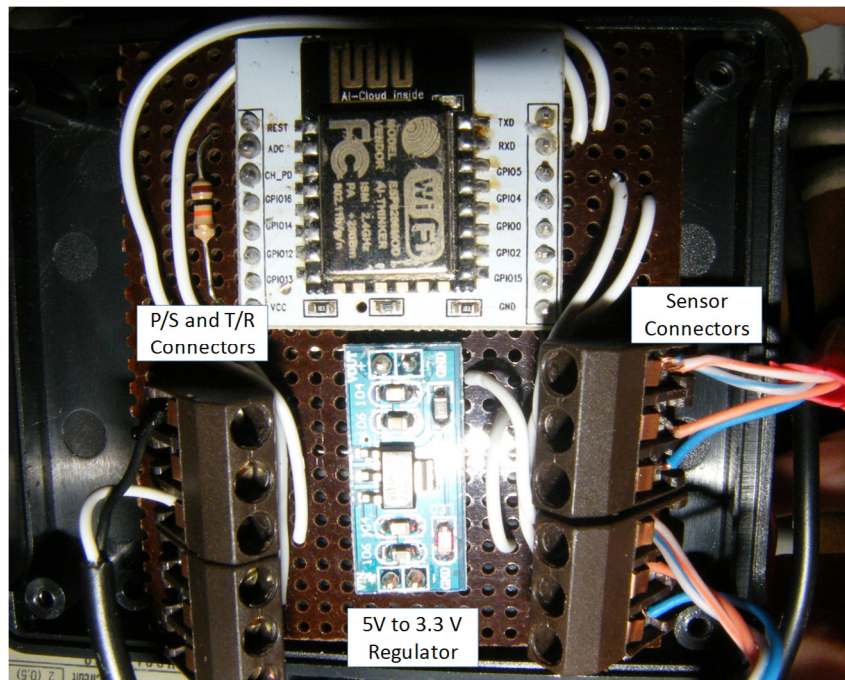


The sensor board was mounted in a piece of plastic pipe and the reverse sealed to exclude ambient light. With the led echoing pulses, the sensor modules can be tested if they are powered from a 3.3 V supply, without the ESP12 unit attached.



The unit was held in place and light excluded using a ring of flexible, sticky putty, such as Bluetack. Make sure the led on the back of the unit is responding to the pulses from the meter. It should blink off when a pulse is received.

The ESP12 was mounted on perforated sheet with a 5V to 3.3 V regulator, the pullup resistors and connectors for the external 5 V power supply and the sensor heads. A 5 V power supply with a step down regulator was used as this was cheaper than buying a 3.3 V regulated supply.



The ESP12 was programmed on a breadboard before installation, but the RX and TX lines were brought out to a connector for upgrades.

## Programming the ESP12 with Arduino IDE

The ESP12 was programmed using the Arduino IDE. To do this an additional module has to be added in to the basic Arduino IDE. This is described in <https://github.com/esp8266/Arduino>.

In addition to the ESP library, the library simplifying communications with an MQTT broker is needed. This is available from <https://github.com/knolleary/pubsubclient/>. Copy the ZIP file to your computer and then use the Arduino IDE to unpack it load it into the correct place as follows <https://learn.sparkfun.com/tutorials/installing-an-arduino-library>.

Sketch **ESP178dualpulse.ino** drives the ESP12. The basic operations it performs are

- Timing a 60 second reporting period
- Counting pulses on the input lines using interrupts
- Reconnecting to the wifi with a static IP address, if it is disconnected.
- Reconnecting to the message broker if required
- Sending counts to the message broker.

The sketch has comments which explain what is going on and pointers to which sections are application dependent.

Remember to set up the program for your network by filling in –

- your network ID
- your network password



- the static IP you select for this device
- the static IP of your Raspberry Pi based, data server

## Setting up the data server on the Raspberry Pi

A Raspberry Pi was used for this as it is powerful, cheap and low power. It can handle the expansion for more IOT units.

