

Online HTTP Information Server and SMS Alerts System for Mobile Weather Station through GSM Wireless Communication



IWMI's Yann Chemin (left) and Niroshan Bandara test a locally-developed weather station that can be connected to the mobile phone network.
(PHOTO Neil Palmer/ IWMI).

Lahiru Wijesinghe

All the information in this document is available in the public domain

Introduction

This System is designed as an add-on to the previously developed Mobile Weather Stations. The purpose of the system is to alert the key personals about the rainfall readings above threshold values and update an information server with set intervals.

As the Mobile Weather Stations were built on Arduino open source platform, this system uses an Arduino compatible GSM shield for the SMS and GPRS wireless communication. The program functions are written to work with almost every type of GSM shields on the local and international market including SIM900 and SM5100B shields. These functions uses standard AT commands to communicate with the GSM Module. The program code can be easily configured to change with different service providers.

A simple HTTP server runs under Windows web development environment is used as the test server to receive and display the weather information which sends from the Mobile Weather Stations. GSM modules attached to Mobile Weather Stations send weather information using HTTP POST method. All the information received from Weather Stations is stored in a My SQL database and then displayed on a HTTP web page.

SMS alerts are sent to specific people whose numbers are programmed in the system. The format of the SMS and the threshold values which needs to overcome to send a SMS can be programmed according to the user requirements.

Table of Contents

Sending Weather Information to the Server through a GSM Module Connected to Arduino Board.	3
STEP 1: Downloading and Installing Arduino IDE	3
STEP 2: Interfacing the GSM Module to Arduino	5
STEP 3: Configure Arduino IDE to connect with the Arduino board	7
STEP 4: Type the main C code and include header files	9
STEP 5: Compile and Download the code to Arduino	12
Creating a Simple Web Server to Receive and Display Weather Information	14
STEP 1 – Setting up Web Development Environment. (WampServer for Windows.)	14
STEP 2 – Creating required HTML and PHP files	16
STEP 3 – Creating a database and a table to store the weather information	20
STEP 4 – Launching and testing the server and the database	22

Appendix

Codes used in this Manual	28
---------------------------	----

Sending Weather Information to the Server through a GSM Module Connected to Arduino Board.
STEP 1: Downloading and Installing Arduino IDE

1.1. Download the latest stable Arduino version depending on the operating system (Windows, Mac OS or Linux) from the following Link. (For Windows download the Windows Installer.)

Link: <https://www.arduino.cc/en/Main/Software>

Contribute to the Arduino Software

Consider supporting the Arduino Software by contributing to its development. (US tax payers, please note this contribution is not tax deductible). [Learn more](#) on how your contribution will be used.



SINCE MARCH 10TH 2015, THE ARDUINO IDE HAS BEEN DOWNLOADED SO MANY TIMES. IMPRESSIVE! THIS IDE IS NO LONGER JUST FOR ARDUINO BOARDS. HUNDREDS OF COMPANIES AROUND THE WORLD ARE USING IT TO PROGRAM THEIR DEVICES, INCLUDING COMPATIBLES, CLONES, AND EVEN COUNTERFEIT. YOU CAN HELP ACCELERATE THE DEVELOPMENT OF THE ARDUINO IDE BY CONTRIBUTING TOWARDS THE EFFORT OF MAKING IT BETTER.

\$3

\$5

\$10

\$25

\$50



OTHER

JUST DOWNLOAD

CONTRIBUTE & DOWNLOAD

As the Arduino is an open source software there is no need to pay for the download. But a contribution can be made for the development of the software.

Depending on the choice click JUST DOWNLOAD or CONTRIBUTE & DOWNLOAD to start downloading.



Search the Arduino Website

HomeBuyDownloadProductsLearningForumSupportBlog

LOG INSIGN UP

Download the Arduino Software



ARDUINO 1.6.5

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

This software can be used with any Arduino board. Refer to the [Getting Started](#) page for installation instructions.

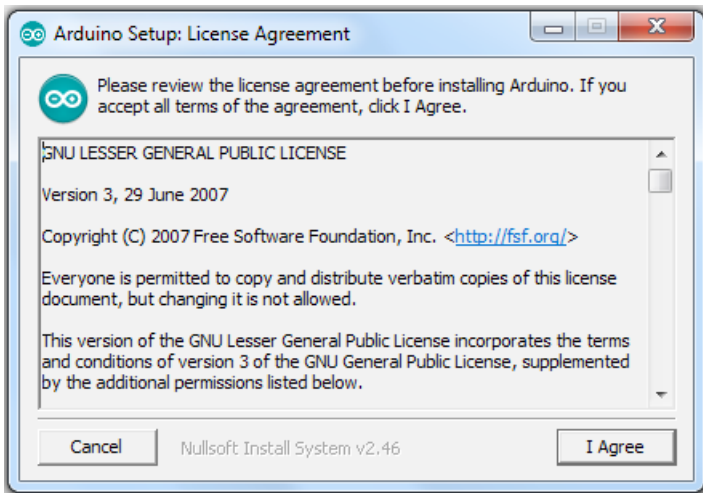
Windows Installer
Windows ZIP file for non admin install

Mac OS X 10.7 Lion or newer

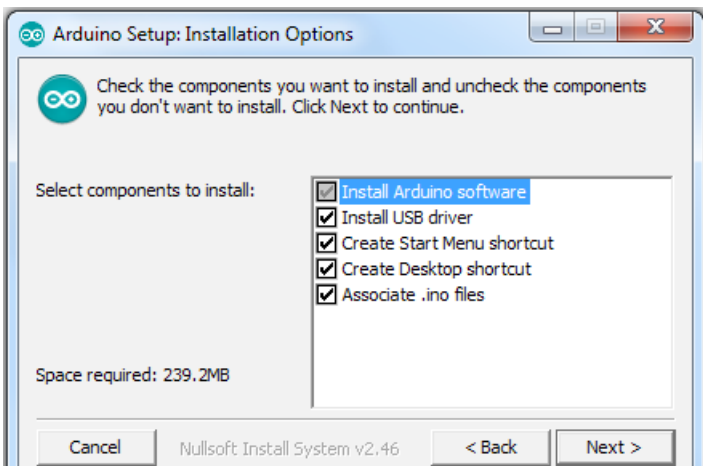
Linux 32 bits
Linux 64 bits

[Release Notes](#)
[Source Code](#)
[Checksums](#)

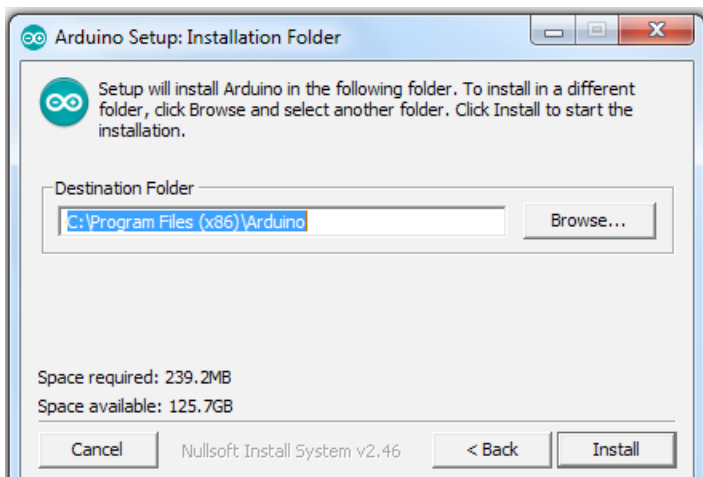
1.2. Run the downloaded Installer and Click I Agree to the License Agreement to continue the installation.



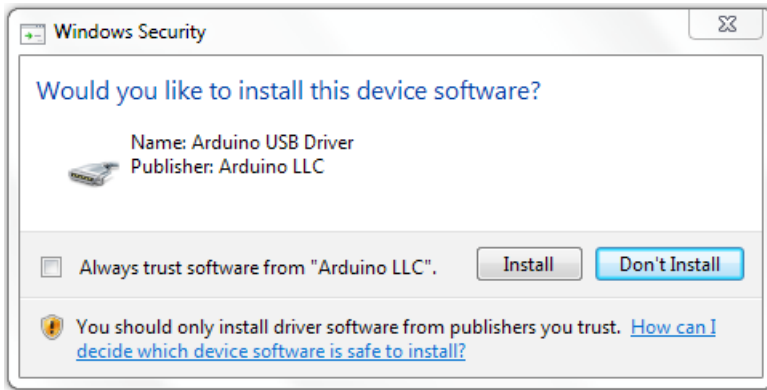
1.3. Select the check boxes to Install USB driver and Associate .ino files. Select the two check boxes to Create Start Menu shortcut and Create Desktop short and click Next.



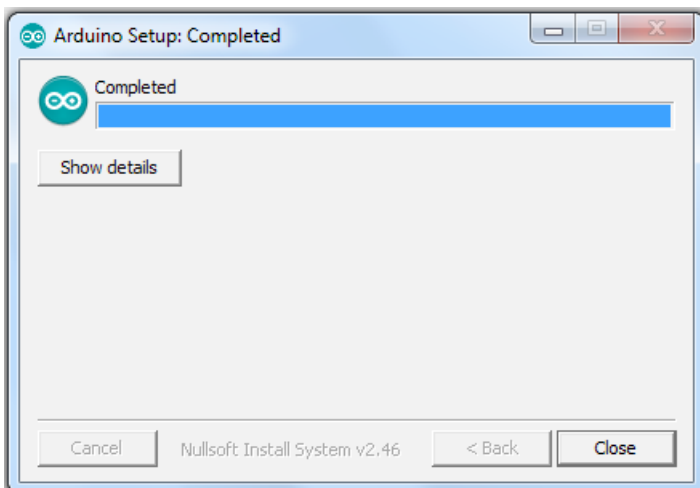
1.4. On the next screen click Install after selecting a destination folder (Default folder: C:\Program Files (x86)\Arduino) for the Arduino installation.



