Flowcode with MySQL databases & MQTT

Flowcode with MySql Databases & MQTT

Contents

Flowcode with MySql Databases & MQTT

Introduction

This document is written as a follow up to this post https://www.flowcode.co.uk/forums/viewtopic.php?f=4&t=2201 to show how easy it is by using Flowcode to update your own database(s) with values.

In addition, many may be uncomfortable using a third-party "cloud" hosting services for various reasons so having the ability to host your own server and database may be advantageous. Depending on how you configure things, your server can gather data from devices on your own private LAN or from anywhere in the world via the internet. It should also be possible to have both Host and Client on the same device, such as a Raspberry Pi. The Flowcode in this example has been written and tested for an ESP32-WROOM on the same LAN as the server.

First things first, this is not a tutorial on using MySql, PHP, HTTP or HTML, rather a guide, and is to demonstrate the ease Flowcode makes updating or interacting with the database. This is a forum for Flowcode, not for LAMP/WAMP installs so please post questions relating to such in a more suitable place. The code discussed in this document is available in the associated Flowcode v10 forum message

LAMP/WAMP (a prerequisite)

If you don't understand the MySql / PHP commands used here or are a complete novice with LAMP / WAMP installs there are many books and tutorials available. For the complete beginner I can recommend "PHP & MySql in Easy Steps" by Mike McGrath. Although the book is old, it is still a good reference and I used it as guide when I created my first database that Flowcode could update. Also, there are many free online tutorials available such as from "w3schools".

Incidentally my installation is still the same as that book documented and although the versions I'm using are not the most up to date they work fine. I'm running PHP v5.5.9, MySql v5.6.16 and Abyss X1 Server on a W10 machine and all MySql / PHPcode that follows is written for these versions.

** NOTE: If you are using later versions please see UPDATE on page 12**

If you are a complete novice then I recommend you use the versions above as they have been tested to work as documented. They can be found here

https://aprelium.com/abyssws/download.php
https://www.php.net/releases/
https://downloads.mysql.com/archives/community/

You will need a configured LAMP / WAMP installation, a plain-text editor such as Notepad++ which is a free download, and it is assumed you are familiar with Command Line Interfaces. If you are using newer versions of PHP/MySql then the code may or may not work as typed, but an online resource will most likely explain why and give alternative examples. Newer versions may use differing syntax.

If you are a Guru then please feel free to provide better code (with explanations) than the examples here, as I don't claim anything other than basic knowledge and I'm sure there are indeed better ways.

Flowcode with MySql Databases & MQTT

Briefly, MySql is a very popular database application used globally. It does not come with any fancy User Interfaces but you could create your own front end to suit your particular need. Your server application is there to accept and process HTTP requests and PHP provides the interaction with MySql. There is a third-party interface called phpadmin which simplifies the MySQL interaction, but you really need to secure your installation if you intend to use.

If you wish to visit a webpage, your browser basically then sends the following:
GET /URL HTTP/version where URL = the resource you wish to connect with.

Flowcode allows you to easily create these "GET" requests.

We will first look at creating a database, then the PHP code to update it, and once that is running we will look at creating a Flowcode chart that will allow your microcontroller based devices to store their readings directly in your database by calling the PHP script hosted on the server.

Creating your Database

So you plan on having sensors or such like deployed and want to collect, collate and store the data from them. First you need to plan your database. Ultimately, this is the most important part of the process as here you need to establish not just its name, what tables and columns it contains, but also security and validation. It isn't the purpose of this guide to advise on such matters and no security/validation is included. You however should strongly consider such.

Let's say you have some sensors that are gathering values for you. These could be anything gathering whatever. You may have multiple instances of the same type of sensor or even multiple types of sensors with each type providing very different sets to the others.

You will need a way to hold each sensor and the value(s) they gather. It would be nice to timestamp each reading and you might also consider assigning a Primary-Key to uniquely identify each record created too.

A Primary Key is a unique value that can be used to identify a Row (record) in your database.

Suppose you are deploying two different types of sensors, then you could assign a table to each with each table having fields appropriate to the values you wish to record.

You might decide on:-

Database = my_db	The name of the database holding ALL information
Table(s) = sensor_a and sensor_b	Individual tables within, holding specific information

With the table for sensor_a being

ID	Timestamp	Sensor	Value
1	15-11-2023 16:00:00	1	17
2	15-11-2023 16:05:24	2	27
3	15-11-2023 16:08:14	1	24

In the above table:-

- ID is the Primary Key providing a unique reference for each new record (record number)
- Timestamp is added when the record is created
- Sensor an alphanumeric representing the name of a device
- Value being the actual data sent

As MySql can automatically insert the information in the first two columns for us, our sensors using Flowcode only need to send the sensor name and value (key-pairs).

Note you can define your own data types for yourself, this is only an example.

Creating the Database

We will create the database, add a User with Privileges then create the above table. Open the MySql Command Line Interface. Useful keys when working in the CLI include

F3 – insert last line entered (paste what you previously typed)

Up /Down – scrolls through the previously typed entries

From the MySql Command Line Interface prompt (mysql>) enter

mysql> CREATE DATABASE IF NOT EXISTS my_db ;

(The above line creates a database called my_db)

Add User(s) and Privileges (see UPDATE on page 12 if using latest versions)

mysql> GRANT SELECT, INSERT, UPDATE ON my_db.* TO 'user'@'localhost' IDENTIFIED BY '1234' ;

(The above line creates a username on my_db of user with a password of 1234)

To use the new database you created you first need to tell MySql to use it

mysql> USE my_db ;

That's it. You now have a new database called my_db

You can check by running the following commands

mysql>SHOW DATABASES ;

mysql>SHOW GRANTS FOR 'user'@'localhost' ;

Next we will create the table to hold our information.

