

State University of New York at New Paltz

Department of Electrical and Computer Engineering

Embedded Linux
CPS342-01

Home Security Project

Project Documentation

Group Members	Department
Ivan Vivar	CE

Course Professor: Professor Easwaran

Table of Contents

Project Description.....	3
<i>A message from the Project Manager to the reader</i>	4
Installing the necessary Packages on the Raspberry Pi	5
Setting up a SQLite Database from fixed data	6
Blinker LED & SQLite Database	9
Displaying SQLite Database with PHP	11
Problems that Occurred	13

Project Description

The Objective of this project was to use an Arduino microcontroller to run four different types of sensors that would simulate a real time Home monitoring system. The Arduino would then send the sensor data to a Raspberry Pi microcontroller; the Raspberry Pi would then receive the data and store it in its respective database. After database storage is complete the Raspberry Pi would then send the data to a web site that is responsive, scales the website properly depending on the size of the window or device the web site is being run on. The website would then display the sensor data for each sensor for both relevant (current) data and overall (historical) data.

The sensors included in this project were multiple Light Emitting Diodes (LED) sensors each connected by a switch button to toggle between being on and off, a temperature sensor, a buzzer sensor controlled by a distance(sonar) sensor and a motion sensor which triggered an LED . This sensor and button combination was chosen in order to represent a light that corresponded to different rooms in the house. It might be of value to the home owner to check whether the lights of the house were left on after they have left. The Temperature sensor was incorporated in order to notify the home owner of the current temperature of the house through the website. The combination of a distance sensor with a buzzer was used to represent a home's alarm system. This sensor combination along with a switch button would allow the home owner to check the state in which the home's alarm system was in on the website, it could either be in a *Alarm On*, *Alarm Off* or *Alarm Triggered* state. The last sensor combination used is the motion detector sensor and a LED, this sensor combination was used in order to represent a home's exterior motion detector lighting usually located in the backyard, driveway or a house's exterior

where it is usually very dark, inactive or a blind spot for the home owners when inside the house to detect possible outside threats.

These sensors would then produce the data accordingly and the Raspberry Pi would take the data and pass it onto a Sqlite database and display the information on a responsive website that uses HTML, CSS, and PHP.

A message from the Project Manager to the reader

This project consisted of the knowledge of but not limited to the knowledge of both Arduino and Raspberry Pi microprocessors, C, Python, ATD, HTML, CLI (Command Line Instructions), sqlite database, HTML, Vim, nano, CSS, and PHP. Some of these materials needed for this project were out of my scope of knowledge and because of this reason this project has not met its *Project Description* as mentioned above because a lot of preliminary steps were taken in order to familiarize myself with the new material which consumed most of the duration of the time's project. Due to this fact, this documentation will be outlined by a traditional step by step fashion with a list of problems that happen at a given time listed in the *Problems that occurred* section in order to describe the problem in a timely fashion. These parts in the documentation are denoted by the asterisk symbol (*) followed by a certain number (i.e. 1, 2 and etc). This is done to still preserve the step by step fashion and not go back and forth from project procedures to project errors that occurred.

Installing the necessary Packages on the Raspberry Pi

STEP 1: From the *LXTerminal* on the Raspberry Pi install these packages via Command Line.

- `sudo apt-get update`
- `sudo apt-get upgrade`
- `sudo apt-get gpm` //Allows for mouse cursor movement in terminal.
- `sudo apt-get install sqlite3` //Sqlite3 is use to construct and maintain our databases. (*1)
- `sudo apt-get install lighttpd` //lightweight http server
- `sudo apt-get install geany` //IDE for php scripts (*2)
- `sudo apt-get install php5`
- `sudo apt-get install php5-common`
- `sudo apt-get install php5-dev`
- `sudo apt-get install php5-cli`
- `sudo apt-get install php5-cgi`
- `sudo apt-get install php5-sqlite`

STEP 2: Enable Lighttpd fastcgi module and the PHP PDO drivers

- `lighttpd-enable-mod fastcgi`
- `lighttpd-enable mod fastcgi -php`

STEP 3: Reconfigure lighttpd

- `service lighttpd force-reload`

Setting up a SQLite Database from fixed data

Having no previous knowledge of SQLite and databases as a whole, before passing sensory data onto a database the preliminary step of how to create a database, update a database, and display a database was taken by first using a file with fixed data prewritten.

