ΕΞΩΦΥΛΟ

Acknowledgements

## Abstract

Contents

[Abstract 2](#_Toc399804255)

[1. Introduction 7](#_Toc399804256)

[1.1. Motivation 7](#_Toc399804257)

[1.2. Purposes and targets 7](#_Toc399804258)

[1.3. Project Structure 7](#_Toc399804259)

[2. Implementation Methodology 7](#_Toc399804260)

[3. Project Plan 7](#_Toc399804261)

[3.1. State Of The Art 8](#_Toc399804262)

[4. Basic Part 9](#_Toc399804263)

[4.1. Hardware Components 9](#_Toc399804264)

[4.1.1. Raspberry PI 9](#_Toc399804265)

[Web Server 10](#_Toc399804266)

[Advantages of Raspberry PI being a Web Server 12](#_Toc399804267)

[4.1.2. Arduino 13](#_Toc399804268)

[4.1.2.1. Arduino Mega 2560 R3 13](#_Toc399804269)

[4.1.2.2. Arduino Ethernet Shield 14](#_Toc399804270)

[4.1.3. Sensors 15](#_Toc399804271)

[4.1.3.1. Relative Humidity and Air Temperature (DHT22 or RHT03) 15](#_Toc399804272)

[4.1.3.2. Air Pressure and Temperature (BMP085) 17](#_Toc399804273)

[4.1.3.3. Precipitation 20](#_Toc399804274)

[4.1.3.4. Wind Speed 22](#_Toc399804275)

[4.1.3.5. Wind Direction 23](#_Toc399804276)

[4.1.4. Prototype Architecture 25](#_Toc399804277)

[4.2. In Depth Project Analysis 26](#_Toc399804278)

[4.2.1. Arduino Sketch [23] 26](#_Toc399804279)

[4.2.1.1. Sensors Values Readings 27](#_Toc399804280)

[4.2.1.2. HTTP Request 35](#_Toc399804281)

[4.2.1.3. Pan Tilt Camera Mount Control 37](#_Toc399804282)

[4.2.2. Android Application [28] 39](#_Toc399804283)

[Charts Creation Procedure 40](#_Toc399804284)

[Current Weather Conditions Report 44](#_Toc399804285)

[Min / Max Report 45](#_Toc399804286)

[Data Presentation in Table Layout 45](#_Toc399804287)

[4.2.3. Web Site 48](#_Toc399804288)

[4.2.4. Server – Side PHP Scripts 49](#_Toc399804289)

[Storing Data in **Database** 49](#_Toc399804290)

[Android Application Fetching Script 50](#_Toc399804291)

[References 52](#_Toc399804292)

**Nomenclature**

* MC / MCU Microcontroller
* RH Relative Humidity
* °C Degrees Celsius
* hPA Hecto Pascal
* EMC Electromagnetic Compatibility
* URL Uniform Resource Locator
* URI Uniform Resource Identifier
* TCO Total Cost of Ownership
* RPI Raspberry PI
* OS Operating System
* ISP Internet Service Provider
* DDNS Dynamic Domain Name System
* WDT Watchdog Timer

[Figure 1 - The Arduino Mega 2560 R3 Microcontroller Board 13](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802512)

[Figure 2 - The Arduino Ethernet Shield 14](#_Toc399802513)

[Figure 3 - RHT03 15](#_Toc399802514)

[Figure 4 - RHT03 Communication Process 17](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802515)

[Figure 5 - BMP085 Sensor 17](#_Toc399802516)

[Figure 6 - BMP085 Application Circuit 18](#_Toc399802517)

[Figure 7 - Calculation Process of Pressure and Temperature for BMP085 19](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802518)

[Figure 8 - BMP085 Temperature and Pressure Reading Process 19](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802519)

[Figure 9 - Bouncing problem at a digital input. MCU will read nine state changes instead of one 21](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802520)

[Figure 10 – Rain Reading And Switch Debouncing Implementation 21](#_Toc399802521)

[Figure 11 - Connection of the Rain Gauge to the Arduino Mega 22](#_Toc399802522)

[Figure 12 - Anemometer Reading And Switch Debouncing Implementation 22](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802523)

[Figure 13 - Connection of the Anemometer to the Arduino Mega 23](#_Toc399802524)

[Figure 14 - Wind Vane resistor arrangement 23](#_Toc399802525)

[Figure 15 - Wind Vane Powering Up 24](#_Toc399802526)

[Figure 16 - Connection of the Wind Vane to the Arduino Mega 25](#_Toc399802527)

[Figure 17 - Schematic wiring of all components 26](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802528)

[Figure 18 - Function to read 1 byte from the bmp085 28](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802529)

[Figure 19 - Function to read 2 bytes from the bmp085 28](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802530)