Creating the Table (see UPDATE on page 12 if using latest versions)

To create the sensor_a table documented above we will issue the following command.

mysql> CREATE TABLE IF NOT EXISTS sensor_a (ID INT AUTO_INCREMENT PRIMARY KEY, Timestamp TIMESTAMP, Sensor VARCHAR(10), Value INT) ;

where:-

- ID = name of 1st Column
- INT = numeric data type for 1st column
- AUTO_INCREMENT = automatically increment 1st column upon creation
- PRIMARY KEY = unique reference for each new record
- Timestamp = name of 2nd column
- TIMESTAMP = automatically inserted date/time in 2nd column based on server time settings
- Sensor = name of 3rd column
- VARCHAR(10) = Variable length string (10 characters maximum) for 3rd column
- Value = name of 4th column
- INT = numeric data type for 4th column

You can check the above by issuing the following command

mysql>EXPLAIN sensor_a ;

That's it. You have now created a database and table to hold your readings. You could create a table for sensor_b in a similar fashion.

Next we need to create scripts to link everything together.

Creating a PHP script to interact with my_db

Before PHP can interact it needs to connect with your database and this is taken care of by a script. This script will connect using the user/password created above. However as this script contains login information it should not be stored in the same folder as your HTML documents. If using Abyss as your web server the default folder for your HTML is C:\Abyss Web Server\htdocs so perhaps use C:\Abyss Web Server\

Using a plain text editor such as Notepad++ create a new document and enter the following (note it is recommended you type all the following code examples directly as pasting can introduce issues):-

```
<?php
$dbc = @mysqli_connect
( ‘localhost’ , ‘user’ , ‘1234’ , ‘my_db’ )
OR die
( mysqli_connect_error() ) ;
mysqli_set_charset( $dbc , ‘utf8’ ) ;
?>
```

Save the above in the parent directory as connect.php

If using Abyss this will be C:\Abyss Web Server\

We will call this script from within another, whenever we wish to interact with the my_db database.

Next we will create the script to update the actual table

Creating a PHP script to Update Database my_db Table sensor_a

Using Notepad++ or similar create a new document and enter the following:-

```
<?php

# Script to update table sensor_a using GET and only allow GET method.

if ( $_SERVER[ 'REQUEST_METHOD' ] == 'GET' )

{

# Connect to MySql Database my_db.

require( '../connect.php' ) ;

# Create temporary variables from SuperGlobals

$Sensor = mysqli_real_escape_string( $dbc, $_GET['Sensor']);

$Value = mysqli_real_escape_string( $dbc, $_GET['Value']);

# Create and execute MySQL query to insert new record into database "my_db", table "sensor_a".

# Query =

$qry = "INSERT INTO sensor_a ( Sensor , Value ) VALUES ( '$Sensor' , $Value ) " ;

# Execute Query

mysqli_query( $dbc , $qry ) ;

# Close connection.

mysqli_close( $dbc ) ;

}

# Job Done = Enjoy a well earned beer <s>

?>
```

Save the above in C:\Abyss Web Server\htdocs folder as update_sensor_a.php

Next we will test the above scripts.

Testing the above Script

Useful CLI commands for MySql include

SHOW TABLES ;	this shows a list of tables in the current database
EXPLAIN <table-name> ;	this provides details of all columns in the table
SELECT * FROM <table-name> ;	this shows all records in the table

To test what we have done above open a browser and type the following into the address bar.

localhost/update_sensor_a.php?Sensor=1&Value=17 <enter>

(alternatively: 127.0.0.1/update_sensor_a.php?Sensor=1&Value=17 <enter>)

Where localhost/ = host address to connect with

update_sensor_a.php = script we wish to run

Sensor=1 = key-pair to update Sensor column with the value of 1

Value=17 = key-pair to update Value column with the value of 17

Now at your MySql Command Line Interface enter the following

mysql>SELECT * FROM sensor_a ;

If all went well you should now have one line in your table containing ID number / Timestamp / Sensor number / Value with Sensor and Value being 1 and 17 respectively.

Have a play using your browser. Send sensor and value key-pairs and also familiarise yourself with CLI commands. However please do not change anything to do with the table structure yet as the next section will deal with updating from a microcontroller.

If the above does NOT work there is no point in going further until it does. Check your typing as that will most likely be the problem. Also if not using Abyss as your server you will need to establish where to store the “connect” and “update” files (Apache stores them in /var/www and /var/www/html). If using later software versions then perhaps seek online help to update code.

Please note that we have not addressed security, validation, error handling or even anything in the way of returns. These are all things you should consider and there are many examples of such out there.

Before we move on we need to take note of the server’s IP address. Although we used localhost or 127.0.0.1 as the address in our browser, these are just local to your machine and cannot be used to reach it from elsewhere on your LAN. In Windows you can see the address you are using from Windows Command Prompt by entering ipconfig which will return your network details. Take note as we will need them later (e.g. IP Address = 192.168.0.123). It is worth noting that your server requires a static IP address on your LAN so that the internal IP address does not change.

Creating a Flowcode Chart to update the Database

Using Flowcode v10 we will create a chart to update the database we created above (my_db / sensor_a).

We will need a way to connect to the same LAN as your server and Flowcode provides components to enable this from WiFi enabled targets such as the ESP32, to external boards such as the ESP8266.

The example given here is written using Flowcode v10 for the free ESP32-WROOM target. With Flowcode though, it is very easy to choose a different chip as it is target independent.

Rather than use an actual sensor in this example, we will instead just include a User Macro that generates a random integer value. You could instead insert a routine to obtain whatever data you wish.