Step 1: Before creating a database, in the *LXTerminal* go to the directory in which you wish to save the Database in because once created it will save in the directory you are currently in. Once there execute the following CLI (Command Line Instruction). This will start up SQLite and also create a Database in the directory you are in called *FixedData.db*, if a database within this directory already exists then this instruction will open up the already existing database.

- *Sudo sqlite3 FixedData.db*

You will now see something along the lines of what is shown in **Figure 1**.

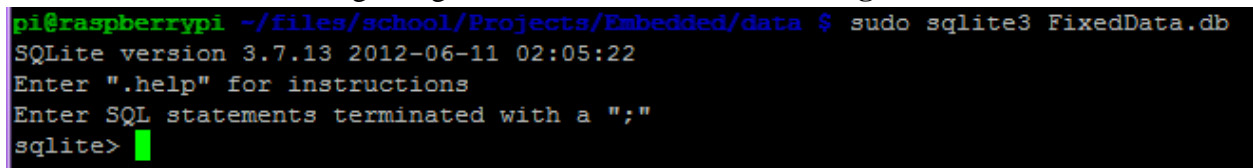
A screenshot of a terminal window on a Raspberry Pi. The prompt is 'pi@raspberrypi ~/files/school/Projects/Embedded/data \$'. The command 'sudo sqlite3 FixedData.db' has been entered. The output shows 'SQLite version 3.7.13 2012-06-11 02:05:22', 'Enter ".help" for instructions', and 'Enter SQL statements terminated with a ";"'. The prompt has changed to 'sqlite>' with a green cursor.

Figure 1: SQLite start up message.

Notice that the cursor in your terminal has changed to *sqlite>* this is showing you that you are now in the sqlite domain and you can execute SQLite CLI. Once here SQLite instructions do not need a semicolon at the end of each instruction however, non-SQLite CLI still do if still operating within the SQLite domain region. Use the CLI *.help* to find a list of SQLite CLI and their description.

Step 2: Create a table within your database for your data. Execute the following CLI.

- *Sqlite> create table BBData(Time int, Temp int)*

BBData, is the name of the table you are creating within your *FixedData.db* database. (*Time int*, *Temp int*), represents an argument where the parameters are Column names and their data type. Each Column is separated by a *','*. In this example the Column names are *Time* and *Temp* and there both have a data type of *int*.

Step 3: Create a separator, which is a string symbol that allows SQLite to recognize the difference in columns for either an imported (fixed) or generated (sensory data) data for the table.

.separator “,”

As seen in **Figure 2** you can check to see if you have done **Step 3** correctly by executing a SQLite `.show` instruction.

```
sqlite> .separator ','  
sqlite> .show  
    echo: off  
  explain: off  
  headers: off  
    mode: list  
nullvalue: ""  
  output: stdout  
separator: ","  
    stats: off  
    width:  
sqlite> █
```

Figure 2: `.separator` and `.show` instructions.

Step 4: Import fixed data (*3)

As shown in **Figure 3** use the `.import` instruction followed by the *filename* in this case *tempData.csv* to import a set of fixed data to the table you are currently on.

```
sqlite> .import tempData.csv █
```

Figure 3: `.import` instruction

Step 5: Checking your Database and Table(s) (*4)

Execute the instructions `.tables` and `.schema`. `.tables` will give you a list of all the tables within the database you are currently in and `.schema` will show you the layout of those tables including column names and their data types.

```
pi@raspberrypi ~/files/school/Projects/Embedded/data $ sqlite3 FixedData.db  
SQLite version 3.7.13 2012-06-11 02:05:22  
Enter ".help" for instructions  
Enter SQL statements terminated with a ";"  
sqlite> .tables  
BBData  
sqlite> .schema  
CREATE TABLE BBData(Time int, Temp int);  
sqlite> █
```

Figure 4: `.tables` and `.schema` outcomes when executed.