[Figure 20 - BMP085 Calibration 28](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802531)

[Figure 21 - BMP085 Read Uncompensated Pressure Function 29](#_Toc399802532)

[Figure 22 - BMP085 Read Uncompensated Temperature Function 30](#_Toc399802533)

[Figure 23 - BMP085 Final Temperature Calculations 31](#_Toc399802534)

[Figure 24 - BMP085 Final Pressure Calculations 32](#_Toc399802535)

[Figure 25 - Weather Meters Configuration 33](#_Toc399802536)

[Figure 26 - Rain Final Calculation 33](#_Toc399802537)

[Figure 27 - Wind Final Calculation 33](#_Toc399802538)

[Figure 28 - URL contruction 35](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802539)

[Figure 29 - HTTP Request Execution 36](#_Toc399802540)

[Figure 30 - HTTP Response 37](#_Toc399802541)

[Figure 31 - Remote Camera Control 37](#_Toc399802542)

[Figure 32 - Generated HTTP Request on button click 38](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802543)

[Figure 33 38](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802544)

[Figure 34 - Servo Control Example 38](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802545)

[Figure 35 - Android App Icon 39](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802546)

[Figure 36 - Splash Screen 39](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802547)

[Figure 37 - Start Screen 39](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802548)

[Figure 38 - Method to construct URLs 40](#_Toc399802549)

[Figure 39 - Android HTTP Request 40](#_Toc399802550)

[**Figure 40 - HTTP Response Handling** 41](#_Toc399802551)

[**Figure 41 - Filling JSONobject Array, one index per sensor** 41](#_Toc399802552)

[Figure 42 - Example of JSON Elements 42](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802553)

[Figure 43 - JSON resolver method 42](#_Toc399802554)

[Figure 44 - Method to handle graph drawing 43](#_Toc399802555)

[Figure 45 - Current Weather Conditions Report 44](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802556)

[Figure 46 - Min / Max Report View 45](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802557)

[Figure 47 - ArrayList that holds ArrayLists 45](#_Toc399802558)

[Figure 48 - Table Layout Creation 46](#_Toc399802559)

[Figure 49 - Table Layout View 46](#_Toc399802560)

[Figure 50 - Adding Data to Table Layout 47](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802561)

[Figure 51 - Script to store data to database 49](#_Toc399802562)

[Figure 52 - Dynamic SQL Query Construction 50](#_Toc399802563)

[Figure 53 - URL example 50](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc399802564)

[Figure 54 - SQL Table to JSON 50](#_Toc399802565)

[Figure 55 - SQL Query to select every n-th row 51](#_Toc399802566)

[Table 1 - RHT03 Technical Specifications 12](#_Toc394960346)

[Table 2 - RHT03 Pin Sequence Number (1, 2, 3, 4 from left to right) 12](#_Toc394960347)

[Table 3 - Main Properties of the BMP085 13](#_Toc394960348)

[Table 4 - Measurement modes of the BMP085 and resolution effects 16](#_Toc394960349)

[Table 5 - BMP085 Pin Connections to Arduino Mega 16](#_Toc394960350)

[Table 6 - Analog Voltage output with corresponding direction 20](#_Toc394960351)

[Table 7 - BMP085 Calibration Coefficients 24](file:///D:\Dropbox\GraduationalProject\SenseBox\Report2%20(Autosaved).docx#_Toc394960352)

# Introduction

The purpose of this project is to measure the weather and virtualize the data. The data gathering process is implemented upon Arduino, an open source hardware prototyping platform, which allows an easy implementation of sensors and interactive elements. Data gathered by Arduino are stored on a database and on the next level these data are used to make graphs on different platforms. Also the project included a live weather camera that rotates to spot almost everywhere the user wants.

## Motivation

Building my own weather station was an idea that came into my mind about two years ago. I first thought of building it for educational purposes and as a hobby. Later on I realized that the project could also serve as my final year’s project so I declared it.  
  
My interested on weather data has been inherited from my favorite sport – Sailing. For sailors observing the weather is a major part of their daily routine so this was the icing on the cake to begin!

## Purposes and targets

Sdasdasd

## Project Structure

The project is split in four parts.

1. The Arduino part in which data are gathered.
2. The server part that hosts the server-side PHP code that is used to store and retrieve data from the database.
3. The Android Application part that visualizes the data.
4. The Web Site part that also visualizes the data.

# Implementation Methodology

Cvsdfsdf

# Project Plan

The project has been implemented using the latest technologies available for its needs. The next chapter state of the art provides a more in depth analysis about the background of the project.