Briefly, we will

- Initialise components
- Connect to WiFi
- Enter a loop
- Obtain sensor data
- Convert data into String values
- Create a String to send that will include the script to use and key-pairs
- Obtain the length of above String
- Connect to the server
- Tell server the length of String we are sending
- Send String
- Close the connection
- Wait before repeating

In the attached Flowcode chart you will need to first provide the following information for the variables listed below:

Variable	**Value**
SSID	Enter your WiFi SSID
Pword	Enter your password for above
Server	Enter the IP address of your server (e.g. 192.168.0.123). Note you can also use URLs
Script	Name of script to be run (e.g. update_sensor_a.php)
Sensor	Unique name for your sensor/device. For each Sensor, Flowcode should allocate a unique Sensor reference to differentiate between them. This can be alphanumeric.

Note: Script and Sensor have preconfigured values in the example but you should modify to suit.

When we tested our database and scripts above, we used a browser to send:-

Localhost/update_sensor_a.php?Sensor=1&Value=17

Now we will use Flowcode to connect with the server then create and send the following string:-

"GET /update_sensor_a.php?Sensor=*Sensor*&Value=*Value* HTTP/1.1\r\nHost: *server*\r\n\r\n"

Where

- GET = is the request method
- update_sensor_a.php = is the script we will call
- Sensor=*Sensor* = is the first key-pair value
- Value=*Value* = is the second key-pair value
- HTTP/1.1\r\nHost: *Server*\r\n\r\n = is required HTTP

Note that we can send as many key-pairs as required, they are just separated by an "&" but be sure to increase the size of the "Send" string to accommodate.

With reference to the attached chart Update_my_db you can see we only need two components to update the database

WLAN ESP32	Component Libraries>Comms>WLAN (ESP32) (2D)
Network Comms	Component Libraries>Comms>Network Communications (2D)

If you look in Network Comms Properties you will see we linked it to the ESP32 and here you can also set your simulation details if required (if set correctly then you can simulate the chart from within Flowcode without the need of compiling to target. This is an excellent feature).

The first comment in the chart is a reminder to set the required variables. In the example the variable Script has been preconfigured to update_sensor_a.php and the variable Sensor to FC1 you can change these to suit.

We then initialise the components before attempting to join your WiFi network. We will keep looping until we successfully connect as defined by the Ret value of "1" before entering the Main Loop of the program (note if simulating you can enter "1" in the Simulation Debugger for variable "Ret" which will allow your chart to progress).

We then branch to a User Macro (Get_Data) in which we will gather whatever information is required to be sent to the database.

In the example we will just use a calculation box to generate a random number, but this could be anything such as obtaining an ADC reading or sampling some sensor.

Importantly, whatever you gather must be converted into a String before we can send, and so we do this conversion before exiting the Macro.

Now back in the Main chart we then build our string to send. Flowcode occasionally has issues when concatenating multiple strings at once so we will create ours step by step and once created we will then calculate the length of the string.

The chart then opens a socket and attempts to connect with the server on port 80. Note that although Flowcode allows you to test for success in these actions we have not incorporated here and you should consider such in any deployment.

We include a short delay to allow time for the server to respond. Depending on latency this could be many hundreds of milliseconds and delays between 200 and 450mS are not unreasonable. If everything is on your own LAN then these can be drastically reduced.

Next we send our string with another short delay before closing our socket.

The chart then has a delay of 30s before repeating but you could perhaps have your chip sleep or such like.

Before compiling to target remember to go to Project Options and set the Programmer Port your ESP is connected to.

Once running (or simulating), again issuing the following command at your MySql CLI should show updated values

mysql>SELECT * FROM sensor_a ;

The above is only intended as a guide to getting things working and I hope it is of some help. Any actual deployment should consider the points made above regarding security, validation, errors and tests.

Thanks go to forum contributor RGV250 for his help in the creation and testing of this guide.

Flowcode with MySql Databases & MQTT

UPDATE if using latest MySql and PHP

To use later versions of MySql and PHP

1) If updating from previous versions, uninstall all previous versions and run a registry cleaner
2) Do a clean install

There are a few differences between versions, but we are only concerned with these steps.

After we create the database my_db we have to use it

mysql>USE my_db ;

We now create a new user by entering the following

mysql>CREATE USER 'user' IDENTIFIED BY '1234' ;

(The above line creates a username of user with a password of 1234)

On page 4 when creating the table sensor_a use the following command

mysql> CREATE TABLE IF NOT EXISTS sensor_a (ID INT AUTO_INCREMENT PRIMARY KEY, Timestamp DATETIME DEFAULT CURRENT_TIMESTAMP, Sensor VARCHAR(10), Value INT) ;

We can then grant privileges by issuing this command

mysql>GRANT SELECT, INSERT, UPDATE ON sensor_a TO 'user' ;

(The above line assigns privileges to user on table sensor_a)

The remaining parts of the guide should work as documented but if you experience any issues then online resources should assist.

Retrieving results from the database

Now that we can populate the database it would be good if we could retrieve values from it too. Again note that we have not addressed security, validation or error handling. These are all things you should consider and there are many examples of such out there

In the previous examples we have a sensor called FC1 that updates a table called sensor_a within the my_db database. If you have been playing, your database should contain many differently named sensors and associated values. If not, use your browser to add in a few more values for FC1 and also add some differently named sensors.

Once you have added more data, make sure the last few updates are not named FC1. If you then display the table contents (mysql>SELECT * FROM Sensor_a;) you will see a list of entries with sensor FC1 appearing up from the bottom.