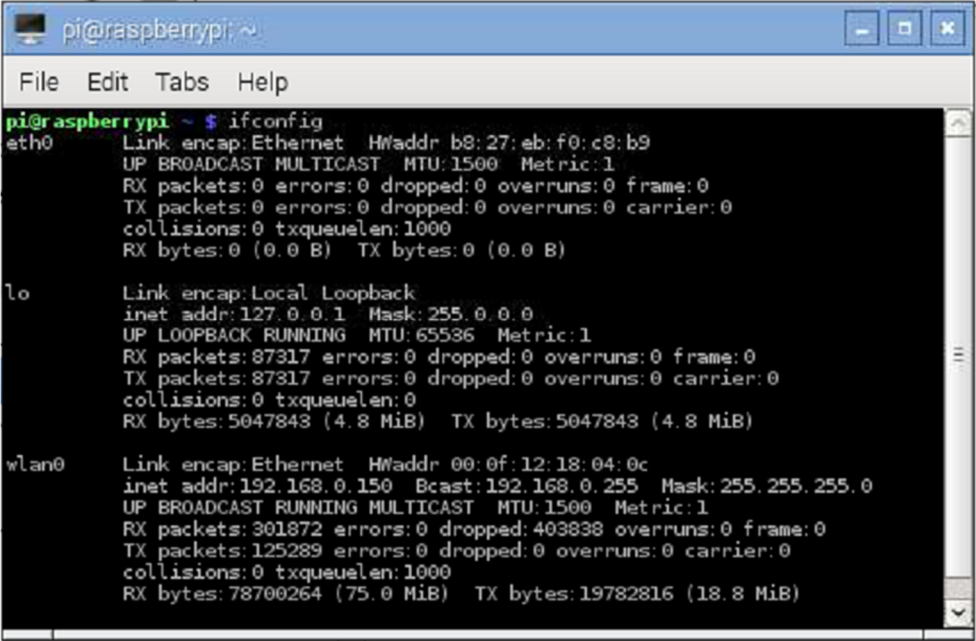
The basic Raspberry Pi setup is described in many places including <https://www.raspberrypi.org/help/quick-start-guide/> . Assuming the Raspberry Pi is setup for general use and all up to date, then various programs need to be installed. If you have not installed packages on a Raspberry Pi before the method is that followed in [http://elinux.org/Add\\_software\\_to\\_Raspberry\\_Pi](http://elinux.org/Add_software_to_Raspberry_Pi) .

Note, on security - to keep this simple and for most uses, it is assumed that the ESP unit, the Raspberry Pi server and the devices going to view the data are on the same local network and that the associated router is firewall protected. Thus the communications from the IOT devices do not implement any additional security. As a minimum change the administration password on the router from the default and use a strong password to allow devices to connect.

## Set a Static IP address on the Raspberry Pi

First set up a static IP address on the Raspberry Pi, this makes it simpler to send messages to it and see the data on it. Find the current IP address of the Raspberry Pi by typing at a command line.

ifconfig



```

pi@raspberrypi ~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:f0:c8:b9
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:87317 errors:0 dropped:0 overruns:0 frame:0
          TX packets:87317 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5047843 (4.8 MiB)  TX bytes:5047843 (4.8 MiB)

wlan0     Link encap:Ethernet  HWaddr 00:0f:12:18:04:0c
          inet addr:192.168.0.150  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:301872 errors:0 dropped:403838 overruns:0 frame:0
          TX packets:125289 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:78700264 (75.0 MiB)  TX bytes:19782816 (18.8 MiB)

```

The relevant entry is under wlan0, the inet addr – here 192.168.0.150.

Choosing a number >100 in the last block for the static address should avoid the router assigning the IP address to another computer, say 150. Your static address should have the same first three number blocks and 150 at the end.

The way static IP addresses are set on the Raspberry Pi has changed since mid-2015 and most descriptions on the web are wrong, if you are using the latest software.

Check your network and password are set up correctly in the file **/etc/wpa\_supplicant/wpa\_supplicant.conf**

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

It should contain the lines

```
network={
    ssid="yournetworkname"
    psk="yourpassword"
}
```

Now change the interface wlan0 section in file **/etc/dhcpd.conf** using

```
sudo nano /etc/dhcpd.conf
```

to

```
interface wlan0
static ip_address=192.168.0.150
static routers=192.168.0.1
static domain_name_servers=8.8.8.8 8.8.4.4
```

Note that you should put in the IP address of your router and that there is a space between the two groups in the last line.