## State Of The Art

* **Open – Source Hardware**

Open - Source has become a popular expression, but mostly with regard to software. The principles and definitions of Open-Source Hardware (OSHW) are closely related to those of Open-Source Software (OSS) from the Open Source Initiative [1, 2]. Interested people can study, modify, distribute, make and sell designs based on those of OSHW products [3]. Through open development, people get the possibility to learn and understand how OSHW works, so that they are able to control and modify their technology. Machines, devices, or other physical things produced under the OSHW license must comply with the definitions of the Open Source Hardware Association [1].

* **Arduino Platform**

Arduino is a prototyping platform containing a microcontroller board (MC) as core element, a programming language and an integrated development environment (IDE) [4]. OSHW MCs from Arduino are based on ATmega8, ATmega168, ATmega2560 and latest boards on [ATmega32u4](http://arduino.cc/en/Main/ArduinoBoardYun), [Sitara AM335x](http://arduino.cc/en/Main/ArduinoBoardTre). They can be used to develop interactive prototypes, which are using input from sensors to control output devices connected to the same board. The capabilities of the MCs allow several methods for in- and outputs of signals.

The simplified IDE with a wire-based programming language also allows beginners to realize complex projects in short time. Documentation of the IDE and the language reference can be found on the Arduino homepage [5]. All code samples are released into the public domain. Additionally, the language can be extended with C++ libraries to enable more functionality. All official software tools from Arduino are published under the OSS license and are platform-independent.

The well documented Arduino hard- and software makes it even possible to rebuild MCs by oneself. However, pre-assembled Arduino boards are relatively inexpensive compared to other commercial MC platforms available on the market

* **Raspberry PI**

The Raspberry PI is a credit – card – sized single board computer developed mainly for educational purposes. As core element it has a Broadcom BCM2835 system on a chip (SoC) [9] which includes an ARM11 family 700MHz processor, a VideoCore IV GPU and 512MB RAM.

The Raspberry PI foundation provides a lightweight version of Debian (Raspbian) and Arch Linux ARM distribution as official operating systems.

Tools are available for Python as the main programming language, C, Java and Perl.

For the needs of the project, the RPI is used as server, hosting the script files that are used to store and retrieve weather data in the database as well as hosting the web page. Later it is explained why the RPI is the perfect small server.

* **Automated Weather Stations**

Automated weather stations (AWS) are meteorological stations, which perform climatic observations and data transmission automatically [6]. Main advantages compared to human weather observations are, that measurements, read out by a central data-acquisition unit, are more reliable and can be performed much more frequently. Sensors of an AWS are operated by a microprocessor. It allows exact sampling of sensor data and processing for averaging or filtering of the samples. The result will be a series of observations which are representative over a limited area.

* **Android Platform**

Android is a mobile operating system based on the Linux Kernel that is currently developed by Google. It powers hundreds of millions of mobile devices around the world. It is the largest installed base of any mobile operating system platform.

Android’s source code is released by Google under open source licenses (Apache License 2.0 and GNU General Public License 2.0 [7, 8]).

Its open nature has encouraged a large community of developers and enthusiasts to use the open – source code as a foundation for community-driven projects, which adds new features for advanced users.

* **Google Chart API**

The Google Chart API [10] is a tool that lets people easily create a chart from some data and embed it in a web page. Google creates a [PNG](http://en.wikipedia.org/wiki/Portable_Network_Graphics) image of a chart from data and formatting parameters in an [HTTP](http://en.wikipedia.org/wiki/HTTP) request. Many types of charts are supported, and by making the request into an image tag, people can simply include the chart in a web page.

# Basic Part

//TODO REQUIREMENT ANALYSIS

## Hardware Components

The Microcontroller controls the data storage process and sampling frequency. Sampling rates of sensors are depending on the time constant value that we set. Sensors and Microcontroller functionality are explained in depth in the upcoming section 4.1.2.

A server is a running instance of a Software capable of accepting requests from the client and giving responses accordingly. In the case of the project the Single – Board Computer Raspberry PI operates as the server. It is a critical part for the project being functional, since it is responsible for storing the data to the database it hosts, for distributing the data where requested and hosting the website (web server). In the next section (4.1.1) more about the PI can be found.

### Raspberry PI

The Raspberry PI holds a critical place in the project. It has to perform several tasks that without them the system is useless. These tasks are the following.

* Web Server
* Website Hosting
* WordPress Hosting
* Database Hosting
* Dynamic DNS Update Client
* Time Lapse Photography

#### Web Server

Figure - Apache Logo

The primary function of a web server is to store, process and deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to text content.

A web browser or web crawler, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so. The resource is typically a real file on the server's secondary storage, but this is not necessarily the case and depends on how the web server is implemented.

While the primary function is to serve content, a full implementation of HTTP also includes ways of receiving content from clients. This feature is used for submitting web forms, including uploading of files or receiving data to be stored on a database.