To see the actual data within a table, execute the following instruction. Make sure to add a semicolon at the end or else the instruction will not execute. This is not a SQLite instruction and therefore requires a semicolon at the end.

*Sqlite> SELECT * FROM BBData;*

```
sqlite> SELECT * FROM BBData;  
1,12  
2,12  
3,13  
4,13  
5,13  
6,14  
7,13  
8,13  
9,14  
10,14  
11,14  
12,15  
13,14  
14,15  
15,16  
16,17  
17,18  
18,19  
19,18  
20,18  
21,20  
22,19  
23,18  
24,17  
1,16  
2,15  
3,14  
4,15  
5,15  
6,15  
7,15  
8,16  
9,16  
10,16  
11,16  
12,17  
13,17  
14,17  
15,17  
16,18  
17,18  
18,18  
19,19  
20,17  
21,16  
22,15  
23,14  
24,11  
1,11
```

Figure 5: Data within the BBData table within the FixedData.db database.

Blinker LED & SQLite Database

This project can be divided into two parts. One being the programming of the sensors, which is the easy part and the other being the programming of passing and displaying information onto the database and website. Due to the fact that I had no knowledge on passing and displaying database information, a preliminary step was taken in order to learn how to pass and display database data as soon as possible. That preliminary step was the creation of a LED program called LED that oscillated between being on and off causing it to blink and every time the LED was on it sent data to the database. The data sent was the status of the LED being on and the Date and Time stamp of that LED being on. The following code is shown in **Figure 6** do note though that the time delay is very short and was made to be short in order to debug the code and check at a more frequent time interval if data was actually being passed to the database.

```
1 import sqlite3
2 import datetime
3 import RPi.GPIO as GPIO # Import class/package for GPIO PINS
4 import time
5 GPIO.setwarnings(False) # Disable warning message
6 n = 0;
7 # function to make LED Blink
8 def blink(pin):
9     GPIO.output(pin,GPIO.HIGH)
10    LED_Status = 'ON'
11    time.sleep(1)
12    GPIO.output(pin,GPIO.LOW)
13    time.sleep(1)
14    return LED_Status
15
16 GPIO.setmode(GPIO.BOARD) # sets up GPIO PINS
17 GPIO.setup(15,GPIO.OUT) # pin 15 - GPIO 22 is an output
18 Date_Time = str(datetime.datetime.now())
19 db = sqlite3.connect('/home/pi/files/school/Projects/Embedded/data/LED_DB') #connects database with this script by db variable
20 cursor = db.cursor() # creates cursor for database
21 db.commit()
22
23 while n<=5: # infinite loop
24     LED_Status= blink(15)
25     TableData = [{LED_Status,Date_Time}]
26     print TableData
27     #cursor.executemany('INSERT INTO LED_STATUS Values(?,?)', TableData )
28     cursor.execute('INSERT INTO LED_STATUS Values(?,?)', (LED_Status,Date_Time))
29     db.commit()
30     n = n + 1
31 GPIO.cleanup() # cleans up signals
32 db.close() # closes connection to database when done
33
```

Figure 6: LED program to practice how to send data to a database.

The Database and tables used for this program is shown in **Figure 7**

```
pi@raspberrypi ~/files/school/Projects/Embedded/data $ sudo sqlite3 LED_DB
SQLite version 3.7.13 2012-06-11 02:05:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
LED_STATUS
sqlite> .schema
CREATE TABLE LED_STATUS(Status Text, DateAndTime stamp Text);
sqlite> █
```

Figure 7: Name of Database, Table and its table layout used for LED program.