Let's decide that we want the last value entered by a sensor, as we may want to act on the most recent value. Remember that every addition to the database is automatically timestamped so that makes things easier for us.

To get the latest value entered for our sensor FC1 we need to search the table for all entries containing FC1, then arrange them by descending time extracting the latest update before sending the value back. Let's say we want to return the sensor name and value such as sensor=xxxx

We can do this with a PHP script that will accept our GET request, interact with the database to extract the information and send it to us.

As before we will send a key-pair in a GET statement but instead of updating the database it will retrieve information

Script to retrieve database values

Using Notepad++ create a new document and enter the following

```
<?php
# Script to request latest sensor value table sensor_a using GET and only allow GET method.
if ( $_SERVER[ 'REQUEST_METHOD' ] == 'GET' )
{
# Connect to MySql Database my_db.
require( '../connect.php' ) ;
# Create temporary variables from SuperGlobals
$Sensor = mysqli_real_escape_string( $dbc, $_GET['Sensor']);
# Create and execute MySQL query to retrieve record from database "my_db", table "sensor_a"
# Query =
$q = "SELECT Value FROM sensor_a WHERE Sensor = '$Sensor' ORDER BY Timestamp DESC LIMIT 1" ;
# Execute Query
$result = mysqli_query( $dbc , $q ) ;
if (mysqli_num_rows($result) > 0) {
    // OUTPUT DATA OF EACH ROW
    while($row = mysqli_fetch_assoc($result)) {
       echo $Sensor, "=",$row["Value"] ;
    }
 } else {
    echo "0 results";
 }
# Close connection.
mysqli_close( $dbc ) ;
}
?>
```

Save the above in htdocs as request_sensor_a.php

Testing the above Script

To the test the above script open your browser and enter

Localhost/request_sensor_a.php?Sensor=FC1

This should return

FC1=xxxx where xxxx is the value.

Again using your browser now enter

Localhost/update_sensor_a.php?Sensor=FC1&Value=12345

This creates a new entry for sensor FC1 with a value of 12345

If we again run

Localhost/request_sensor_a.php?Sensor=FC1

This should return

FC1=12345

If it doesn't then double check your new script for spelling mistakes and errors. Once you have it working have a play. Update your database then retrieve values for different sensors.

Flowcode with MySql Databases & MQTT

Creating a Flowcode Chart to retrieve values from the Database

This is just an example to illustrate how you *could* receive and process a request using Flowcode. The example is written for an ESP32-Lolin board which has an onboard LED connected to pin 22. We will poll the database at intervals and if the returned value exceeds a predefined "threshold" value then the LED will illuminate. If the value is below, then the LED will extinguish.

Briefly, as before we will

- Initialise components
- Connect to WiFi
- Enter a loop
- Create a String to send that will include the script to use and a key-pair
- Obtain the length of above String
- Connect to the server
- Tell server length of String we are sending
- Send String

However in addition to the above we will then

- Receive reply from Server
- Close the connection
- Parse the reply obtaining our data
- Do something with the data
- Wait before repeating

Although when we use the browser to make the request we get the reply FC1=12345 (sensor name and value), in reality far more HTML is sent by the server. In the attached chart we capture the response in a variable called Rx_String and we use the variable Rx_length to obtain the length of the string, which is in excess of 200 bytes.

We will need to parse this string to obtain our data and there are many ways to do this. Again this is just an example and we will use a Circular Buffer to "LookForValue(s)" and also a loop to extract values.

First we should check that we connected and received a page by looking for "200 OK" in our received string and if so continue. Note this just indicates we successfully connected and obtained "some" reply from the server.

Next we should confirm that we have our expected Sensor in the reply and if so we can then loop extracting our value. This will still be in string format so we will also convert to an integer value.

In the chart we include a LED that only comes on if the received value exceeds a predefined Threshold value. In the example it will come on if the returned value of FC1 is greater than 17. You can modify to suit.

In the attached Flowcode chart you will need to first provide the following information for the variables listed below:

Variable	Value
SSID	Enter your WiFi SSID
Pword	Enter your password for above
Server	Enter the IP address of your server (e.g. 192.168.1.123). Note you can also use URLs
Script	Name of script to be run (e.g. request_sensor_a.php)
Sensor	Unique name for your sensor/device. For each Sensor, Flowcode should allocate a unique Sensor reference to differentiate between them. This can be alphanumeric.
Threshold	Trigger level to “do something”. This can be any value you wish

Note: Script, Sensor and Threshold have preconfigured values in the example but you should modify to suit.

If simulating then in the Simulation Debugger you can also view the variables as the chart progresses.

Testing the Flowcode Chart

As mentioned previously, we have a Threshold value for FC1 and if this is exceeded the LED will come on.

Using a browser enter the following

Localhost/update_sensor_a.php?Sensor=FC1&Value=10

This creates a new entry for sensor FC1 with a value of 10.

Localhost/request_sensor_a.php?Sensor=FC1

This should return FC1=10

Connect / run your Flowcode chart and wait at least 30 seconds (or however long your loop delay is). All being well nothing visible should happen, the LED should remain OFF

Now, using a browser enter the following

Localhost/update_sensor_a.php?Sensor=FC1&Value=27

This creates a new entry for sensor FC1 with a value of 27. The next time the Flowchart retrieves a value for FC1 the LED should turn ON. It will stay on until the retrieved value for FC1 is less than (or equal to) the Threshold (17).

You can test this by using your browser (or FC powered device) to update the database with values for FC1 above and below the Threshold.

Flowcode with MySql Databases & MQTT

Update to incorporate JSON encoded values

So far we have used GET to make requests and return values and this is quite easy to implement in our embedded devices allowing even modest microcontrollers to connect with remote servers and exchange information.