1.5. Click Install when asked to install the Arduino USB Driver.

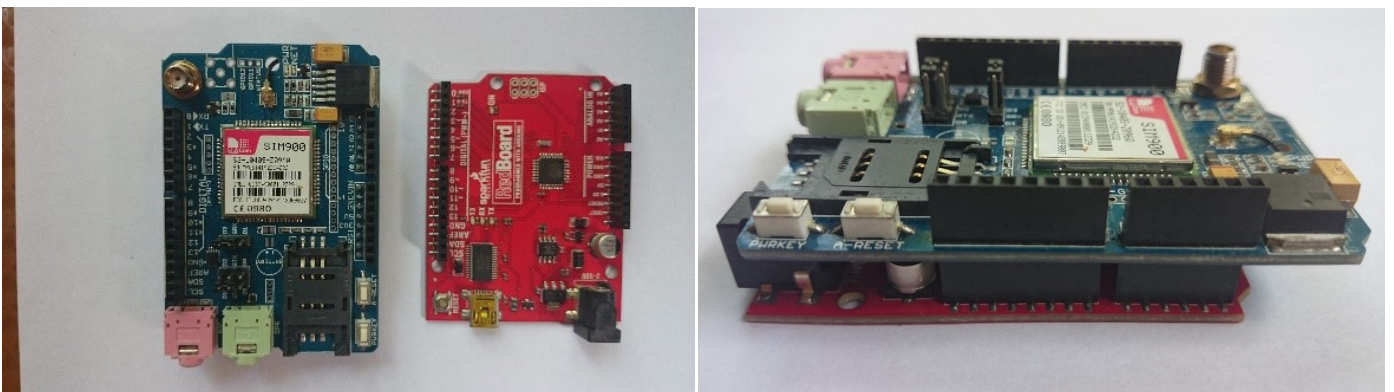


1.6. Once the Installation is complete click Close to exit from the installer.

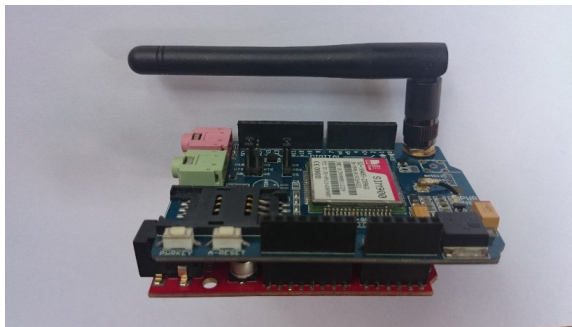
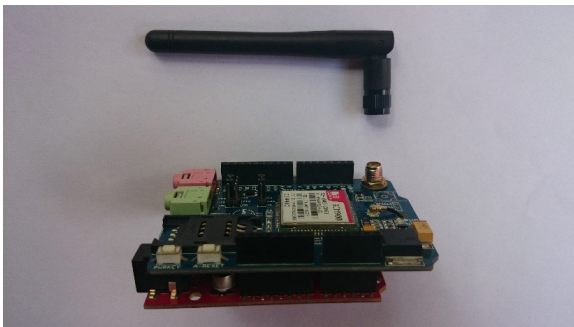


STEP 2: Interfacing the GSM Module to Arduino

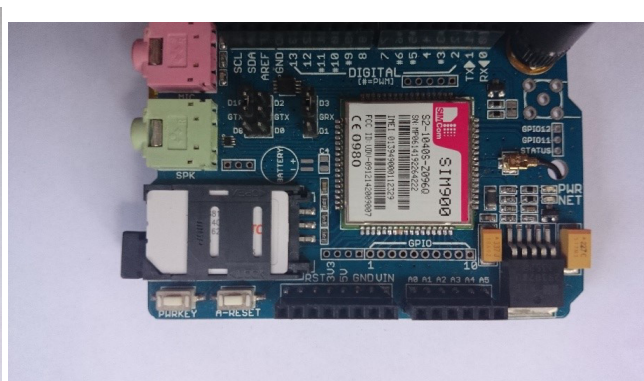
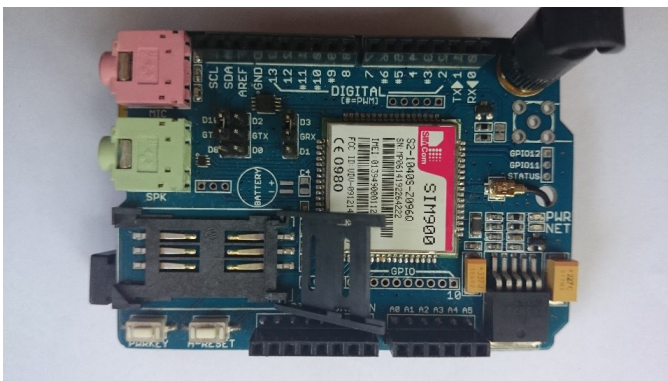
2.1 Attach the GSM shield to the Arduino board as shown in the following figure. Look for the pin names in the GSM shield and the Arduino board and place the shield to match with the board pin names.



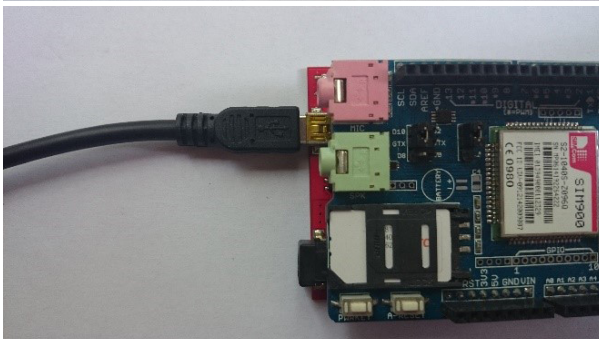
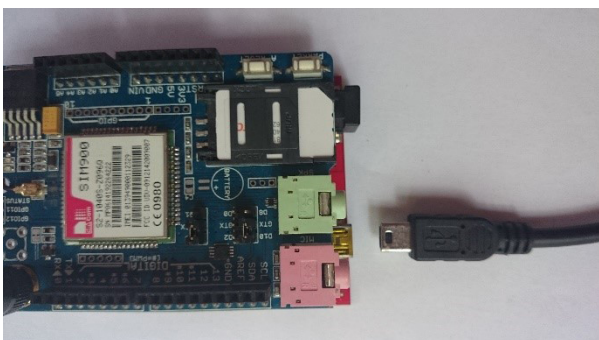
2.2 Plug the antenna to the GSM shield.



2.3 Open the SIM card holder and close the holder placing the SIM card inside.

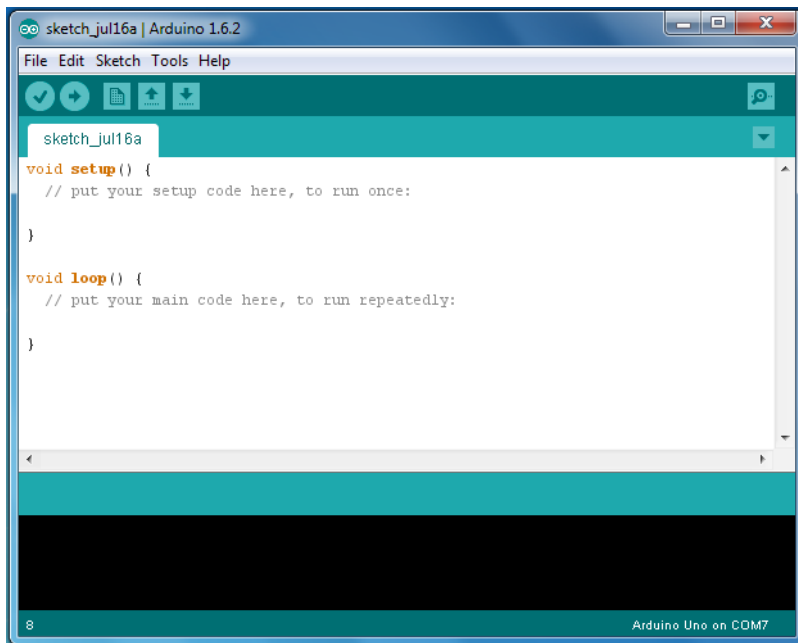


2.4 Connect the USB type B or USB mini port of the USB cable to the Arduino board and the standard USB port side to the PC.

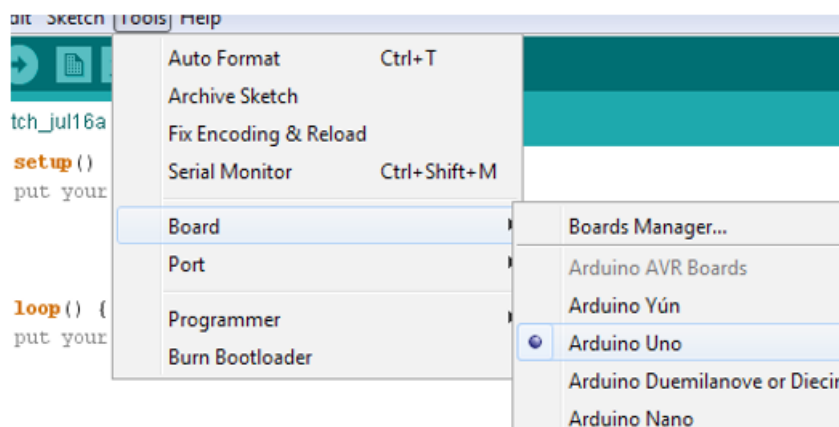


STEP 3: Configure Arduino IDE to connect with the Arduino board.

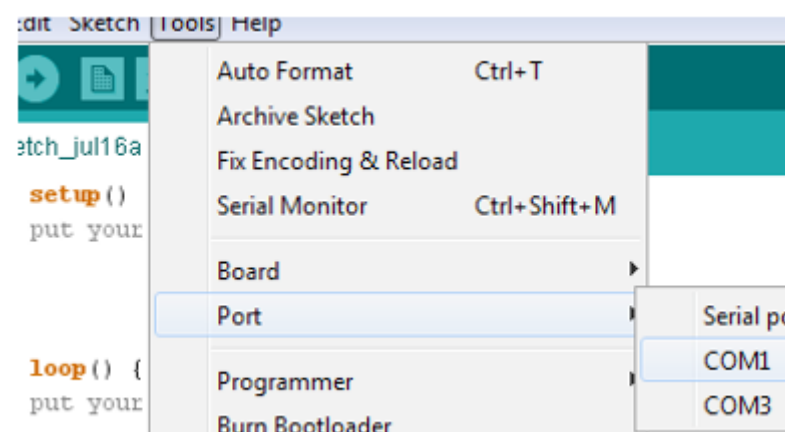
3.1 Run the Arduino software from the shortcut or searching Arduino from the windows search.