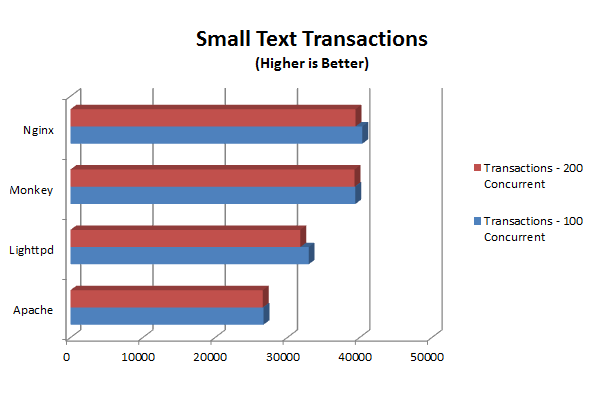
Many generic web servers also support server-side scripting using Active Server Pages (ASP), PHP, or other scripting languages. This means that the behavior of the web server can be scripted in separate files, while the actual server software remains unchanged. Usually, this function is used to create HTML documents dynamically ("on-the-fly") as opposed to returning static documents. The former is primarily used for retrieving and/or modifying information from databases. The latter is typically much faster and more easily cached but cannot deliver dynamic content.

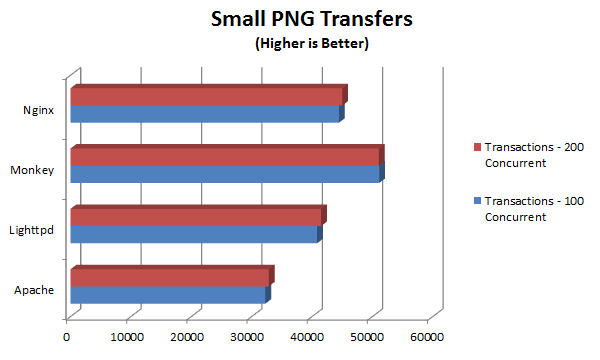
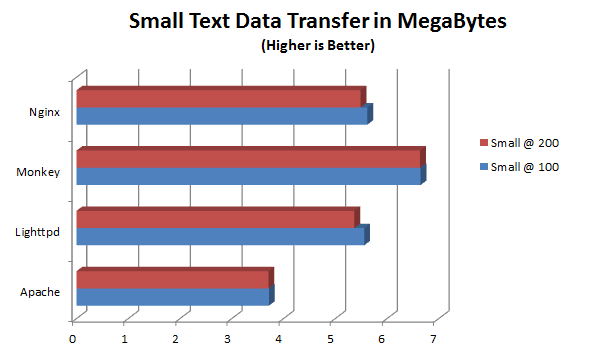
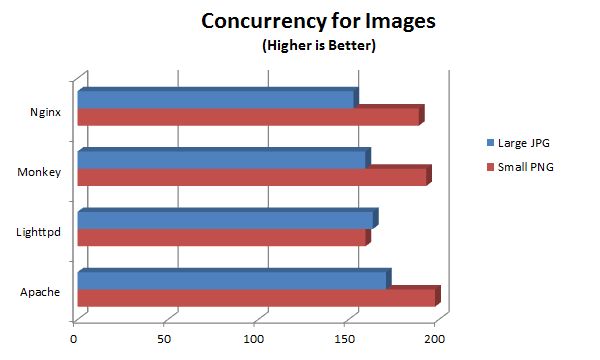
Web servers are not always used for serving the World Wide Web. They can also be found embedded in devices such as printers, routers, webcams and serving only a local network. The web server may then be used as a part of a system for monitoring and/or administering the device in question. This usually means that no additional software has to be installed on the client computer, since only a web browser is required (which now is included with most operating systems).

These was some general information about Web Servers. More specifically, for the project the Apache HTTP Server is used. Colloquially called Apache is notable for playing a key role in the initial growth of the World Wide Web. It is the most popular HTTP server in use since April 1996, serving more than 100 million Web sites.

Apache was chosen among its main competitors Nginx, LightHttpd and Monkey, even if it lacks on speed as shown in the tests, because of its variety of features and the huge community.

The speed tests in the next page compare how each Web Server behave on the rPI.





##### Advantages of Raspberry PI being a Web Server

The Raspberry PI is ideal for being a Home Server. Obviously its ARM11 processor cannot handle a lot of traffic but the advantages are so many that overlay that.

Below there is a list of some of the advantages of the PI being a Server.