Flowcode includes JSON (JavaScript Object Notation) Encode and Decode components which allow us to send and receive data in an open format independent to any programming language.

In reality it makes little difference in these examples if we use JSON or not, as we control what happens “server side” but you may need to interact with a server that requires JSON.

A JSON string contains either an array of values, or an object (an associative array of name/value pairs)

To make a request using GET, our data was sent in pairs such as Sensor=FC1 but in JSON, although still being sent as a pair the format is slightly different. Each element in our pair is enclosed by double-quotes and separated by a colon, and each pair separated by a comma. We then enclose our request in curly brackets.

Now we have our JSON as {"Sensor":"FC1"}

Note that Flowcode automatically converts to {\"Sensor\":\"FC1\"} for transmission.

At the server (running PHP) we need to receive this request and decode.

The received request will be URL encoded and look like this {\%22Sensor\%22:\%22FC1\%22}

So we use urldecode to give {\"Sensor\":\"FC1\"}

We then remove the backslashes to give {"Sensor":"FC1"} which is our JSON

Decoding the JSON provides us with an array of Array ([Sensor] => FC1)

We can then extract our values Sensor = FC1 and use in our request to obtain our reply (e.g. 27)

To send back the reply, we first need to assign each element as an “object” then encode.

Note that in this example, although we are sending JSON the server is only using it to make a database request, not update.

Flowcode with MySql Databases & MQTT

Using Notepad++ create the following document and save as request_json.php

```
<?php
# Only allow GET method.
if ( $_SERVER[ 'REQUEST_METHOD' ] == 'GET' )
{
# Connect to MySql Database my_db.
require( '../connect.php' ) ;

#Parse the incoming HTTP to obtain Sensor value to query.
$parse1 = $_SERVER['QUERY_STRING'] ; // obtain the HTTP string we sent
$parse2 = urldecode($parse1); // decode string
$parse3 = stripslashes($parse2) ;  // remove backslashes from string
$jsn = json_decode($parse3, true) ; // Decode the json into an array
$Sensor = $jsn['Sensor']; // assign sent value to variable

# Create and execute MySQL query to retrieve record from database "my_db", table "sensor_a"
$q = "SELECT Value FROM sensor_a WHERE Sensor = '$Sensor' ORDER BY Timestamp DESC LIMIT 1" ;

# Execute Query
$result = mysqli_query( $dbc , $q ) ;

if (mysqli_num_rows($result) > 0) {
    // OUTPUT DATA OF EACH ROW
    while($row = mysqli_fetch_assoc($result)) {
            $my_obj = new stdClass();
            $my_obj -> Sensor = $Sensor ;
            $my_obj -> Value = $row["Value"] ;
            $myJSON = json_encode($my_obj);
            echo $myJSON ;
      # echo $Sensor, "=",$row["Value"] ;
    }
 } else {
    echo "0 results";
 }

# Close connection.
mysqli_close( $dbc ) ;
}
# Job Done = Enjoy a well earned beer <s>
?>
```

Testing the above Script

To the test the above script open your browser and enter

Localhost/request_json.php? {\"Sensor\":\"FC1\"}

This should return

{"Sensor":"FC1","Value":"xx"}

Where xx = latest value for sensor FC1

Again using your browser now enter

Localhost/update_sensor_a.php?Sensor=FC1&Value=12345

This creates a new entry for sensor FC1 with a value of 12345

If we again enter

Localhost/request_json.php? {\"Sensor\":\"FC1\"}

This should return

{"Sensor":"FC1","Value":"12345"}

If it doesn't then double check your new script for spelling mistakes and errors. Once you have it working have a play. Update your database then retrieve values for different sensors.

Flowcode with MySql Databases & MQTT

Creating a Flowcode Chart to retrieve values from the Database using JSON

This is just an example to illustrate how you *could* send and receive and a JSON encoded request using Flowcode. The example is written for a Raspberry Pi with an LED connected to Port G4. We will poll the database at intervals and if the returned value exceeds a predefined "threshold" value then the LED will illuminate. If the value is below, then the LED will extinguish.

There isn't too much difference when we create our request but decoding using the Flowcode JSON component makes things very easy for us compared to before.

Briefly, as before we will

- Initialise components
- Enter a loop
- Create a String to send that will include the script to use and a JSON encoded key-pair
- Obtain the length of above String
- Connect to the server
- Tell server length of String we are sending
- Send String

Again we will then

- Receive reply from Server
- Close the connection
- Decode the JSON
- Do something with the data
- Wait before repeating

We will create a JSON string to send using the Flowcode JSON Encoder component. I find it easy to think on what we will be sending as an array containing pairs (name and data), with each pair being individually represented in Flowcode as element name and element value/data. In Flowcode each element pair is assigned a value starting from zero.

First Element pair

Sensor	FC1
(Element (0) Name)	(Element (0) Data)

Element pairs

Sensor = FC1	Sensor = FC2
(Element (0) Name and Element (0) Data)	(Element (1) Name and Element (1) Data)

We will use the JSON Encoder component to assign our values and once complete generate our JSON as a string. Once we have our JSON encoded values we will include in our Send string as before.

All being well the server will accept our request and return a JSON encoded reply. Flowcode makes it easy to process the received string.

First we use ParseJSON command to check the reply contains key-pairs and if not we exit.

We can then use further JSON Decoder commands to obtain our sensor name and value, checking if valid.

In the chart we include a LED that only comes on if the received value exceeds a predefined threshold value. In the example it will come on if the returned value of FC1 is greater than 17. You can modify to suit.