The third file to look at is **/etc/network/interfaces**

```
sudo nano /etc/network/interfaces
```

and set the wlan0 section to

```
auto wlan0
allow-hotplug wlan0
iface wlan0 inet manual
xpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

To implement these changes reboot with

```
sudo reboot
```

Test – as before using

```
ifconfig
```

If all is well, your new static IP address should be reported.

## Setting up the MQTT broker

Using a dedicated well thought out message broker saves a lot of hassle. The arguments for using MQTT are well rehearsed, see <https://www.oasis-open.org/committees/download.php/49205/MQTT-OASIS-Webinar.pdf> . This project just uses a very basic implementation using the simplest elements.

Install Mosquitto, a great open source implementation of MQTT, for Linux. This is slightly more complicated than most package installations as it is not in the standard package stores –

```
curl -O http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
rm mosquitto-repo.gpg.key
cd /etc/apt/sources.list.d/
sudo curl -O http://repo.mosquitto.org/debian/mosquitto-repo.list
sudo apt-get update
sudo apt-get install mosquitto mosquitto-clients
```

The last installs the Mosquitto server and a Mosquitto client program which works at the command line as shown below

Test it is running -

```
sudo /etc/init.d/mosquitto stop
```

A suitable message should appear. Start it again.

```
sudo /etc/init.d/mosquitto start
```

To test the ESP12 dual pulse counter is alive you can watch the messages it is sending by subscribing to the topics the ESP unit is transmitting. Assuming the demo ESP program was used, which sends topics 'ESP178/PVWh' and 'ESP178/gridWh', then at the command line of the Raspberry Pi, type

```
mosquitto_sub -v -t 'ESP178/PVWh'
```

Mosquitto should then start listing the messages on the screen every minute. CTRL ^C will exit.

To see both message streams, open another command line window and type

```
mosquitto_sub -v -t 'ESP178/gridWh'
```

## Setting up MYSQL

This is a database server, which will organise the storage of the received data and supply answers to queries about stored data.

```
sudo apt-get install mysql-server python-mysqldb
```

You will be asked for a password; this is the one for access as a root user in Mysql. This also installs the Python library to access Mysql databases, needed later.



Test – open it up with.

```
sudo mysql -h localhost -u root -p
```

type in the password and you should see the Mysql '>' prompt. If you are not continuing immediately and want to exit type -

```
quit;
```

## Setting up the Mysql database

The Mysql reference site is <http://dev.mysql.com/doc/>

Now create a database for the recordings, describe a data record, set up a user for normal access, and let the user have control of the database. Note almost all command lines in Mysql end in ';' .

Open Mysql, as before,

```
mysql -h localhost -u root -p
```

Enter the password.

Create a database

```
> CREATE DATABASE PVgriddb;
```

Test –

```
> SHOW DATABASES;
```

Tell Mysql to work on this database

```
> USE PVgriddb;
```

Detail the records to be stored

```
> CREATE TABLE data (timestamp DATETIME, source VARCHAR(20), count INT);
```

Test –

```
> SHOW TABLES
```

You should see 'data' listed and then

```
> DESCRIBE data;
```

To see the details of 'data'.

Create a user called 'bill', for example

```
> CREATE USER 'bill'@'localhost' ;
```

Create a password for the user

```
> SET PASSWORD FOR 'bill'@'localhost'=PASSWORD('password')
```

Give the user permission to work on the database files

```
> GRANT ALL ON PVgriddb.* TO 'bill'@'localhost';
```

Test –

```
> SELECT user from mysql.user;
```

This will list all the users.

So now there is somewhere to store the data and users who can access it.

For other projects you can other databases and associated tables, similarly and give the user the access rights.

## Installing the Python Libraries

To take the correct data from the message server and put it in the appropriate database, a project specific Python script is used. The demo script is written in Python 2.7. To access the MQTT server from Python use the Paho library, downloaded using pip, to load pip type -

```
sudo apt-get install python-pip
```

then

```
pip install paho-mqtt
```

To access the Mysql database, use the Python library downloaded when Mosquito was downloaded above, python-mysql. (Nothing extra to do.)

## Python script

A very basic knowledge of Python will allow you to follow this section. The script **MQTT2mysql.py** takes messages from the MQTT broker, timestamps them and stores them in the appropriate Mysql database.

A large part of this to do with making it run as a daemon is taken from this excellent explanation <<http://blog.scphillips.com/posts/2013/07/getting-a-python-script-to-run-in-the-background-as-a-service-on-boot/>>. This part will also create a log file in /tmp/myservice.log as a daemon does not have a screen to write to.

Assume the Python script **MQTT2mysql.py** is stored in /home/pi/myprogs , create a new directory called myservice,

```
cd /usr/local/bin
sudo mkdir myservice
```

Copy the file over

```
sudo cp ~/myprogs/MQTT2mysql.py myservice/
```

Make it executable

```
sudo chmod 755 MQTT2mysql.py
```

Test it runs

```
sudo ./MQTT2mysql.sh
```

Test - open Python, load the program, run it and check the log file in /tmp/myservice.log.