* **Power consumption** - The Pi draws about five to seven watts of electricity. This is about one tenth of what a comparable full-size box can use. Since servers are running constantly night and day, the electrical savings can really add up. I have calculated that the basic Pi kit (Pi board, case, and power supply) will pay for itself with about one year's worth of electricity savings if it's left to run 24x7x365.
* **No moving parts** - The Pi uses an SD card for storage, which is fast and has no moving parts. There are also no fans and other things to worry about. A Class 10 SD card is usually the best performing compared to lower class cards, but this will mainly only affect boot time where there is the most I/O.
* **Small form factor** - The Pi (with a case) is smaller than an adult’s palm. A comparable full-size box is a lot times bigger. This means the Pi can be integrated inside of devices, too.
* **Status lights** - There are several status lights on the Pi's motherboard. With a clear case one can see NIC activity, disk I/O, power status, etc.
* **Huge community support** - The Pi has phenomenal community support. Support can be obtained quite easily for the hardware and/or GNU/Linux software that runs on the Pi mainly in user forums, depending on the GNU/Linux distribution used.
* **Overclocking capability** - The Pi can be overclocked if there are performance problems with the application used. Especially with a heat sink installed, the risk of damaging the board is almost non-existing.
* **Software** - The Raspberry Pi primarily uses Linux kernel-based operating systems. That makes it even more attractive due to all the benefits that Linux-based OS has in the field of servers. Some of these are:
* Stability – Linux systems are well known for their ability to run for years without failure.
* Security - Linux is also innately more secure than Windows is, whether on the server, the desktop or in an embedded environment. That's due largely to the fact that Linux, which is based on Unix, was designed from the start to be a multiuser operating system. Only the administrator, or root user, has administrative privileges, and fewer users and applications have permission to access the kernel or each other. That keeps everything modular and protected
* Total cost of Ownership – There is no beating Linux’s TCO, since software is generally free.
* Freedom – With Linux, there is no commercial vendor trying to lock you into certain products or protocols. Instead, it allows the user to mix and match and choose what works best for each job.
* **Affordable** - Compared to other similar alternatives, the Pi (revision B) offers the best specs for the price. It is one of the few devices in its class that offers 512 MB of RAM.

#### Website Hosting

//TODO

#### D:\Dropbox\GraduationalProject\SenseBox\Raw Data\Pics\wp_logo.pngWordPress Hosting

WordPress [34] is a free and open source blogging tool and a content management system (CMS) based on PHP and MySQL. Features include a plugin architecture and a template system. WordPress was used by more than 22.0% of the top 10 million websites as of August 2013. WordPress is the most popular blogging system in use on the Web, at more than 60 million websites.

It was first released on May 27, 2003, by its founders, Matt Mullenweg and Mike Little, as a fork of b2/cafelog. The license under which WordPress software is released is the GPLv2 (or later) from the Free Software Foundation. [Source: Wikipedia]

WordPress was chosen among the others CMS because it encapsulates a variety of advantages. Some of these are the following.

* Ease of use – WordPress is very easy to use out of the box.
* It is free to use - In addition to the freedom from licensing costs and restrictions, with WordPress, you get full access to the source code. This means that any changes can be made without reliance on any one particular web developer.
* It is well supported – WordPress is well supported by its huge community. The support comes mainly from its own Support Page and StackOverflow.
* It is continually improved - WordPress is constantly being updated and maintained. WordPress has developers all over the world contributing to its continual improvement. It is already a great system and it’s getting even better every day
* Plethora of Plugins

Hosting and running WordPress is intense for the Raspberry PI due to its limited processing power. As soon as you start using WordPress its CPU usage instantly raises to 50% and more. Although, even if it is not running super-fast, it is enough for educational and experimental purposes, like this.

#### Dynamic DNS Update Client

The main problem when hosting a Home Web Server are Dynamic IPs. This means that the public IP of a broadband connection changes. How often it changes it depends from the ISP.

ISPS prefer Dynamic IP over Static IP [more on A.1] because addresses are much more difficult to manage. Even most "static" IP addresses are assigned dynamically through DHCP using static reservations, but with either a statically configured IP address or a DHCP static reservation, if the client connection device changes (hardware replacement, upgrade, etc), then someone needs to make a change either on that device or in the DHCP configuration. Dynamic assignment avoids this problem.

Using DHCP also simplifies the network design and allows for easier changes later. If the ISP chooses to make changes to the way they are using their IP addresses, they can do so without making manual changes on client connection devices. Examples of such changes could include any of the following (or others):

* Increasing the size of the network (moving from a /24 network to a /23)
* Decreasing the size of the network (moving from a /24 to a /25)
* Changing the network in use entirely (moving from 10.0.0.0/24 to 192.168.0.0/24)
* Changing the gateway IP address
* Changing the DNS server addresses

DHCP also allows for the ISP to provide additional configuration to some devices. For instance, if they want to provide images/configuration on a TFTP server, this can be provided to the device through DHCP.