In the attached Flowcode chart (written for a Raspberry Pi) you will need to first provide the following information for the variables listed below:

Variable	**Value**
Server	Enter the IP address of your server (e.g. 192.168.1.123). Note you can also use URLs
Script	Name of script to be run (e.g. request_json.php)
Sensor	Unique name for your sensor/device. For each Sensor, Flowcode should allocate a unique Sensor reference to differentiate between them. This can be alphanumeric
Threshold	Trigger level to “do something”. This can be any value you wish

Note: Script, Sensor and Threshold have preconfigured values in the example but you can modify to suit.

If simulating, then in the Simulation Debugger you can also view the variables as the chart progresses.

Testing the Flowcode Chart

As mentioned previously, we have a Threshold value for FC1 and if this is exceeded the LED will come on.

Using a browser enter the following (assuming you still have the previous scripts)

Localhost/update_sensor_a.php?Sensor=FC1&Value=10

This creates a new entry for sensor FC1 with a value of 10.

Localhost/request_sensor_a.php?Sensor=FC1

This should return FC1=10

Connect / run your Flowcode chart and wait at least 30 seconds (or however long your loop delay is). All being well nothing visible should happen, the LED should remain OFF

Now using a browser enter the following

Localhost/update_sensor_a.php?Sensor=FC1&Value=27

This creates a new entry for sensor FC1 with a value of 27. The next time the Flowchart retrieves a value for FC1 the LED should turn ON. It will stay on until the retrieved value for FC1 is less than (or equal to) the Threshold (17).

Flowcode with MySql Databases & MQTT

Installing RAMP (Raspberry Pi, Apache, Mariadb, PHP)

The Raspberry Pi SBC has its own community dedicated to all things “Pi” and is thoroughly documented on the internet. I am certainly far from being in any way a “RPi” expert (nor linux) so the following is just an example of how it *could* be done and is not intended to be a tutorial on any of the required applications. If you need further help then an internet search will probably solve. As before, you are strongly recommended to consider security which is not discussed here.

There are many versions of the RPi available and I used the popular 3B running the latest OS (Bookworm, at time of writing). I found that the Desktop version was slow to run on my 3B so used the Headless version, accessing via SSH.

When you create your OS image, RPi Imager gives you the option to set name / password / hostname / WiFi credentials and to enable SSH.

Being headless you need to know the IP address of the RPi to login. The easiest way is simply to login to your router and see what address has been allocated to it.

It is a good idea to have a static IP on your RPi so you can easily locate it. Again from the web it appears that setting the RPi to use static can create issues, therefore I kept the RPI using DHCP and instead Bound an address to the RPi MAC within my router.

Once connected, to ensure you are fully updated you should run

```
sudo apt update
sudo apt upgrade
```

The RPi needs to be awake 24/7 and it is recommended to disable “Screen Saving” despite being headless, and remove any unnecessary USB devices (info obtained from internet sources).

First we will install the Apache server. Previously we used Abyss, which is available for linux systems, however Apache is by far the most popular.

Login to your RPi via SSH

```
sudo apt install apache2 –y
```

Will install the server application. You can test this by pointing a browser to your RPi’s address and you should receive a webpage displaying the Apache Debian Default page.

Apache stores it's webfiles in this location /var/www/html and we need to set permissions to modify.

We will add our current username to the default Apache group (www-data group) and bestow rights over /var/www/html.

```
sudo usermod -a -G www-data $USER
sudo chown -R -f www-data:www-data /var/www/html
sudo chmod -R 770 /var/www/html
sudo reboot
```

The above will make and apply the necessary changes. Note that Apache uses virtual hosts to manage multiple sites which we will not go into here.

Next, we need to install PHP and this can be done by issuing the following

```
sudo apt install php8.2 libapache2-mod-php8.2 php8.2-mbstring php8.2-mysql php8.2-curl php8.2-gd php8.2-zip –y
```

To test that it is working we will create a PHP file called testindex.php within /var/www/html and access via a browser.

```
Sudo nano testindex.php /var/www/html
```

This will open up the nano editor and we will populate the new file with

```
<?
echo "Hello World" ;
?>
```

(CTRL+O then CTRL+X will save and close).

Test by pointing your browser to http://<ip-address>/testindex.php

Now we will install MySQL. Although it is possible to install the actual MySQL on the RPi it is recommended to use MariaDB which is more or less identical and is the default for RPi. If interested an internet search will give the entwined history. Note that although we will be using MariaDB we will still refer to it as MySQL

As always it is recommended to first ensure everything is up to date by issuing

```
sudo apt update
sudo apt upgrade
```

Flowcode with MySql Databases & MQTT

Install MariaDB by issuing

```
sudo apt install mariadb-server
```

To enable PHP to talk to Mysql we need to install the connector by issuing

```
sudo apt install php-mysql
```

By default MySQL doesn't have any password enabled so we will secure by issuing

```
sudo mysql_secure_installation
```

Note that there is no set password for root (yet) so just press ENTER. You will be able to set a password as you go through.

To login to MySQL use the following command

```
sudo mysql –u root –p
```

This will request the password you just set in the previous step and once logged in you can create and manage databases as before.

Now as before on our WAMP install, we will create a database called my_db

```
CREATE DATABASE my_db;
```

Then we will assign a user and rights

```
CREATE USER 'user'@'localhost' IDENTIFIED BY '1234';
GRANT ALL PRIVILEGES ON my_db.* TO 'user'@'localhost';
```

Once done we need to flush the privileges table to allow the new user we created access

```
FLUSH PRIVILEGES;
```

The next step is to create a table called sensor_a which is done by issuing

```
USE my_db ;
```

This switches us to our newly created my_db database

```
CREATE TABLE IF NOT EXISTS sensor_a ( ID INT AUTO_INCREMENT PRIMARY KEY, Timestamp
DATETIME DEFAULT CURRENT_TIMESTAMP, Sensor VARCHAR(10), Value INT ) ;
```

As before we need to create PHP scripts to access and update.