3.2 Select the Arduino board by selecting Board in Tools Menu. (Arduino Uno or Arduino Mega)



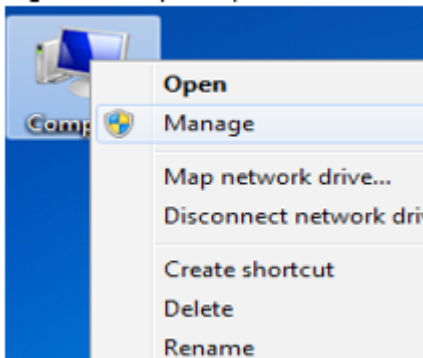
3.3 Select the Serial port which the Arduino board is connected by selecting Port in Tools Menu.



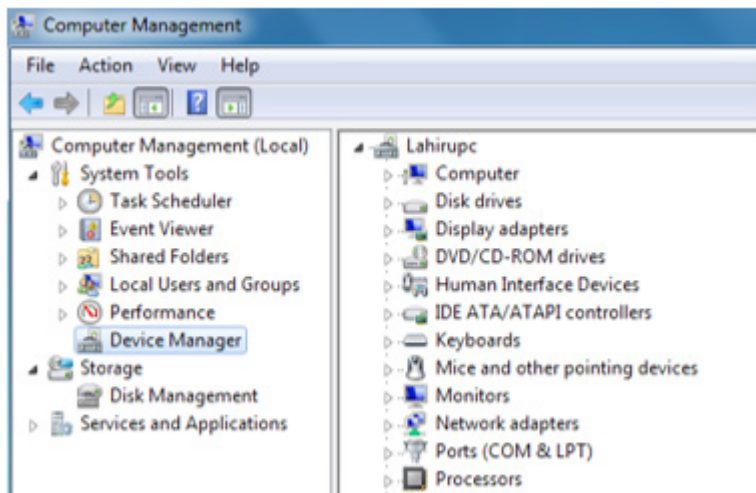
COM Port name assigned for the Arduino USB connection can be found under the Ports category in Device Manager.

To open Device Manager in Windows XP, Vista and 7:

Right click My Computer and then click Manage.

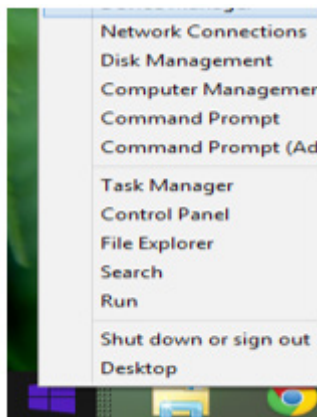


Click on Device Manager under System Tools from the left panel in Computer Management.



To open Device Manager in Windows 8:

Right click on Windows Start Button and click Device Manager.



3.4 If the COM Port of the Arduino Board is not listed under Ports (COM & LPT), the drivers for the specific Arduino board has to install manually. Drivers are located in the drivers folder in Arduino installation directory (C:\Program Files (x86)\Arduino\drivers).

STEP 4: Type the main C code and include header files.

4.1 After the Board and Port are selected successfully, type the test code given in appendix under GSM_Test.ino to test the GSM Module with SMS and GPRS functionality. And edit the code according to the requirement.

Break down of the code:

Including necessary header files.

```
#include <SoftwareSerial.h>
#include "GSM_Module.h"
```

Create a String variable and store the Host Name of the server. Replace the string with the host name or the address of the server.

```
String HOST = "lahiruwije.ddns.net";
```

Create a variable to store the http port number which needs to connect with the server.

```
int PORT = 80;
```

Create a String variable to store the mobile number of the SMS alert receiver. Replace this number with the relevant person's number.

```
String Mobile_No = "+94776141305";
```

Create a String variable to store the content of the SMS alert. Edit the string as necessary.

```
String SMS_Body = "Test SMS";
```

Variable to store the return states from the SMS and GPRS functions.

```
int Status;
```

Variables to store the relevant data which needs to send to the server. Assign the values to these variables by reading the sensors. Read the Mobile Weather Station Manual to understand about the receiving the values from the sensors.

```
int temperature = 42;
int humidity = 20;
```

Create a variable with GSM_Module class. State the module type as the input parameters to the class. (SIM900 or SM5100B, Here the type is selected as SIM900)

```
GSM_Module Module(SIM900);
```

Declaration of the setup function which runs at the beginning of the runtime.

```
void setup(){
```

Initialize the hardware serial port if need to communicate with a PC to observe the status and the errors in connecting with the module. Type the baudrate as the input parameters to the function. (Here the baudrate is set to 9600 which is default for most systems.)

```
Serial.begin(9600);
```

Initialize the module with the given baudrate as the input parameter which needs to communicate with the module and assign the status which returned from the function to the Status variable.

```
Status = Module.Init(9600);
```

Check the status for errors. If the status is OK print "Module Ready" on hardware serial and continue. If the status is not OK print "Module Initializing Failed." with the error code on hardware serial and halt the program.

```
if( Status == OK ){
```

```

        Serial.println("Module Ready.");
    }else{

        Serial.println("Module Initializing Failed.");
        Serial.print("ERROR : ");
        Serial.println(Status);

        while(1){};
    }
}

```

After successful initialization of the module, refresh the module and wait for 5 seconds.

```

Module.Refresh();
delay(5000);

```

Declaration of the loop function which runs after the setup functions within a never ending loop.

```

void loop()

```

Sending a SMS to the phone number which stored in Mobile_No variable with the content stored in SMS_Body variable. Assign the status which returned from the Send_SMS function to the Status variable.

```

Status = Module.Send_SMS(Mobile_No, SMS_Body);

```

Check the status for errors occurred in the function while sending the SMS. If there is no errors and the status is OK print "SMS Sent" on hardware serial. If the status is not OK print the SMS error code on hardware serial.

```

if( Status == OK ){

    Serial.println("SMS Sent");

}else{

    Serial.print("SMS ERROR : ");
    Serial.println(Status);

}

```

Connecting to the server on the host name stored in HOST variable with the port number stored in PORT variable and send the data stored in temperature and humidity variables through POST method. Assign the status which returned from the POST_Data function to the Status variable.

```

Status = Module.POST_Data(HOST, PORT, temperature, humidity);

```

Check the status for errors occurred in the function while connecting to the server and sending the data. If there is no errors and the status is OK print "Connecting Successful" on hardware serial with the last response from the module. If the status is not OK print the TCP error code on hardware serial with the last response from the module.

```

if( Status == OK ){

    Serial.print("Connecting Successful");
    Serial.print(" ");
    Serial.println(Module.Received_String());

}else{

    Serial.print("TCP ERROR Code : ");
    Serial.print(Status);
    Serial.print(" ");
    Serial.println(Module.Received_String());

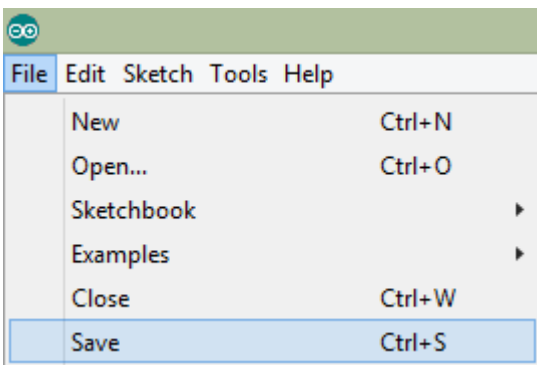
}

```

Listen to the module for any response from server and print them on hardware serial.

```
while(1){  
  
    do{  
  
        Module.Wait(1000);  
  
    }while(!Module.String_Received());  
  
    Serial.println(Module.Received_String());  
  
};
```

4.2 Save the project in an accessible folder by clicking save in File menu.



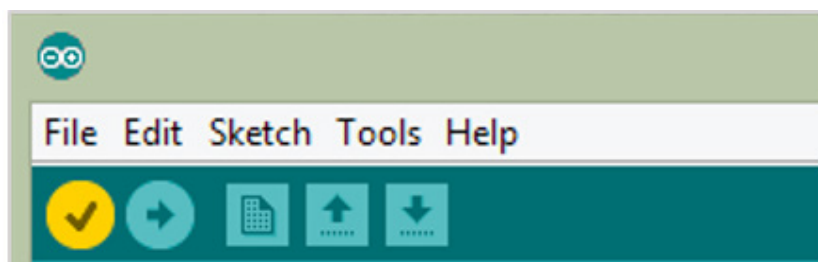
4.3 Create three new text files and save them as GSM_Module.cpp, GSM_Module.h and Module_Codes.h in the same directory where the .ino file saved. Type the c codes given in appendix with above file names in each created file. GSM_Module.cpp file with the header file define the necessary functions and classes needed to communicate with the GSM module. Module_Codes.h file defines the error codes returning from each function and the AT commands.

4.4 Close the project and reopen the project to open the project with included header files.



STEP 5: Compile and Download the code to Arduino.

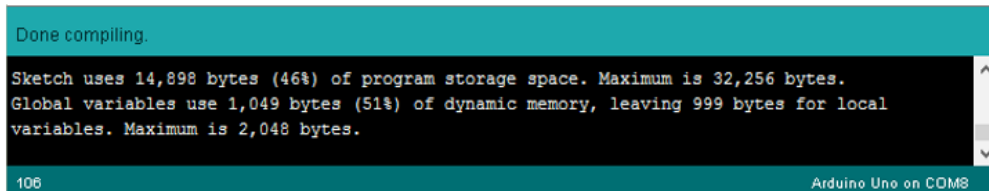
5.1 Compile and verify the code by clicking the verify button below the File menu.



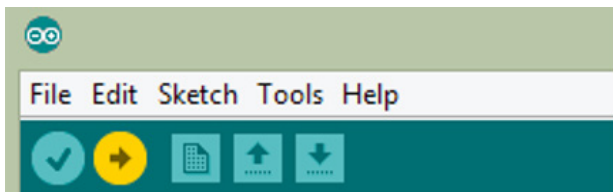
A progress bar at the bottom of the code shows the compilation progress while the code compiles.



Once the compilation is done it shows "Done compiling" with the memory usage of the code in Arduino board.