```
sudo nano /tmp/myservice.log
```

This should contain some information appropriate to what else is running.

As stated above, to be of use this program should run in the background, as a daemon- background process, and should be started like Mysql and Mosquitto are automatically at start-up. To do this needs a special script file so that it behaves correctly, **MQTT2mysql.sh** . This looks complicated, but to use it for another project only a few obvious lines need modified. Note this type of file has to have the correct end of line codes, do not edit it on a windows computer unless you know what this involves, use the Raspberry Pi. Assuming it is in /home/pi/myprogs, copy it to the appropriate system directory

```
sudo cd ~/myprogs
sudo cp MQTT2mysql.sh /etc/init.d
```

Make it executable,

```
cd /etc/init.d
sudo chmod 755 MQTT2mysql.sh
```

Test it by typing

```
sudo ./MQTT2mysql.sh start
sudo ./MQTT2mysql.sh stop
sudo ./MQTT2mysql.sh status
```

These should start the program, stop it and give its status.

This script needs to be registered with the system so it will start automatically, so type

```
sudo update-rc.d MQTT2mysql.sh defaults
```

Reboot and test it is running with

```
sudo ./MQTT2mysql.sh status
```

So now there is a program to continually put received messages in a database.

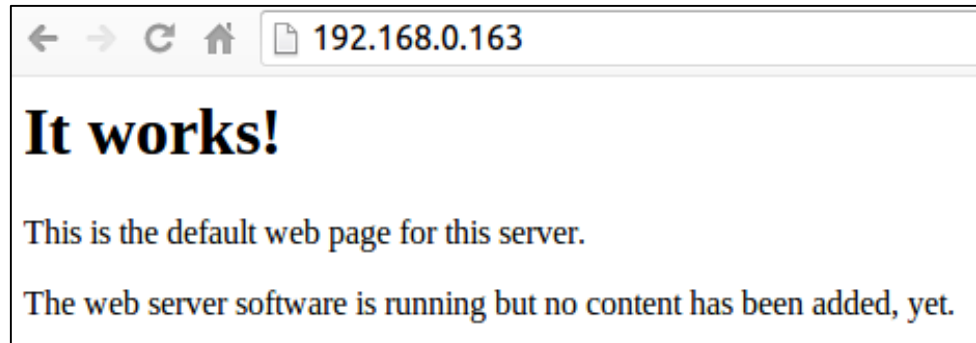
### Setting up the local web server

Apache is a webserver which is used here to allow devices on the local network to look at the collected data. (Apache can be used to display pages over the world wide web, if required , see <https://www.raspberrypi.org/documentation/remote-access/web-server/apache.md> )

Install Apache on the raspberry Pi by typing -

```
sudo apt-get install apache2 -y
```

Test – using another device, open a browser and in the address line, type in the static address of your Raspberry Pi. The default Apache screen should appear -



The files Apache will use to produce the web pages in a specific directory by default in the Raspberry Pi, /var/www in Raspbian Wheezy and in /var/www/html in Raspbian Jessie. In what follows Wheezy is assumed, if Jessie is running use the appropriate directory instead. The owner of this directory is root by default so the normal 'pi' user needs to be allowed too. Carry out these instructions before copying in the project specific files.

Create a "www-data" user and group the owner of the /var/www directory.

```
sudo chown www-data:www-data /var/www
```

Grant the 'www-data' group permission to write to this directory.

```
sudo chmod 775 /var/www
```

Then add the 'pi' user to the 'www-data' group.