Creating the PHP scripts

Creating a PHP script to interact with my_db

Much like our previous scripts, we will create one to “connect” to our database and we will store this in a parent directory /var/www. We will then create a script(s) that calls on this when interacting with the database.

This script will contain the login details of our my_db database and it is not visible in the HTML folder, having to be called from within other scripts.

We create it by invoking “nano” an inbuilt editor.

```
sudo nano connect.php /var/www
```

This will open up the editor and we can insert the following

```
<?php

// Set variables
$servername = "localhost";
$database = "my_db";
$username = "username";
$password = "password";

// Create connection
 $conn = mysqli_connect($servername, $username, $password, $database);

// Check connection
if (!$conn)
  {    die("Connection failed: " . mysqli_connect_error());    }

// Set character set
mysqli_set_charset( $conn, "utf8" ) ;
?>
```

CTRL+O then CTRL+X will save and close

Creating a PHP script to Update Database my_db Table sensor_a

Now that we have our "connector" we can incorporate this into the script that updates the database. Again we will use nano to create the script and this time it will be saved in /var/www/html to make it visible.

```
sudo nano update.php /var/www/html
```

This will open up the editor and we can then insert

```
<?php

// Script to only allow GET method and call our connector.

if ( $_SERVER[ 'REQUEST_METHOD' ] == 'GET' )
 { require( '/var/www/connect.php' ) ; }
else
 {  die ; }

// Get  variables from request

 $Sensor = mysqli_real_escape_string( $conn, $_GET['Sensor']);
 $Value = mysqli_real_escape_string( $conn, $_GET['Value']);

// Create query and insert values into table

$qry = "INSERT INTO sensor_a (sensor, value) VALUES ('$Sensor', '$Value')";

if (mysqli_query($conn, $qry))
 {  echo "OK";  }
else
 {  echo "Error: " . $qry . "<br>" . mysqli_error($conn); }
mysqli_close($conn);

?>
```

CTRL+O then CTRL+X will save and close the editor.

Testing the Scripts

To test the above scripts we can use a browser to send values. Open a browser and enter the following

http://<ip-address>/update.php?Sensor=x&Value=y (where x and y are your variables)

If all is well you should receive “OK” in your browser.

If you then login to your database

```
sudo mysql –u root –p
```

```
USE my_db;
```

Then issue the following

```
SELECT * FROM sensor_a ;
```

It will return the records in your table and you should see your latest addition.

The above is just an example of how you could implement a RPi database and it should be a simple matter to re-use the Flowcode previously created and the other scripts to interact.

Outside Access

Until now we have been accessing our databases from within our own LAN. However you may wish to access from outside your network. Please remember that in doing so you should implement security measures that we have so far omitted. It isn't the intent of this document to advise on such but you should give serious consideration to it.

Dynamic Addressing

Unless you have a static IP, and most home users don't, you will need a way for your server to be found outside your own LAN. Typically, your external IP address will change from time to time and is not under your control. Dynamic addressing solves this problem.

If you subscribe to a Dynamic DNS service (No-IP and DynDNS being two examples), then they assign a unique web address to you (you do get a choice) that you then use as your address (e.g. www.myaddress.mydynamic123.net or such like). A client on your internal network, or your router itself automatically updates your dynamic service provider with your current external IP address allowing your dynamic service provider to redirect traffic to you.

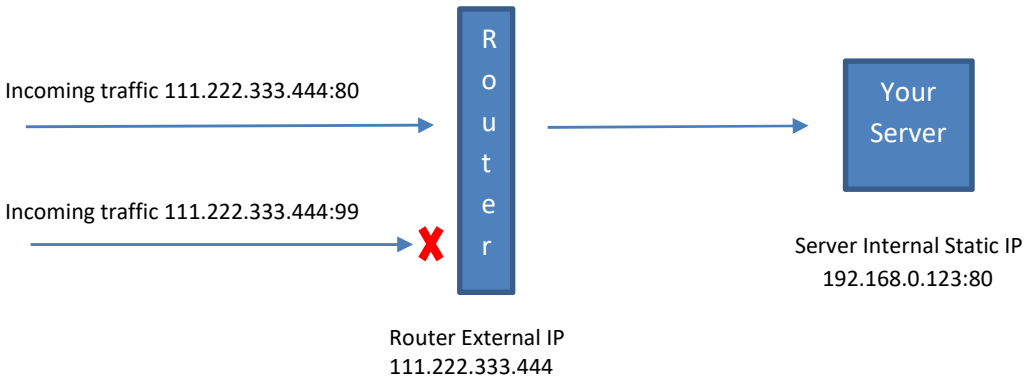
This works well for hosting services that are accessed via a browser (e.g a website) but if you are writing code for a microcontroller it is a little more complicated. When a browser arrives at your dynamic address, the service sends a redirect back to the browser containing your actual physical IP address contained in the returned Headers. This redirect is seamless for a browser, but your microcontroller will need additional code to identify and respond.

Port Redirection / NAT

Using a Static IP address (or Dynamic), traffic arrives at your router and we need some way to send that to your server within your LAN. Each protocol/service uses different Ports and you will need to identify which Port you are using. For example, websites/servers typically use Ports 80 and 443 with 8080 and 8443 as alternatives (e.g. aaa.bbb.ccc.ddd:80). Your server on your LAN will (or should) have its own internal static address and Port Redirection ties the two together.