These changes on IPs present a problem if the customer wants to provide a service to other users on the Internet, such as a web service. As the IP address may change frequently, corresponding domain names must be quickly re-mapped in the DNS, to maintain accessibility using a well-known URL. That problem Dynamic DNS service providers comes to solve.

Dynamic DNS (DDNS) is a method of automatically updating a name server in the Domain Name System (DNS), often in real time, with the active DNS configuration of its configured hostnames, addresses or other information.

The term is used to describe two different concepts. At the administration levels of the Internet, "dynamic DNS updating" refers to systems that are used to update traditional DNS records without manual editing. These mechanisms are explained in RFC 2136, and use the TSIG mechanism to provide security.

Another kind of dynamic DNS permits lightweight and immediate updates to its local database, often using a web-based mechanism. It is used to resolve a well-known domain name to an IP address that may change frequently. It provides a persistent addressing method for devices that change their location or configuration.

For the needs of the project No – IP service provider is used. No – IP offers for free a client that continually checks for IP address changes in the background and automatically updates the DNS at No – IP whenever it changes. The client is a very lightweight written in C.



Figure 2 - DDNS client

#### Time Lapse Photography

Raspberry PI also handles the creation of the daily Time Lapse video.

Time-lapse photography is a technique whereby the frequency at which film frames are captured (the frame rate) is much lower than that used to view the sequence. When played at normal speed, time appears to be moving faster and thus lapsing. For example, an image of a scene may be captured once every second, then played back at 30 frames per second; the result is an apparent 30 × (24 [cinema] / 25 [PAL] / 30 [NTSC] frames per second) times speed increase. Time-lapse photography can be considered the opposite of high speed photography or slow motion.

Processes that would normally appear subtle to the human eye, e.g. the motion of the sun and stars in the sky, become very pronounced.

This procedure occupies the rPI around the clock, since it must capture pictures, through the USB camera, every two minutes and every day at midnight it encodes the pictures into a video. More on this will be analyzed on chapter 4.2.3.

### Arduino

The Arduino and the Ethernet Shield can be imagined as the ‘brain’ of the weather station. It provides all the core functionality from sensor manipulation up to data logging. The Arduino Mega and the Arduino Ethernet Shield are introduced below.

#### Arduino Mega 2560 R3

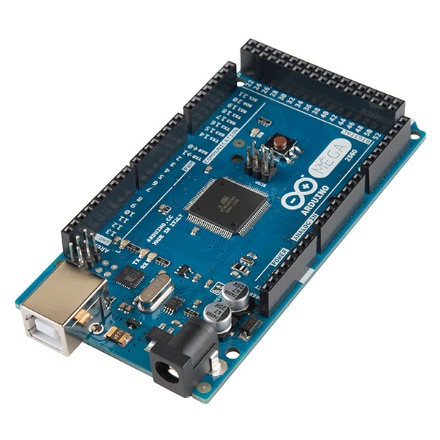


Figure 3 - The Arduino Mega 2560 R3 Microcontroller Board

Arduino Mega 2560 [11] is the board that is powered by ATmega2560. Its main advantages compared to other Arduino boards are the clock speed of 16MHz, the flash memory of 256Kbytes of the ATmega2560 chip and the 8Kbytes SRAM. The large flash and SRAM memories offer the Arduino vast of space for more interactive components to be added in future improvements.

The Arduino Mega provides 54 digital pins. They can be used as input or output and are operating at 5V. Each pin can provide or receive a maximum of 40mA and has an internal pull-up resistor of 20 – 50 kΩ. The pull-up resistor is disconnected by default but they can be accessed from software by setting the **pinMode()** as **INPUT\_PULLUP**. This effectively inverts the behavior of the INPUT mode, where HIGH means the sensor is off, and LOW means the sensor is on. In addition, some pins have specialized functions:

* **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).**

Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.

* **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).**

These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.

* **PWM: 2 to 13 and 44 to 46.**

Provide 8-bit PWM output with the [analogWrite()](http://arduino.cc/en/Reference/AnalogWrite) function.

* **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).**

These pins support SPI communication using the SPI library.

* **LED: 13.**

There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

* **TWI: 20 (SDA) and 21 (SCL).**

Support TWI communication using the [Wire library](http://arduino.cc/en/Reference/Wire).