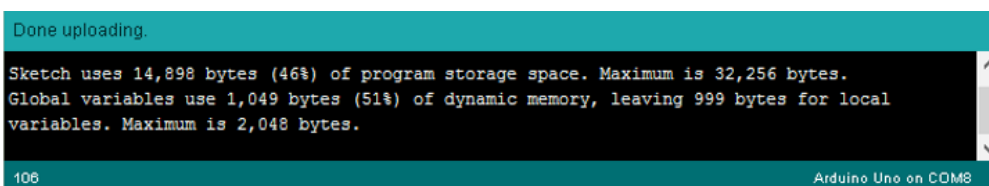
5.2 After the compilation is done upload the code into the Arduino board by clicking upload button next to verify button. The code compiles again before uploading.



A progress bar at the bottom shows the progress of uploadig.



Once the uploading is done it shows "Done uploading".



Creating a Simple Web Server to Receive and Display Weather Information.

STEP 1 – Setting up Web Development Environment. (WampServer for Windows.)

1.1 Download the applicable WampServer Installer file according to the System Type (32bit or 64bit) from the following link.

Link: <http://www.wampserver.com/en/>

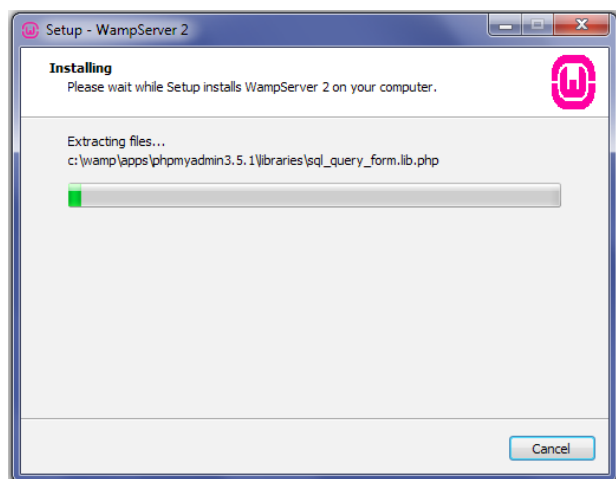
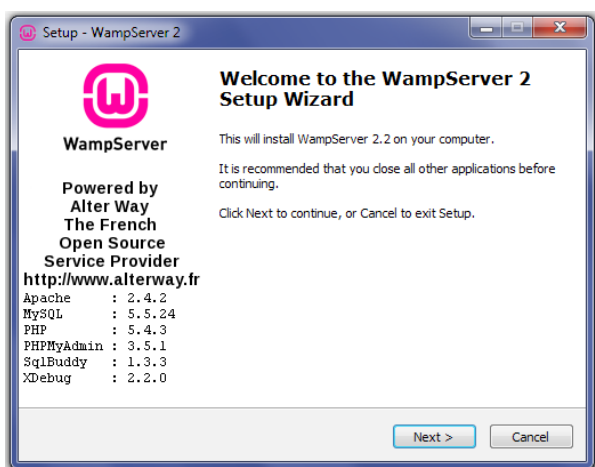


To check the system type, right-click on My Computer, and then click Properties.

System

Processor: Intel(R) Core(TM) i3 CPU M 350 @ 2.27GHz 2.27 GHz
Installed memory (RAM): 2.00 GB (1.86 GB usable)
System type: 32-bit Operating System, x64-based processor
Pen and Touch: No Pen or Touch Input is available for this Display

1.1 Double click on the downloaded file and just follow the instructions until it extract all the files into the computer. The WampServer package is delivered with the latest releases of Apache, MySQL and PHP.



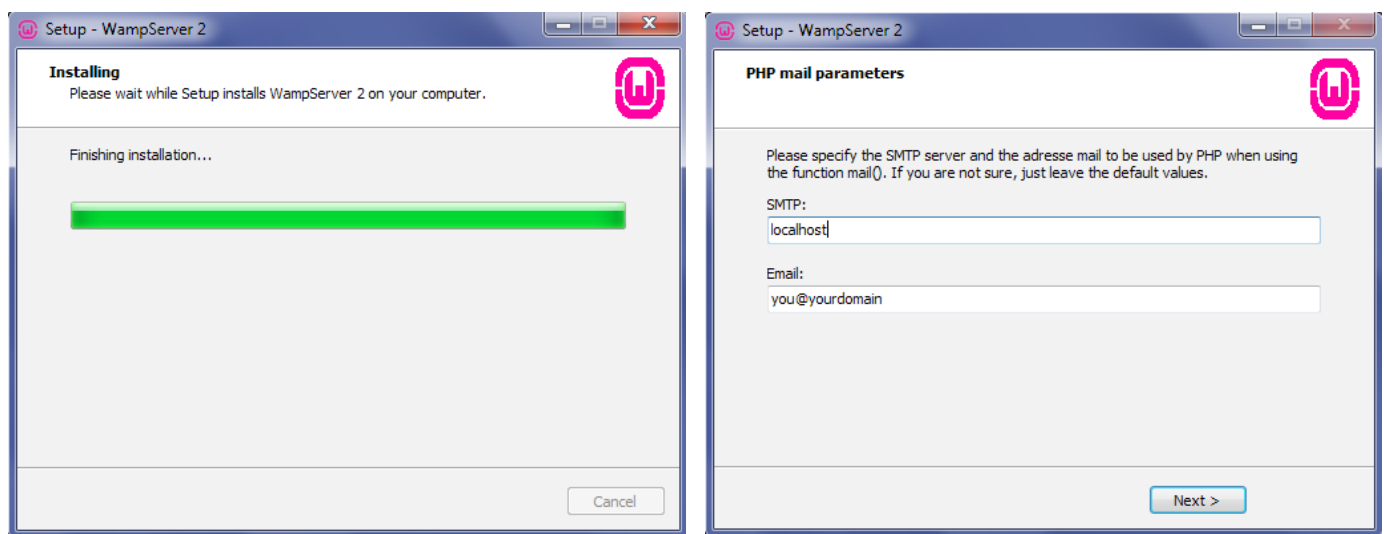
1.2 Once the files are extracted, select the default browser when asked. WampServer defaults to Internet Explorer upon opening the local file browser window. If the default browser isn't IE, then look in the following locations for the corresponding .exe file:

- Opera: C:\Program Files (x86)\Opera\opera.exe
- Firefox: C:\Program Files (x86)\Mozilla Firefox\firefox.exe
- Safari: C:\Program Files (x86)\Safari\safari.exe
- Chrome: C:\Users\xxxxx\AppData\Local\Google\Chrome\Application\chrome.exe

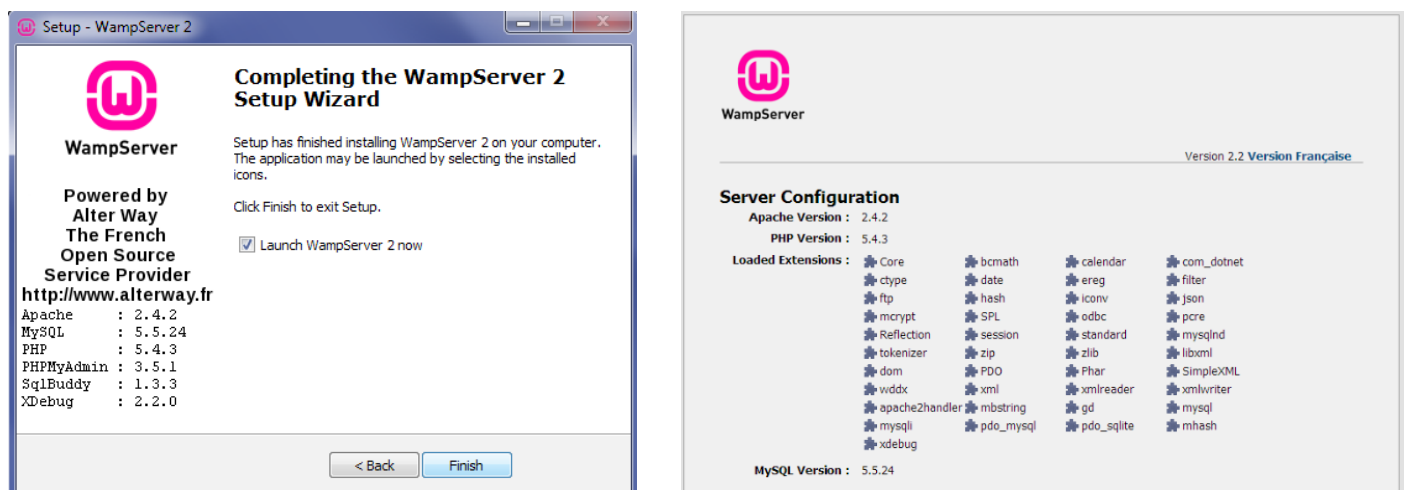
Select the default browser's .exe file, then click Open to continue.

1.3 A Windows Security Alert window will open, saying that Windows Firewall has blocked some features of the program. To allow Apache HTTP Server to communicate on a private or public network, click Allow Access.

1.4 Once the progress bar is completely green, the PHP Mail Parameters screen will appear. Leave the SMTP server as localhost, and change the email address. Click Next to continue.



1.5 The Installation Complete screen will now appear. Check the Launch WampServer Now box, then click Finish to complete the installation. Once you have completed the installation process, test that your installation is working properly by going to <http://localhost/> in your browser. You should see the WampServer homepage displayed.

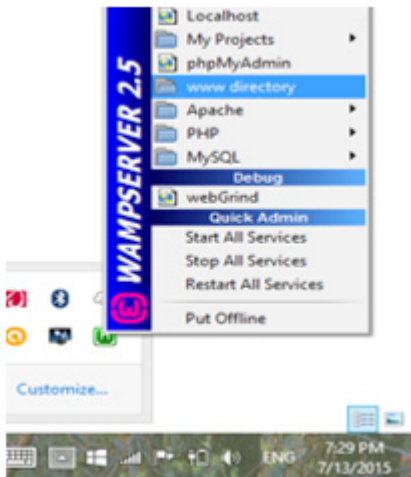


1.6 Download and Install the Visual C++ Redistributable Packages which install runtime components that are required to run C++ applications built with Visual Studio 2012.

Link: <http://www.microsoft.com/en-us/download/details.aspx?id=30679>

STEP 2 – Creating required HTML and PHP files

2.1 Open www directory of the WampServer by clicking WAMPSERVER icon from the windows system tray.



2.2 Cut & paste the files inside the www directory to a new folder. Create a new text file and save it as index.html. Type the html code given in appendix under index.html and edit the code according to the requirement.