There are many articles explaining this on the internet and the Flowcode WiKi also provides guidance. Each router will be different so it is better you search for details specific to yours.

Briefly though, with Port Redirection / NAT you are telling your router that incoming traffic originating from the internet on Port-x, should go to your server's internal address Port-x. Only traffic on the specified Port will be routed to your server.



Flowcode with MySql Databases & MQTT

Installing a MQTT Broker on a Raspberry Pi

Creating your own MQTT broker can be very advantageous, especially if you wish to create your own "smart home", and it should only take a few minutes get up and running. The Flowcode WiKi gives a good guide to installing, and the following are the steps I took.

Login to your RPi and issue the following commands to first ensure everything is up to date.

```
sudo apt update
sudo apt upgrade
```

Once updated we will install both mosquitto and client application by entering

```
sudo apt install mosquitto mosquitto-clients
```

To ensure it is running enter

```
sudo systemctl enable mosquitto
```

Then check by entering

```
sudo systemctl status mosquitto
```

All being well you should see "active (running)" in the reply. Next we will setup access authorisation. Enter the following with your chosen username.

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd <username>
```

This will then prompt you to set a password for the above username.

Now we will force mosquitto to use authentication and allow remote access by modifying the configuration file. The following command will allow us to use nano to edit the file.

```
sudo nano /etc/mosquitto/mosquitto.conf
```

Move the cursor down and add the following lines to end of the file

```
listener 1883
allow_anonymous false
password_file /etc/mosquitto/passwd
```

CTRL+0 then CTRL+X will save and close the file.

To apply the changes we then restart the mosquitto broker

```
sudo systemctl restart mosquitto
```

Next we will test that it is all working.

Testing your MQTT installation

To test over my LAN, I used both CLI and a free client called MQTTX available for Windows, Mac and linux which is quite user friendly. MQTTX, once set up with credentials allows you to Publish to a Broker and Subscribe too. Other such clients are available and you will need to provide the following to allow such clients to connect. Note that your client may differ but the basics are here.

Name:	Give your connection a name, for example Pi-MQTT
Host:	The target IP address (or URL if over internet)
Port:	1883 which is the standard port number
Client ID:	Unique ID to identify your client (e.g. MQTTX-1). Note some services will provide you with an ID to use.
Username:	Username you set previously
Password:	Password you set previously

With the above details entered I can then Connect, create / enter a Topic and Publish messages. As the client is also Subscribing, whatever I Publish appears almost instantaneously in the Subscribe pane.

With MQTTX connected, from my command prompt I published a test message by entering:-

mosquitto_pub -h <address> -t test/message -m Hello -u <username> -P <password>

where:-

-h = Hostname / IP address of Broker

t = Topic

-m = Message to be sent

-u = Username you set previously

-P = Password you set previously

Again almost instantaneously the message appeared on MQTTX

In the command prompt, to Subscribe I then entered

mosquitto_sub -t test/message -u <username> -P <password>

This starts the client to listen for messages under the Topic test/message.

Publishing from MQTTX saw the message appear at the client.

All being well you now have a functional MQTT Broker, and next we will create a Flowcode chart to interact.

Flowcode with MySql Databases & MQTT

Creating a Flowcode chart to interact with MQTT Broker

The Flowcode WiKi page is very informative and contains examples of using the component that you may wish to explore. I decided to create my own Subscribe and Publish charts specific to my need as documented below. It isn't anything fancy but hopefully easy to follow.

Subscribe Chart

With reference to the chart posted alongside this in the forum, I am using a Raspberry Pi as target but others such as ESP32 should work equally well. The chart Subscribes to the Topic "test/LED" on my Broker and looks for a message (payload) containing either On or Off, action taken upon content.

Components I used are RPi TCP/IP, Network Comms, MQTT Client, LCD and a LED, although you can easily adapt to your own need.

First we need to set a few parameters

- Set your Pi WiFi access if being used.
- If simulating set Network Interface in Network Comms component
- In MQTT component set:-
    - Host = IP address or URL
    - Client ID = unique identifier
    - Name = your username
    - Password = your password
- Set variable "Topic" to the topic you wish to Subscribe to. This example use "test/LED"

The chart first initialises components then tries to connect with the Broker. If successful it then proceeds to Subscribe to the chosen Topic, then loops with a short delay awaiting incoming messages.

When a message arrives, the chart branches and tests to see if the Topic is correct. To do this we use the compare$ function to ensure that the Topic matches that which we specified (it should as we are only subscribing to a specific Topic) and if so returns 0.

We then test the message (payload) for our specific content and if found branches to action. If message = On we switch the LED on, and if message = Off we switch it off. Simple stuff indeed.

I didn't need to download to a RPi as Flowcode allows simulation, so using that instead and using MQTTX to create a new topic, test/LED, I then sent some test messages.

The chart LED actioned with each On / Off sent, and as expected ignored all else.

Next is a chart to Publish

Publish Chart

The next chart is to Publish values to the Broker and is based on the previous chart. All components are the same but this time instead of Subscribing to Topic “test/LED” it will Publish to Topic “test/message”. Of course you can choose any Topic you wish.

In the chart I call a User Macro called Get_Data and in here you could have a routine to gather data from a sensor or whatever. I just assign a random value and convert to a string.

Once connected, it displays a status message then loops. In the loop I branch to the macro before publishing then printing the message on the display.

The chart is very simple and should be easy to follow.

Again I ran in Simulation, and every few seconds it published a random value to the Broker. With MQTTX connected and Subscribing, these messages automatically appeared in the Received pane.

My Publish and Subscribe charts are really quite basic and are just to show how easy Flowcode makes interaction with MQTT.