Displaying SQLite Database with PHP

The code shown below was the attempt to make the data in the LED_STATUS table in LED_DB database to display on a website.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>
5    Ivan's Embedded Linux Project
6  </title>
7  </head>
8  <body>
9    <h1>House Lights</h1>
10   <p>
11     Here you can monitor the lights in your house to see if and when they were turned on.
12   </p>
13   print "<table border=1>"; <!--Table created-->
14   print "<tr><td>Status</td><td>DateAndTimestamp</td></tr>";
15   <?php
16   try{
17     $db = new PDO('sqlite:/home/pi/files/school/Projects/Embedded/data/LED_DB'); //accessing LED_DB
18     if ($db == false)
19     {
20       die("Unable to connect to Sqlite3 DataBase<br>");
21     }
22
23     $result = $db->query('SELECT * FROM LED_STATUS'); //querying database
24     foreach($result as $row) //displaying database row by row onto website through the table created earlier
25     {
26       print "<tr><td>".$row['Status']. "</td>";
27       print "<td>".$row['DateAndTimestamp']. "</td></tr>";
28     }
29     print "</table>";
30
31     $db = NULL; //close database
32   }
33   catch(PDOException $e)
34   {
35     print 'Exception : '.$e->getMessage();
36   }
37   ?>
38
39 </body>
40 </html>
41
```

Figure 8: House Lights.php code.

Figure 9 displays the output of the PHP code. As seen below the PHP code is not displaying the data from the LED_STATUS table at all.

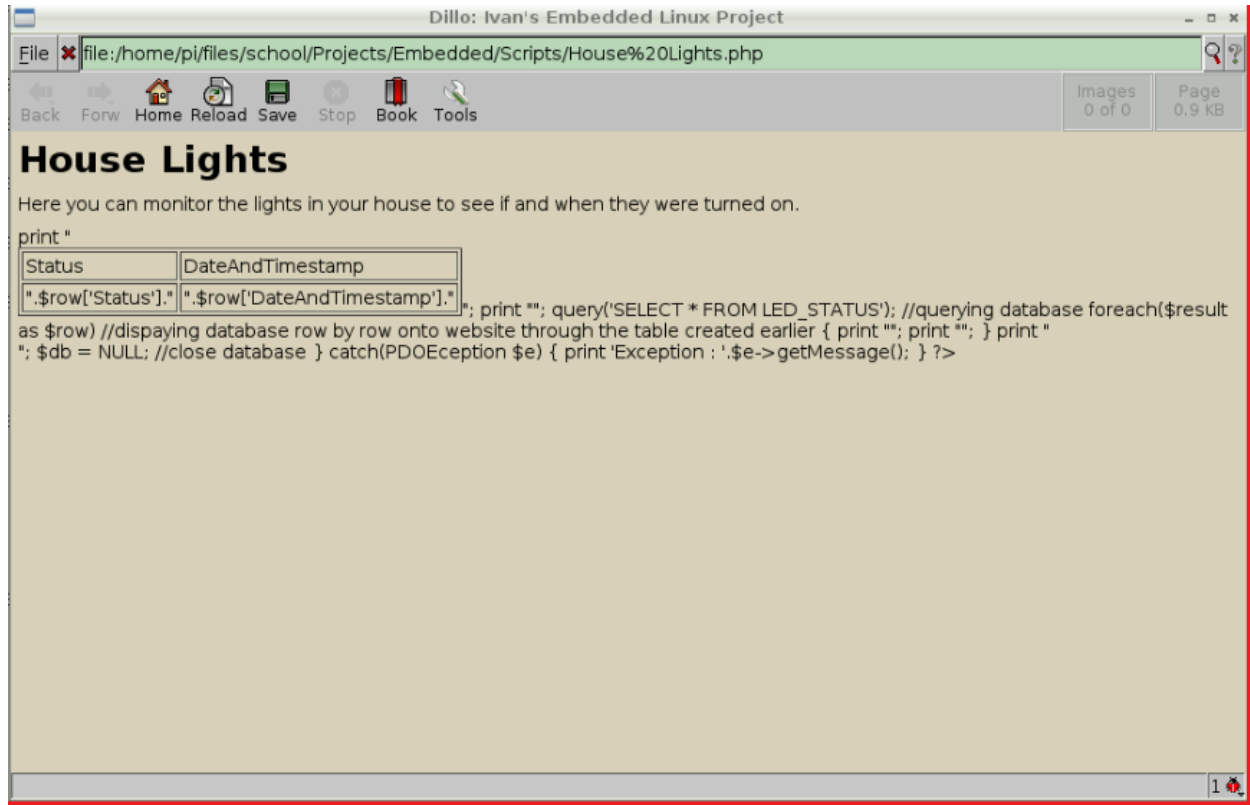


Figure 9: House Lights.php code output.

Problems that Occurred

(**) A hardware problem that existed once all the pieces were physically at my disposal was the use of the Temperature sensor. Temperature is an analog signal and microcontrollers such as the Arduino and Raspberry pi cannot read analog signal without the use of a ATD (Analog to digital converter). The ATD takes in the analog signal(sinusoidal) and converts it into a digital signal(square wave). The Arduino's GPIO (general purpose input / outputs) ports are ATD. However, the Temperature sensor would have needed a ribbon cable from the Raspberry pi's GPIO pins to connect to the Arduino leaving me