Break down of the code:

Title of the page

```
<title>Sensor Data</title>
```

Start of the XML Script

```
<script>
```

XML function definition.

```
function loadXMLDoc()
```

Create an XMLHttpRequest Object:

```
var xmlhttp;
```

For Firefox, Chrome, Opera and Safari browsers.

```
if (window.XMLHttpRequest)
{
    xmlhttp = new XMLHttpRequest();
}
```

For Internet Explorer.

```
else
{
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Handling the onreadystatechange event which is triggered when a request to a server is sent. This performs actions depending on the readyState. The readyState property holds the status of the XMLHttpRequest. If status is OK and the request finished and response is ready this gets the database values with an id of myDiv.

```

xmlhttp.onreadystatechange = function()
{
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200)
        document.getElementById("myDiv").innerHTML=xmlhttp.re-
sponseText;
    }
}

Status = 200      : OK
readyState = 4    : Request finished and response is ready

```

Send a Request to the Server from the GET method. The getdatabase.php is the php code written get the values from the database (This code is explained in the next section).

```

xmlhttp.open("GET","getdatabase.php",true);
xmlhttp.send();

```

Refresh the page by calling LoadXMLDoc function in 500ms intervals.

```

window.setInterval(function(){
    loadXMLDoc();
}, 5000);

```

End of the XML Script

```

</script>

```

Header of the web page

```

<h1> Temperature and Humidity Sensor Readings </h1>

```

Shows the database data which gets from the XML script by the id of myDiv.

```

<div id="myDiv"></div>

```

2.3 Create three new text files and save them as add.php, connect.php and getdatabase.php in the www directory. Type the php codes given in appendix in the each php files created and edit the codes according to the requirement.

Break down of the codes:

add.php

Start of the PHP Script

```

<?php

```

Include the connect.php file to connect with the database

```

include("connect.php");

```

Make a connection with the database and assign the connection to a variable called link.

```

$link = Connection();

```

Create variables to store the values included in the POST request. Two variables are for the temperature and humidity values.


```
$templ = $_POST["templ"];
$huml = $_POST["huml"];
```

Create a mysql query string containing the time and the values included in the POST request with respect to the columns of the mysql table. templog is the name of the table here. And the timestamp, temperature and humidity are the columns of the table.

```
$query = "INSERT INTO `templog` (`timestamp`,`temperature`,`humidity`)
VALUES (NOW(),'" . $templ . "', '" . $huml . "')";
```

Send a mysql query to insert the data in the POST request in to the database table.

```
mysqli_query($link,$query);
```

Close the connection with the database.

```
mysqli_close($link);
```

Send a raw HTTP header to the client.

```
header("Location: index.php");
```

End of the PHP Script

?>

connect.php

Start of the PHP Script

```
<?php
```

Create a function call Connection to connect with the database

```
function Connection()
```

Create variables for server address username password and the name of the database.

```
$server = "localhost";
$user = "root";
$pass = "";
$db = "arduino_db";
```

Connect with the server using the given username and password and assign the connection to a variable named mysqli.

```
$mysqli = mysqli_connect($server, $user, $pass, $db);
```

If the connection with the server failed, return with an error message.

```
if (!$mysqli) {
    die("Database connection failed: " . mysqli_error());
}
```

Select the given database in the server.

```
$db_select = mysqli_select_db($mysqli, $db);
```

If the selection of the database failed, return with an error message.

```

        if (!$db_select) {
            die("Database selection failed: " . mysqli_error());
        }
    }

```

After a successful connection, return the connection over mysqli variable.

```

    return $mysqli;

```

End of the PHP Script

?>

getdatabase.php

Start of the PHP Script

```

<?php

```

Include the connect.php file to connect with the database

```

    include("connect.php");

```

Make a connection with the database and assign the connection to a variable called link.

```

    $link = Connection();

```

Send a mysql query to get the data from the table named tempLog by the descending order of the timeStamp column and save the result in a variable named result.

```

    $result = mysqli_query($link, "SELECT * FROM `tempLog` ORDER BY `timeStamp` DESC");

```

Send a string with HTML tags to print a table with three columns containing Timestamp, Temperature and Humidity.

```

    echo "<table border = '1' cellspacing = '1' cellpadding = '1'>
    <tr>
    <td> Timestamp &nbsp;    </td>
    <td> Temperature </td>
    <td> Humidity </td>
    </tr>";

```

If the mysql query result is not false, get the data from the table row by row until it reaches the end of the table and print the data in each row in the previously created table.

```

    if($result!==FALSE){
        while($row = mysqli_fetch_array($result)) {
            printf("<tr><td> &nbsp;   :%s </td><td> &nbsp;   :%s&nbsp;    </td><td> &nbsp;   :%s&nbsp;    </td></tr>", $row["timeStamp"], $row["temperature"], $row["humidity"]);
        }
    }

```

Clear the variable result releasing the memory allocated for the data acquired from the database.

```

    mysqli_free_result($result);

```

Close the connection with the database.

```

    mysqli_close($link);

```

Send a string with the HTML tag for the end of the table

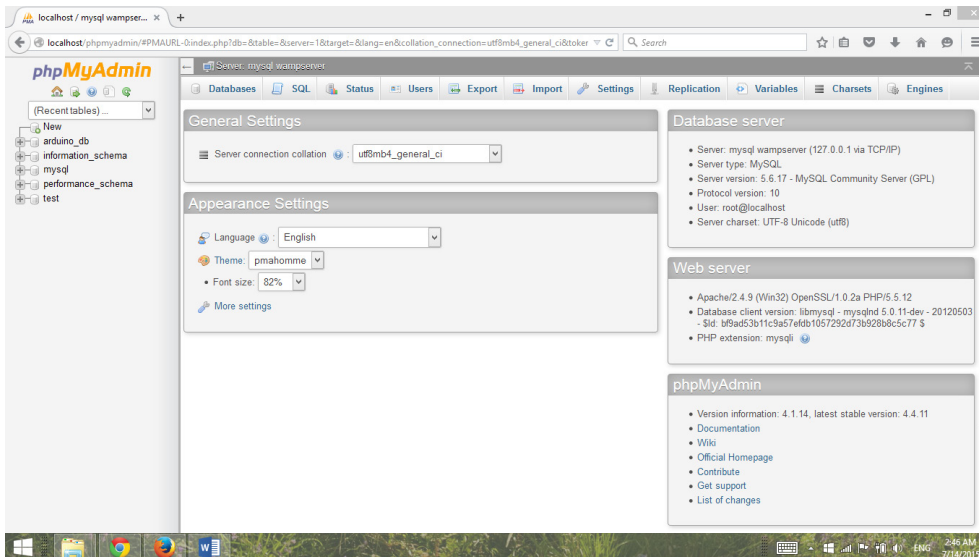
```
echo "</table>"
```

End of the PHP Script

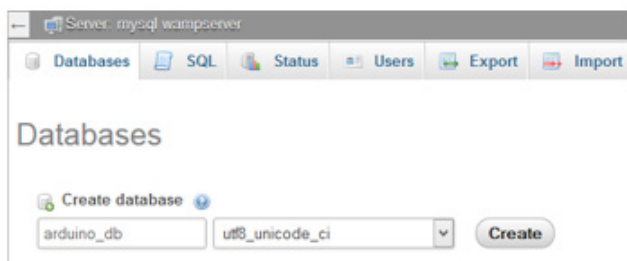
??

STEP 3 – Creating a database and a table to store the weather information.

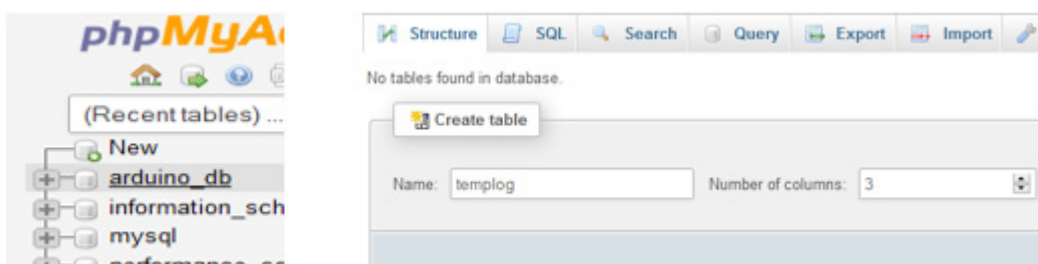
3.1 Open phpMyAdmin by entering `http://localhost/phpmyadmin/` in the address bar of the web browser.



3.2 Select the Databases tab. Type a name for the database (This name should be as same as the name given in connect.php file. Here it is `arduino_db`) and select the Collation as `utf8_unicode_ci`. Click the Create button to create the database with given name.



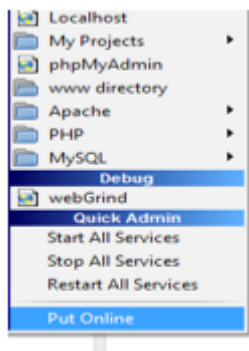
3.3 Select the created database from the left side panel. Type the name of the table given in php files created in the previous step. (Here it is `templog`. The name given in mysql queries in `add.php` and `getdatabase.php` and the name given in `connect.php` file should be as same as the name given here.) Select the number of columns need to store the variables. (Here it is selected as 3 columns for the timestamp, temperature and humidity). Click the Go button to start creating the table.



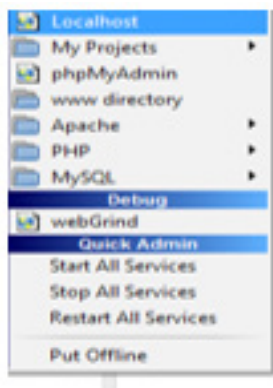
3.4 Type the names of columns as same as the names given in mysql queries in the php files. (Here the column names are timeStamp, temperature and humidity). Select the type of the timeStamp column as TIMESTAMP and the type of other columns as INT. Type the Length/Values of the columns selected as type INT as 11. Select the index of the timeStamp column as PRIMARY. Leave the other fields as it is. Click Save button to create and save the table with given configuration.

STEP 4 – Launching and testing the server and the database.

4.1 Make the server online by clicking on the Put Online button in the WAMPSERVER icon from the windows system tray.



4.2 Open the server home page by clicking the localhost button in the WAMPSERVER tray icon. Header of the web page and the headers of the table columns should display without an error after a successful configuration.



Appendix

Arduino Sample Code & Header Files

Test.ino – Sample Code to test the SMS and HTTP POST functions.