(BMP085 sensor uses the Wire protocol to communicate with Arduino)

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 210 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

#### Arduino Ethernet Shield

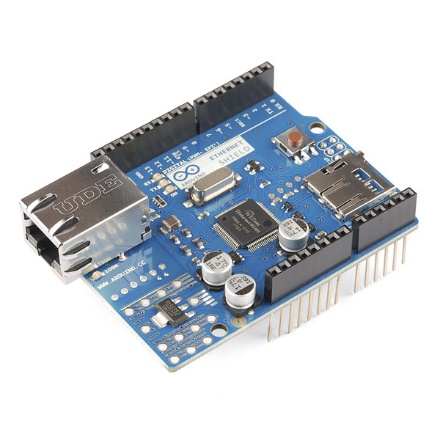


Figure 4 - The Arduino Ethernet Shield

The Ethernet shield [12] is powered by a Wiznet W5100 ethernet controller and also includes a micro SD card slot for storage purposes. It is made exclusively for Arduino boards and connects over the SPI port.

The Wiznet W5100 provides a network (IP) stack capable of both TCP and UDP. It supports up to four simultaneous socket connections. The ethernet shield connects to an Arduino board using long wire-wrap headers which extend through the shield. This keeps the pin layout intact and allows another shield to be stacked on top.

The Ethernet Shield has a standard RJ-45 connection, with an integrated line transformer and Power over Ethernet enabled.

It worth mentioning that pins SPI pins cannot be used as digital in or output any more when using the Ethernet Shield as they are being used by the SPI bus. Pins 4, 10 are used for the SD card and cannot be used as well. All the other pins are free to be used.

The Ethernet Shield is used in the project for data logging and as server for controlling the pan-tilt camera mount.

To integrate functionalities of the Ethernet Shield into the whole system, Ethernet [13] and SD [14] libraries are used.

### Sensors

In this section the functionality and the characteristics of each sensors used by the prototype will be analyzed. In addition, connection and communication examples are given in this section.

#### Relative Humidity and Air Temperature (DHT22 or RHT03)

The RHT03 sensor from is made of two parts, a capacitive humidity sensor and a thermistor. There is also a basic chip inside that does some analog to digital conversion and offers a digital signal with the temperature and humidity.

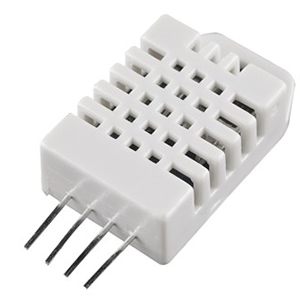


Figure 5 - RHT03

The RHT03 was chosen because of its high Resolution and Accuracy. Technical Specifications are shown in the table below [15].

|  |  |
| --- | --- |
| **Model** | **DHT 22 also known as RHT03** |
| Power Supply | 3.3 – 6V DC |
| Output Signal | Digital Signal via MaxDetect 1-wire bus |
| Sensing Element | Polymer Humidity Capacitor |
| Operating Range | Humidity: 0 – 100% RH  Temperature: - 40 ~ 80 °C |
| Accuracy | Humidity: +- 2% RH (Max +-5% RH)  Temperature: +- 0.5 °C |
| Resolution or Sensitivity | Humidity: 0.1% RH  Temperature: 0.1 °C |
| Repeatability | Humidity: +- 1% RH  Temperature: +- 0.2 °C |

Table 1 - RHT03 Technical Specifications

|  |  |
| --- | --- |
| Pin | Function |
| 1 | Power Supply |
| 2 | Data – Signal |
| 3 | NULL |
| 4 | Ground |

Table 2 - RHT03 Pin Sequence Number (1, 2, 3, 4 from left to right)

##### Communication and signal

Wire Bus is used for communication between the MCU and RHTY03.

**Illustration of MaxDetect’s wire bus:**

Data is comprised of integral and decimal part, the following is the formula for data.

* 8bit integral Relative Humidity data
* 8bit decimal Relative Humidity data
* 8bit integral Temperature data
* 8bit decimal Temperature data
* 8but checksum

When MCU send start signal, RHT03 change from standby-status to running-status. When MCU finishs sending the start signal, RHT03 will send response signal of 40-bit data that reflect the relative humidity and temperature to MCU. Without start signal from MCU, RHT03 will not give response signal to MCU. One start signal for one response data from RHT03 that reflect the relative humidity and temperature. RHT03 will change to standby status when data collecting finished if it don't receive start signal from MCU again.

Below one can find the overall communication process. The interval of whole process must beyond two seconds.