with the inability to connect the other sensors. One option was to buy a IC(Integrated Circuit) chip for the Raspberry Pi so its GPIO pins could be converted to ATD. However that would have cost additional money which was already a constraint. Another option that was occurred to me was to buy a 2 in 1 keyboard in order to have 1 free USB port on my Raspberry Pi in which I could connect the Arduino to and have the Arduino and the Raspberry Pi communicate by running all Arduino programs on the pi and not need to worry about serial communication and obtain the sensor data by just setting variables and passing it to the database. However due to this project already costing over \$100 and money being a constraint, it was not feasible to try this method because it would have left me spending more money and an extra keyboard. Due to the University's Internet's firewalls and port restrictions while living on campus sshing into my Pi was not really an option because it would have left me with the possibility of not being able to reproduce my project anywhere else that was not my room in the residence hall. Although the only place I found that I could access my Raspberry Pi via ssh was in my room it sometimes was not cooperating in the sense that on the host computer I was not allowed to do certain tasks sometimes like anything internet related, or package related such as sqlite3 because I was given errors such as *packages not install please try sudo apt-get upgrade*. SSHing into my Pi was unreliable and a direct connection was more feasible.

(*1) Originally `sudo apt-get install sqlite` was installed which produced a download package of Sqlite2 which through errors and research, was founded to not have database functionality so it was replaced with Sqlite3.

(*2) Due to the little knowledge of both vim and nano on the command line, my efforts in proceeding with this project due to time constraints led me to switch over to the GUI of the OS. Once the code for the website was being implemented, *Leafpad*, a program preinstalled with the Rasbian OS was used to write all HTML, and PHP code. However, due to the fact that all HTML and PHP code was using HTML5 syntax layout when adding the line of code to the top of all HTML AND PHP files `<!DOCTYPE html>` . *Leafpad* produced the following error :