```
#include <SoftwareSerial.h>
#include "GSM_Module.h"

String HOST      = "lahiruwije.ddns.net";    // Host name or address of the web server
int    PORT      = 80;

String Mobile_No = "+94776141305";           // Mobile Number which the SMS Sends
String Test_SMS  = "Test SMS";               // Content of the SMS

int Status;

int temperature  = 42;
int humidity     = 20;

// For SIM900
GSM_Module Module(SIM900);

// For SM5100B
//GSM_Module Module(SM5100B);

void setup()
{
    //Initialize serial port for communication.
    Serial.begin(9600);

    //Initialize the module.
    Serial.println("Module Initializing..");

    Status = Module.Init(9600);

    if( Status == OK ){

        Serial.println("Module Ready.");

    }else{

        Serial.println("Module Initializing Failed.");
        Serial.print("ERROR : ");
        Serial.println(Status);

        while(1){};
    }

    Module.Refresh();
    delay(5000);
}
```

```

void loop() {

    //Sending a test SMS
    Serial.println("Sending a Test SMS");
    Status = Module.Send_SMS(Mobile_No, Test_SMS);

    if( Status == OK ){

        Serial.println("SMS Sent");

    }else{

        Serial.print("SMS ERROR : ");
        Serial.println(Status);

    }

    //Connecting to the Server and Sending Data
    Serial.println("Connecting to Server");

    Status = Module.POST_Data(HOST, PORT, temperature, humidity);

    if( Status == OK ){

        Serial.print("Connecting Successful");
        Serial.print(" ");
        Serial.println(Module.Received_String());

    }else{

        Serial.print("TCP ERROR Code : ");
        Serial.print(Status);
        Serial.print(" ");
        Serial.println(Module.Received_String());

    }

    // Listen to the Module print received data.
    while(1){

        do{

            Module.Wait(1000);

        }while(!Module.String_Received());

        Serial.println(Module.Received_String());

    };

}

```

GSM_Module.h - Header file with GSM Function Prototypes

```

#include "Module_Codes.h"
#include <SoftwareSerial.h>

```

```

#include <avr/pgmspace.h>
#include <string.h>

#define Default_RX 10
#define Default_TX 3

class GSM_Module{
public:
    GSM_Module(uint8_t Type);
    int Init(int baud);
    void Listen();
    void Refresh();
    bool String_Received();
    bool Module_Waiting();
    String Received_String();
    String Decoded_String();
    void Wait(int time_out);
    void Read_Response();
    String Decode_Sys_Info();
    String Decode_GPRS_Info();
    String Decode_SIM900_GPRS_Info();
    String Decode_TCP_Info();
    int Decode_Response();
    bool OK_Received();
    bool Sys_Info_Received();
    bool GPRS_Info_Received();
    bool SMS_Ready();
    bool GPRS_Ready();
    bool Netwok_OK();
    bool Socket_Connecting();
    String Match_System_Code(int code);
    int Wait_For_Respond(int Response_Type,bool OK_Needed,bool *Flag,int Time_
out);
    int Query_GPRS();
    int SIM900_Reset_IP();
    int SIM900_Check_IP_Stack();
    int SIM900_Single_Connection();
    int SIM900_Configure_APN(String APN,String USER_NAME,String PASSWORD);
    int Activate_PDP();
    int SIM900_Get_IP();
    int Configure_TCP(String Host, int Port);
    int Start_TCP_Connection();
    int Query_Socket_Status();
    void Send_AT_Command(String command);
    int ERROR_CODE(int Status, int Error);
    int Send_Data(String Data);
    int Send_SMS(String Number, String MSG);
    int TCP_Connect(String HOST, int PORT);
    int Close_TCP_Connection();
    int POST_Data(String HOST, int PORT, int temp, int hum);
};

```

```

GSM_Module.cpp - C++ file with GSM Function Definitions
#include "GSM_Module.h"
#include <Arduino.h>

```

```

SoftwareSerial cell(Default_RX, Default_TX);

```

```

String APN                = "HUTCH3G";
String USER_NAME          = "";
String PASSWORD            = "";

uint8_t Module_Type       = 0;

char incoming_char         = 0;
String Incoming_String     = "";
String Decoded_Data        = "";

bool Module_Waiting_Flag   = 0;
bool String_Flag           = 0;
bool OK_Flag              = 0;
bool CME_Error_Flag        = 0;
bool CMS_Error_Flag        = 0;
bool System_Info_Flag      = 0;
bool GPRS_Info_Flag        = 0;
bool TCP_Info_Flag         = 0;
bool GPRS_Flag             = 0;
bool Call_Ready_Flag       = 0;
bool SMS_Ready_Flag        = 0;
bool Network_OK_Flag       = 0;
bool TCP_Connect_Flag      = 0;
bool TCP_Connecting_Flag   = 0;
bool SIM900_IP_Stack_Flag  = 0;

int Response_Type          = 0;

GSM_Module::GSM_Module(uint8_t Type){

    Module_Type             = Type;

}

int GSM_Module::Init(int baud){

    int Status              = 0;

    if( Module_Type == SIM900 ){

        cell.begin(baud);
        delay(1000);
        cell.println("AT");

        if( Wait_For_Respond(OK, 0, &OK_Flag, 10) == OK ){

            return OK;

        }else{

            digitalWrite(SIM900_Power_Pin, HIGH);
            delay(1000);
            digitalWrite(SIM900_Power_Pin, LOW);
            delay(3000);
            digitalWrite(SIM900_Power_Pin, HIGH);

            delay(6000);

            for( int attempts = 0 ; attempts < 20 ; attempts++){

```



```

        cell.println("AT");
        Status = Wait_For_Respond(OK, 0, &OK_Flag, 100);

        while ( Status == UNKNOWN_RESPONSE ){
            Status = Wait_For_Respond(OK, 0, &OK_Flag, 100);
        }

        if ( Status != ERROR_RESPONSE_TIMED_OUT ){
            return Status;
        }
        delay(1000);
    }

    return 0;
}

}else{
    cell.begin(baud);
    do{

        do{

            Wait(1000);

        }while(!String_Received());

        Status = Decode_Response();

        Refresh();

    }while(!(SMS_Ready() && Netwok_OK()));

    return OK;
}
}

void GSM_Module::Refresh(){

    Module_Waiting_Flag = 0;
    String_Flag = 0;
    incoming_char = 0;
    Incoming_String = "";

}

bool GSM_Module::String_Received(){

    return String_Flag;

}

bool GSM_Module::Module_Waiting(){

    return Module_Waiting_Flag;

}

void GSM_Module::Listen(){

```

```

if(cell.available() >0)
{
    incoming_char = cell.read();
    Incoming_String += incoming_char;
    if (incoming_char == '\n'){
        if((Incoming_String == "\r\n") || (Incoming_String == " \r\n")){
            Incoming_String = "";
        }else{
            String_Flag = 1;
        }
    }else if (incoming_char == '>'){
        Module_Waiting_Flag = 1;
    }
}
}

String GSM_Module::Received_String(){

    return Incoming_String;

}

String GSM_Module::Decoded_String(){

    return Decoded_Data;

}

void GSM_Module::Wait(int time_out){

    int loops      = 0;
    int time_steps = 0;
    bool break_loop = 0;
    GSM_Module::Refresh();

    while(((time_out<0) || (time_steps < time_out)) && !(break_loop)){

        GSM_Module::Listen();
        loops = loops + 1;
        if(loops > 2000){
            loops = 0;
            time_steps = time_steps + 1;
        }
        break_loop = Module_Waiting_Flag || String_Flag;
    }
}

void GSM_Module::Read_Response(){

    OK_Flag          = 0;
    CMS_Error_Flag   = 0;
    CME_Error_Flag    = 0;
    System_Info_Flag  = 0;
    GPRS_Info_Flag    = 0;
    TCP_Info_Flag     = 0;
    Response_Type     = 0;
    SIM900_IP_Stack_Flag = 0;

    if(Incoming_String.endsWith("OK\r\n")){

```

```

    OK_Flag                = 1;
    Response_Type          = 1;

}else if(Incoming_String.startsWith("+CME")){

    CME_Error_Flag        = 1;
    Response_Type          = 2;

}else if(Incoming_String.startsWith("+CMS")){

    CMS_Error_Flag        = 1;
    Response_Type          = 3;

}else if(Incoming_String.startsWith("+SIND")){

    System_Info_Flag       = 1;
    Response_Type          = 4;

}else if(Incoming_String.startsWith("+CGATT")){

    GPRS_Info_Flag        = 1;
    Response_Type          = 5;

}else if(Incoming_String.startsWith("+SOCKSTATUS")){

    TCP_Info_Flag          = 1;
    Response_Type          = 6;

}else if(Incoming_String.startsWith("AT")){

    Response_Type          = 7;

}else if(Incoming_String.startsWith("STATE")){

    Response_Type          = 8;
    SIM900_IP_Stack_Flag   = 1;
}
}
String GSM_Module::Decode_Sys_Info(){

    int String_Length = Incoming_String.length();
    int System_Code    = -1;

    if(String_Length == 53){

        System_Code = 10 + 2*(Incoming_String.substring(15,16)).toInt();

    }else if(String_Length == 10){

        System_Code = (Incoming_String.substring(7,8)).toInt();

    }else if(String_Length == 11){

        System_Code = (Incoming_String.substring(7,9)).toInt();

    }

    if(System_Code == 3){

        Call_Ready_Flag = 1;

```

```

}else if(System_Code == 4){

    SMS_Ready_Flag = 1;

}else if(System_Code == 11){

    Network_OK_Flag = 1;

}

return Match_System_Code(System_Code);
}

String GSM_Module::Decode_GPRS_Info(){

    if(Incoming_String.substring(8,9) == "0"){

        GPRS_Flag = 0;
        return GPRS_Info_0;

    }else if(Incoming_String.substring(8,9) == "1"){

        GPRS_Flag = 1;
        return GPRS_Info_1;

    }

    return Unknown_code;

}

String GSM_Module::Decode_SIM900_GPRS_Info(){

    if(Incoming_String.substring(7,17) == "IP INITIAL"){

        SIM900_IP_Stack_Flag = 1;
        return SIM900_GPRS_Info_0;

    }else{

        SIM900_IP_Stack_Flag = 0;
        return SIM900_GPRS_Info_1;