```
sudo usermod -a -G www-data pi
```

Reboot the Raspberry Pi

```
sudo reboot
```

Test – create a test file in the web directory

```
cd /var/www
```

Delete the existing index page

```
rm index.html
```

Create a new blank one

```
touch index.html
```

If all this works without having to invoke 'sudo', 'pi' has the appropriate permissions.

To create web scripts like the following example, PHP is required and links for Apache and Mysql. Graphing results is carried out by another great open source program, Gnuplot. There are reference sites for these programs at <http://php.net/> and <http://gnuplot.info/> respectively.

To install them type

```
sudo apt-get install php5 libapache2-mod-php5 -y
```

```
sudo apt-get install php5-mysql
```

```
sudo apt-get install gnuplot -x11
```

Test PHP with

```
nano index.php
```

Add PHP code

```
<?php echo "Hello world"; ?>
```

Exit and save with CTRL^X.

Open a browser and point it at the Raspberry Pi static IP address and "Hello world" should appear.

### Installing the web server files

Everything is now in place for the web server project specific files.

Copy these files into /var/www on Wheezy or /var/www/html on Jessie

**Index.php** – the file that builds the web page

**datapv.plt** – the script to use with gnuplot to plot the PV pulse count data

**datagrid.plt** -the script to use with gnuplot to plot the grid pulse count data

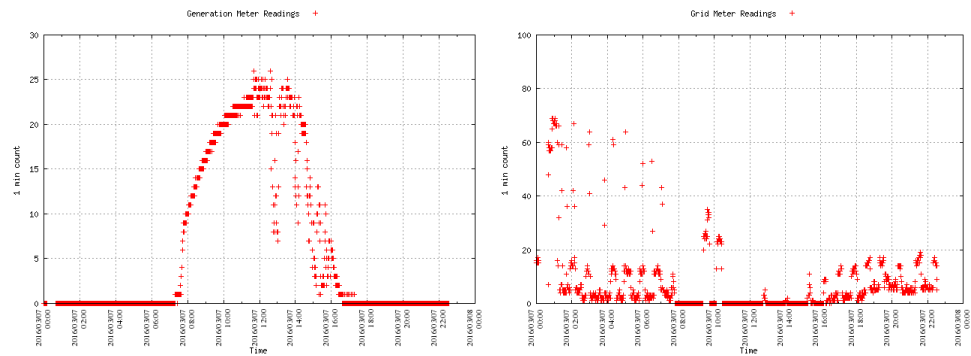
The web page is built up by

- extracting the current day's data from the Mysql database
- store this data in a temporary file
- calling Gnuplot to take the data and produce a temporary graphic image file according to the instructions in the .plt file
- loading these graphics files into the web page

For this to work there needs to be some data in the database, so have the IOT unit running and the Python script **MQTT2mysql.py** running for a few minutes first.

Final test – in a browser go to the Raspberry Pi page, type 192.168.0.150 , or the static IP address you have set on the Raspberry Pi into a browser. It may be a little

slow depending on the vintage of the Raspberry Pi. But you should see something like this.



## Implementing other IOT projects

To implement other IOT projects, the ESP units need to publish topics to the MQTT server. A suitable database needs to be created with its associated tables and the normal 'pi' user granted permissions over these. The Python script needs to be modified to subscribe to the required topics and to put the data in the correct database. The PHP and Gnuplot scripts need to be changed to pick up the correct data. The tediously slow process of installing and testing all the various servers on the Raspberry Pi does not need to be repeated, so additional projects can be up and running quite quickly. The next time most of the effort goes on the fun ESP part of the project.

## Future Development

Clearly a lot more can be done with the data than just graphing it. I am interested in using the PV data and weather forecasts to predict the solar output for the next hour or so to allow me to switch on and off loads in the house maximising my use of the PV energy. I plan to use ESP IOT controllers to adjust the loads around the house. The ESP01 is great for this. There are many good such projects out there e.g. <http://telecnatron.com/articles/ESP8266-WIFI-Remote-Controlled-Mains-Switch/index.html> . I cooked my own as my first ESP8266 project.

Note the IOT device can also receive messages from the MQTT broker, to carry out instructions or receive time signals. The system described is not restricted to information flow in one direction.

The Python script can be tidied up so that the topics and database can be passed as command line parameters to save rewriting this script for each project. Given the length of the project, some links may have changed and you may hit problems I didn't. <http://www.esp8266.com/> is great for queries on everything ESP related. Numerous sites deal with the Raspberry Pi, the official forum is a good place to start <https://www.raspberrypi.org/forums/> . I will try to answer questions, but I am by no way an expert on the ESP8266, in Linux or the packages used above.