HTML WARNING: line 1, Document follows HTML5 working draft; treating as HTML4

Which lead me to believe that the HTML5 syntax being used was not being read and producing errors such as:

HTML WARNING: line 13, attribute lacks closing quote

HTML WARNING: line 21, <th> or <td> outside <table>

HTML WARNING: line 21, <th> or <td> outside <table>

HTML WARNING: line 21, unexpected closing tag: </tr>

HTML WARNING: line 26, <th> or <td> outside <table>

HTML WARNING: line 27, <th> or <td> outside <table>

HTML WARNING: line 30, unexpected closing tag: </table>

To bypass this error, *Geany*, a php IDE was installed and solved the solution to this problem. It also helped when running the same php scripts on *Leafpad* on *Geany* because *Leafpad* does not number the lines of code and *Geany* does so it was easier to find and debug *Leafpad* problems using *Geany* as seen in **Figure 10**. Dillo is one of the many web browser preinstalled on the Raspberry Pi and was used to see the actual output of my HTML/PHP file.

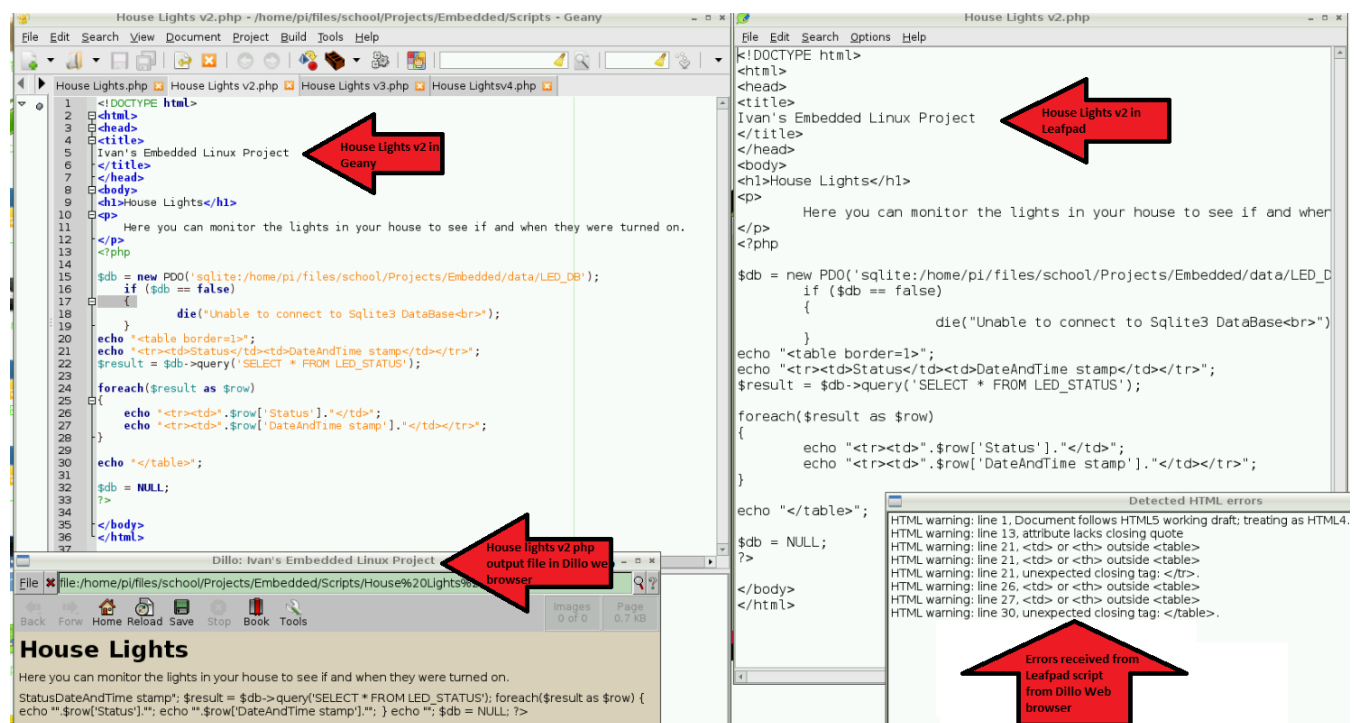


Figure 10: Geany (Top Left) , Dillo (Bottom Left) , Leafpad (Top Right) and Error window (Bottom Right)

(*3) If you are downloading a file from the internet and the data within the file is showing up as empty. What you need to do is right click on the file and click on 'save link as'. Before importing the desired file make sure the desired file is in the same directory as your database is located, SQLite will not be able to locate the file if it is not in the same directory as the database.

(*4) When checking a desired database, you must first use the terminal to go to the directory in which the database is in. If you are not in the same directory as where the database was created originally and execute the same instruction shown in **Step 1** then you will be creating a new database with the same name in a different location, instead of opening up the previous database with the already created table and imported data.