    }

}

String GSM_Module::Decode_TCP_Info(){

    if(Incoming_String.substring(18,22) == "0104"){

        TCP_Connecting_Flag = 1;
        TCP_Connect_Flag = 0;
        return TCP_Info_0;

    }else if(Incoming_String.substring(18,22) == "0102"){

        TCP_Connect_Flag = 1;

```

```

    TCP_Connecting_Flag = 0;
    return TCP_Info_1;

} else if(Incoming_String.substring(18,22) == "0100"){

    TCP_Connect_Flag      = 0;
    TCP_Connecting_Flag   = 0;
    return TCP_Info_2;
}
return Unknown_code;
}

int GSM_Module::Decode_Response(){

    Read_Response();

    Decoded_Data          = "";

    switch(Response_Type){

        case 1:
            Decoded_Data = "OK";
            return OK;

        case 2:
            Decoded_Data = CME_Error;
            return ERROR_CME;

        case 3:
            Decoded_Data = CMS_Error;
            return ERROR_CMS;

        case 4:
            Decoded_Data = Decode_Sys_Info();
            return TYPE_SYS_INF0;

        case 5:
            Decoded_Data = Decode_GPRS_Info();
            return TYPE_GPRS_INF0;

        case 6:
            Decoded_Data = Decode_TCP_Info();
            return TYPE_TCP_INF0;

        case 7:
            Decoded_Data = "ECH0";
            return ECH0;

        case 8:
            Decoded_Data = Decode_SIM900_GPRS_Info();
            return SIM900_GPRS_INF0;

        default:
            Decoded_Data = Incoming_String;
            return UNKNOWN_RESPONSE;

    }
}

bool GSM_Module::OK_Received(){

```

```

    return OK_Flag;
}

bool GSM_Module::Sys_Info_Received(){
    return System_Info_Flag;
}

bool GSM_Module::GPRS_Info_Received(){
    return GPRS_Info_Flag;
}

bool GSM_Module::SMS_Ready(){
    return SMS_Ready_Flag;
}

bool GSM_Module::GPRS_Ready(){
    return GPRS_Flag;
}

bool GSM_Module::Netwok_OK(){
    return Network_OK_Flag;
}

bool GSM_Module::Socket_Connecting(){
    return TCP_Connecting_Flag;
}

String GSM_Module::Match_System_Code(int code){
    switch(code){
        case 0:
            return Sys_Codes_0;
        case 1:
            return Sys_Codes_1;
        case 3:
            return Sys_Codes_3;
        case 4:
            return Sys_Codes_4;
        case 7:
            return Sys_Codes_7;
        case 10:
            return Sys_Codes_10;
        case 11:
            return Sys_Codes_11;
        case 12:
            return Sys_Codes_12;
    }
}

```



```

        default:
            return Unknown_code;
    }
}

int GSM_Module::Wait_For_Respond(int Response_Type, bool OK_Needed, bool *Flag, int
Time_out){

    int Status = 0;

    Wait(Time_out);

    if(String_Received()){

        Status = Decode_Response();

        if( Status == ECHO ){

            Wait(Time_out);

            if(String_Received()){

                Status = Decode_Response();

            }else if((Response_Type == WAITING_FOR_DATA) && Module_Waiting()){

                if(Flag) return 1; else return 0;

            }else{

                return ERROR_RESPONSE_TIMED_OUT;

            }

        }

        if( Status < 0 ){

            return Status;

        }else if(Status == Response_Type){

            if(OK_Needed){

                Wait(1000);

                if(String_Received()){

                    Status = Decode_Response();

                    if(Flag) return Status; else return 0;

                }

            }

            if(Flag) return 1; else return 0;

        }else{

            return ERROR_WRONG_RESPONSE_TYPE;

        }

    }

}

```

```

    }

    }else if((Response_Type == WAITING_FOR_DATA) && Module_Waiting()){

        if(Flag) return 1; else return 0;

    }

    return ERROR_RESPONSE_TIMED_OUT;

}

int GSM_Module::Query_GPRS(){

    Send_AT_Command(AT_CMD_Query_GPRS);
    return Wait_For_Respond(TYPE_GPRS_INFO, 1, &GPRS_Flag, 100);

}

int GSM_Module::SIM900_Reset_IP(){

    Send_AT_Command(AT_CMD_SIM900_Reset_IP);
    return Wait_For_Respond(OK, 0, &OK_Flag, 1000);

}

int GSM_Module::SIM900_Check_IP_Stack(){

    int Status = 0;

    Send_AT_Command(AT_CMD_SIM900_IP_Stack);

    Status = Wait_For_Respond(OK, 0, &OK_Flag, 1000);

    if( Status == OK ){

        return Wait_For_Respond(SIM900_GPRS_INFO, 0, &SIM900_IP_Stack_Flag, 1000);

    }else{

        return Status;

    }

}

int GSM_Module::SIM900_Single_Connection(){

    Send_AT_Command(AT_CMD_S9_Single_Conn);
    return Wait_For_Respond(OK, 0, &OK_Flag, 1000);

}

int GSM_Module::SIM900_Configure_APN(String APN, String USER_NAME, String PASSWORD){

    String AT_CMD_APN_Config = AT_CMD_S9_APN_Config;
    AT_CMD_APN_Config.concat("");
    AT_CMD_APN_Config.concat(APN);
    AT_CMD_APN_Config.concat("");

```

```

AT_CMD_APN_Config.concat(",");
AT_CMD_APN_Config.concat("");
AT_CMD_APN_Config.concat(USER_NAME);
AT_CMD_APN_Config.concat("");
AT_CMD_APN_Config.concat(",");
AT_CMD_APN_Config.concat("");
AT_CMD_APN_Config.concat(PASSWORD);
AT_CMD_APN_Config.concat("");

Send_AT_Command(AT_CMD_APN_Config);
return Wait_For_Respond(OK, 0, &OK_Flag, 1000);
}

int GSM_Module::Activate_PDP(){
    if(Module_Type == SIM900){
        Send_AT_Command(AT_CMD_S9_Activate_PDP);
    }else{
        Send_AT_Command(AT_CMD_Activate_PDP);
    }
    return Wait_For_Respond(OK, 0, &OK_Flag, 5000);
}

int GSM_Module::SIM900_Get_IP(){
    Send_AT_Command(AT_CMD_S9_Get_IP);
    return Wait_For_Respond(OK, 0, &OK_Flag, 10);
}

int GSM_Module::Configure_TCP(String Host, int Port){
    int Status;
    String AT_TCP_CONF;

    if(Module_Type == SIM900){
        AT_TCP_CONF = AT_CMD_S9_TCP_Config;
    }else{
        AT_TCP_CONF = AT_CMD_TCP_Config;
    }

    AT_TCP_CONF.concat("");
    AT_TCP_CONF.concat("TCP");
    AT_TCP_CONF.concat("");
    AT_TCP_CONF.concat(",");
    AT_TCP_CONF.concat("");
    AT_TCP_CONF.concat(Host);
    AT_TCP_CONF.concat("");
    AT_TCP_CONF.concat(",");

    if(Module_Type == SIM900){

```

```

    AT_TCP_CONF.concat("");
}

AT_TCP_CONF.concat(Port);

if(Module_Type == SIM900){
    AT_TCP_CONF.concat("");
}
Send_AT_Command(AT_TCP_CONF);

Status = Wait_For_Respond(OK, 0, &OK_Flag, 1000);

if(Status == OK){
    if(Module_Type == SIM900){
        return Wait_For_Respond(OK, 0, &OK_Flag, 10000);
    }else{
        return Status;
    }
}

}

int GSM_Module::Start_TCP_Connection(){
    Send_AT_Command(AT_CMD_Start_TCP);
    return Wait_For_Respond(OK, 0, &OK_Flag, 100);
}

int GSM_Module::Query_Socket_Status(){
    TCP_Connect_Flag      = 0;
    TCP_Connecting_Flag   = 0;

    Send_AT_Command(AT_CMD_Query_Socket);
    return Wait_For_Respond(TYPE_TCP_INF0, 1, &TCP_Connect_Flag, 10);
}

void GSM_Module::Send_AT_Command(String command){
    String AT_Command = "AT+";
    AT_Command.concat(command);

    cell.println(AT_Command);
}

```

```

int GSM_Module::ERROR_CODE(int Status, int Error){

    return Error+Status;

}

int GSM_Module::Send_Data(String Data){

    int Status          = 0;
    int ST_length       = Data.length();
    String Send_CMD     = AT_CMD_Send_Data;

    Send_CMD.concat(ST_length);

    if( Module_Type == SIM900 ){

        Send_AT_Command(AT_CMD_SQ_Send_Data);

    }else{

        Send_AT_Command(Send_CMD);
    }

    Status = Wait_For_Respond(WAITING_FOR_DATA, 0, &Module_Waiting_Flag, 1000);

    if(Status == OK){

        cell.print(Data);
        cell.write(0x1A);

        return Wait_For_Respond(OK, 0, &OK_Flag, 1000);
    }

    return Status;
}

int GSM_Module::Send_SMS(String Number, String MSG){

    int Status = 0;

    Send_AT_Command(AT_CMD_SMS_Mode);
    Status = Wait_For_Respond(OK, 0, &OK_Flag, 1000);

    if( Status == OK ){

        String SMS_Number = AT_CMD_SMS_Number;
        SMS_Number.concat(Number);
        SMS_Number.concat("\"");

        Send_AT_Command(SMS_Number);
        Status = Wait_For_Respond(WAITING_FOR_DATA, 0, &Module_Waiting_Flag, 1000);

        if( Status == OK ){

            cell.print(MSG);
            cell.write(0x1A);

            Status = Wait_For_Respond(OK, 0, &OK_Flag, 1000);

            while( Status == UNKNOWN_RESPONSE ){

```

```

        Status = Wait_For_Respond(OK, 0, &OK_Flag, 1000);
    }
    return Status;

}

}

return ERROR_CODE(Status, ERROR_SMS_NUMBER_NOT_SET);
}

}

return ERROR_CODE(Status, ERROR_SMS_MODE_NOT_SELECTED);
}
}

int GSM_Module::TCP_Connect(String HOST, int PORT){

    int STATUS = 0;

    STATUS = Query_GPRS();
    if( STATUS != OK ){
        return ERROR_CODE(STATUS, ERROR_GPRS_NOT_ATTACHED);
    }

    if( Module_Type == SIM900 ){

        STATUS = SIM900_Reset_IP();
        if( STATUS != OK ){
            return ERROR_CODE(STATUS, ERROR_IP_RESET_FAILED);
        }

        STATUS = SIM900_Check_IP_Stack();
        if( STATUS != OK ){
            return ERROR_CODE(STATUS, ERROR_IP_STACK_FAILED);
        }

        STATUS = SIM900_Single_Connection();
        if( STATUS != OK ){
            return ERROR_CODE(STATUS, ERROR_SINGLE_CONNECTION_FAILED);
        }

        STATUS = SIM900_Configure_APN(APN, USER_NAME, PASSWORD);
        if( STATUS != OK ){
            return ERROR_CODE(STATUS, ERROR_APN_SETTING_FAILED);
        }
    }

    STATUS = Activate_PDP();
    if( STATUS != OK ){
        return ERROR_CODE(STATUS, ERROR_PDP_ACTIVATION_FAILED);
    }

    if( Module_Type == SIM900 ){

        STATUS = SIM900_Get_IP();
        if( STATUS != UNKNOWN_RESPONSE ){
            return ERROR_CODE(STATUS, ERROR_COULD_NOT_GET_IP);
        }
    }
}

```



```

STATUS = Configure_TCP(HOST, PORT);
if( STATUS != OK ){
    return ERROR_CODE(STATUS, ERROR_TCP_CONFIG_FAILED);
}

if( Module_Type != SIM900 ){

    STATUS = Start_TCP_Connection();
    if( STATUS != OK ){
        return ERROR_CODE(STATUS, ERROR_TCP_START_FAILED);
    }

    do{

        STATUS = Query_Socket_Status();

    }while(Socket_Connecting());

    if( STATUS != OK ){
        return ERROR_CODE(STATUS, ERROR_TCP_CONNECTION_FAILED);
    }
}
return STATUS;
}

int GSM_Module::Close_TCP_Connection(){

    Send_AT_Command(AT_CMD_SIM900_Reset_IP);
    return Wait_For_Respond(OK, 0, &OK_Flag, 1000);
}

int GSM_Module::POST_Data(String HOST, int PORT, int temp, int hum){

    int Status = 0;

    Status = TCP_Connect(HOST, PORT);

    if(Status == OK){

        String POST_Data_Line_5 = POST_Data_Line_5_1;
        POST_Data_Line_5.concat(temp);
        POST_Data_Line_5.concat(POST_Data_Line_5_2);
        POST_Data_Line_5.concat(hum);

        String POST_Data = POST_Data_Line_1;
        POST_Data.concat(POST_Data_Line_2);
        POST_Data.concat(HOST);
        POST_Data.concat(POST_Data_Line_3);
        POST_Data.concat(POST_Data_Line_4);
        POST_Data.concat((POST_Data_Line_5.length()-4));
        POST_Data.concat(POST_Data_Line_5);

        Status = Send_Data(POST_Data);

        if(Module_Type == SIM900){
            if(Status == OK){
                return Close_TCP_Connection();
            }else{
                return Status;
            }
        }else{

```

```

        return Status;
    }
    }else{
        return Status;
    }
}

```

Module Codes.h - Header File with Error Code Definitions & AT Commands.

```

#define ERROR_RESPONSE_TIMED_OUT          -1
#define ERROR_CME                         -2
#define ERROR_CMS                         -3
#define ERROR_WRONG_RESPONSE_TYPE        -4
#define UNKNOWN_RESPONSE                  -5
#define ECHO                              -6

#define ERROR_GPRS_NOT_ATTACHED           -10
#define ERROR_IP_RESET_FAILED             -20
#define ERROR_IP_STACK_FAILED             -30
#define ERROR_SINGLE_CONNECTION_FAILED    -40
#define ERROR_APN_SETTING_FAILED          -50
#define ERROR_PDP_ACTIVATION_FAILED       -60
#define ERROR_COULD_NOT_GET_IP            -70
#define ERROR_TCP_CONFIG_FAILED           -80
#define ERROR_TCP_START_FAILED            -90
#define ERROR_TCP_CONNECTION_FAILED       -100

#define ERROR_SMS_MODE_NOT_SELECTED       -10
#define ERROR_SMS_NUMBER_NOT_SET          - 20

#define OK                                 1

#define TYPE_SYS_INFO                     2
#define TYPE_GPRS_INFO                    3
#define TYPE_TCP_INFO                     4
#define WAITING_FOR_DATA                   5
#define SIM900_GPRS_INFO                   6

#define SIM900                             2
#define SM5100B                             3

#define SIM900_Power_Pin                    7

const char Sys_Codes_0[3]                  = "S1";           //SEARCHING SIM";
const char Sys_Codes_1[3]                  = "S2";           //SIM INSERTED";
const char Sys_Codes_3[3]                  = "S3";           //CALL READY";
const char Sys_Codes_4[3]                  = "S4";           //SMS READY";
const char Sys_Codes_7[3]                  = "S5";           //NO SIM - ERROR";
const char Sys_Codes_10[3]                 = "S6";           //NO SIM FOUND";
const char Sys_Codes_11[3]                 = "S7";           //REGISTERED TO NETWORK";
const char Sys_Codes_12[3]                 = "S8";           //SIM READY";

const char Unknown_code[3]                 = "UC";           //UNKNOWN CODE";

const char GPRS_Info_0[3]                  = "G1";           //GPRS NOT ATTACHED";
const char GPRS_Info_1[3]                  = "G2";           //GPRS ATTACHED";

const char TCP_Info_0[3]                   = "T1";           //CONNECTING";
const char TCP_Info_1[3]                   = "T2";           //CONNECTED";
const char TCP_Info_2[3]                   = "T3";           //CONNECTION FAILED";

```

```

const char No_Response[3]      = "NR";                //NO RESPONSE";

const char CME_Error[3]       = "CE";                //CME ERROR";
const char CMS_Error[3]       = "CS";                //CMS ERROR";

const char SIM900_GPRS_Info_0[3] = "90";
const char SIM900_GPRS_Info_1[3] = "91";

const char AT_CMD_Query_GPRS[7]      = "CGATT?";
const char AT_CMD_Activate_PDP[10]   = "CGACT=1,1";
const char AT_CMD_TCP_Config[13]     = "SDATACONF=1,1";
const char AT_CMD_Start_TCP[15]      = "SDATASTART=1,1";
const char AT_CMD_Query_Socket[14]   = "SDATASTATUS=1";
const char AT_CMD_Send_Data[14]      = "SDATASEND=1,1";
const char AT_CMD_SMS_Mode[7]        = "CMGF=1";
const char AT_CMD_SMS_Number[8]      = "CMGS=\"";

const char AT_CMD_SIM900_Reset_IP[8] = "CIPSHUT";
const char AT_CMD_SIM900_IP_Stack[10] = "CIPSTATUS";
const char AT_CMD_S9_Single_Conn[9]   = "CIPMUX=0";
const char AT_CMD_S9_APN_Config[7]    = "CSTT= ";
const char AT_CMD_S9_Activate_PDP[6]  = "CIICR";
const char AT_CMD_S9_TCP_Config[10]   = "CIPSTART=";
const char AT_CMD_S9_Get_IP[6]        = "CIFSR";
const char AT_CMD_S9_Send_Data[8]     = "CIPSEND";

const char POST_Data_Line_1[25]       = "POST /add.php HTTP/1.1\r\n";
const char POST_Data_Line_2[7]        = "Host: ";
const char POST_Data_Line_3[52]       = "\r\nContent-Type: application/x-www-form-urlencoded\r\n"; //49
const char POST_Data_Line_4[19]       = "Content-Length: "; //20
const char POST_Data_Line_5_1[11]     = "\r\n\r\ntemp1="; //16
const char POST_Data_Line_5_2[7]      = "&hum1="; //16

```

PHP and HTML Codes to test the Server

Index.html

```

<html>
<head>
<title>Sensor Data</title>
<script>
function loadXMLDoc()
{
    var xmlhttp;
    if (window.XMLHttpRequest)
    { // code for IE7+, Firefox, Chrome, Opera, Safari
        xmlhttp=new XMLHttpRequest();
    }
    else
    { // code for IE6, IE5
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange=function()
    {
        if (xmlhttp.readyState==4 && xmlhttp.status==200)
        {
            document.getElementById("myDiv").innerHTML=xmlhttp.response-

```

```

Text; // your div
    }
    }
    xmlhttp.open("GET","getdatabase.php",true); //your php file
    xmlhttp.send();
}
window.setInterval(function(){
    loadXMLDoc();
}, 5000);

</script>
</head>
<body>
    <h1>Temperature and Humidity Sensor Readings</h1>

    <div id="myDiv"></div>

</body>
</html>

```

connect.php

```

<?php

function Connection(){
    $server="localhost";
    $user="root";
    $pass="";
    $db="arduino_db";

    $mysqli = mysqli_connect($server, $user, $pass, $db);

    if (!$mysqli) {
        die("Database connection failed: " . mysqli_error());
    }

    $db_select = mysqli_select_db($mysqli, $db);
    if (!$db_select) {
        die("Database selection failed: " . mysqli_error());
    }

    return $mysqli;
}

?>

```

add.php

```

<?php
include("connect.php");

$link=Connection();

$temp=$_POST["temp"];
$hum=$_POST["hum"];

```

```

$query = "INSERT INTO `templog` (`timeStamp`, `temperature`, `humidity`)
VALUES (NOW(), '" . $templ . "', '" . $huml . "')";

mysqli_query($link, $query);
mysqli_close($link);

header("Location: index.php");
?>

```

getdatabase.php

```

<?php

include("connect.php");

$link=Connection();

$result=mysqli_query($link,"SELECT * FROM `tempLog` ORDER BY `timeStamp` DESC");

echo "<table border='1' cellspacing='1' cellpadding='1'>
<tr>
<td>Timestamp</td>
<td>Temperature</td>
<td>Humidity</td>
</tr>";

if($result!==FALSE){
    while($row = mysqli_fetch_array($result)) {
        printf("<tr><td> &nbsp;%s </td><td> &nbsp;%s&nbsp;</td><td> &nbsp;%s&nbsp;</td></tr>", $row["timeStamp"], $row["temperature"], $row["humidity"]);
    }
    mysqli_free_result($result);
    mysqli_close($link);
}
echo "</table>"

?>

```