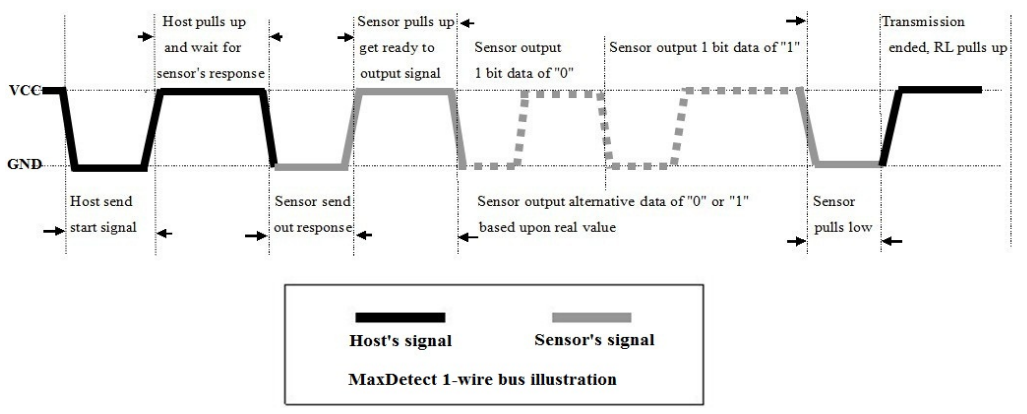


Figure - RHT03 Communication Process

#### Air Pressure and Temperature (BMP085)

For air pressure readings and temperature the digital sensor BMP085 from Bosch Sensortec is used [16]. It was chosen for the prototype because of its good accuracy through individual precise calibration and its long term stability feature. The main characteristics are summarized in the table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parameter** | **Condition** | **Min** | **Typical** | **Max** | **Unit** |
| Resolution | Typical | 0.04 | 0.01 | 0.01 | hPA |
| Accuracy | Typical |  | +- 1.0 |  | hPA |
| Response Time | Mode | 4.5 |  | 25.5 | Ms |
| Operation Range |  | 700 |  | 1100 | hPA |
| Supply Voltage |  | 1.8 | 2.5 | 3.6 | Volts |

Table 3 - Main Properties of the BMP085

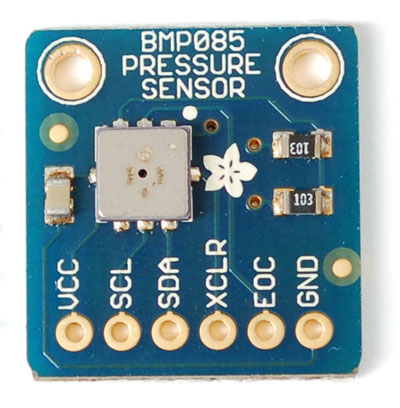


Figure 6 - BMP085

Figure 7 - BMP085 Sensor

It is based on piezo - resistive

Technology for EMC robustness, high accuracy and linearity as well as long term stability. It is designed to be connected directly to a MCU via the I2C bus. The pressure and temperature data has to be compensated by the calibration data of the E2PROM of the BMP085.

The sensor consists of a piezo – resistive sensor, an analog to digital converter and a control unit with E2PROM and a serial I2C interface. It delivers the uncompensated value of pressure and temperature. The E2PROM has stored 176 bit of individual calibration data. This is used to compensate offset, temperature dependence and other parameters of the sensor.

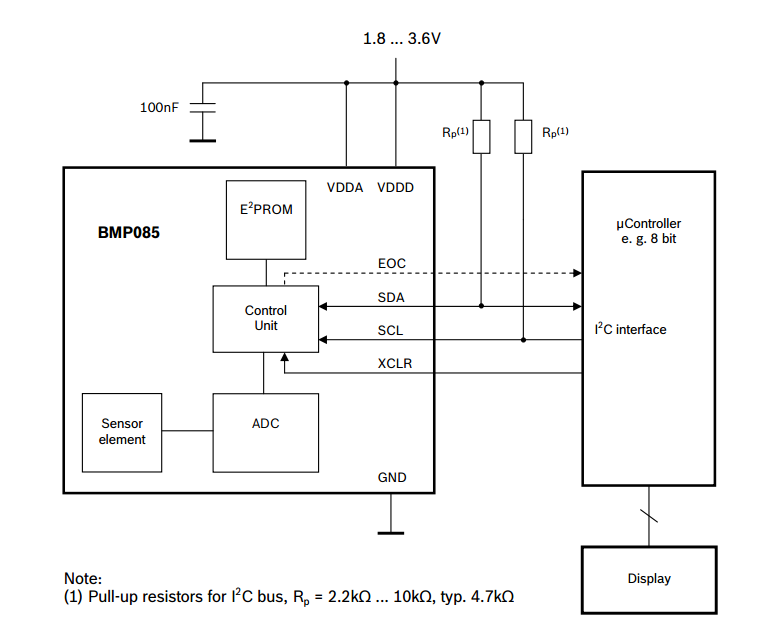


Figure 8 - BMP085 Application Circuit

The I²C interface and a read-only-memory register (E²PROM) are part of the control unit of the BMP085. In the E²PROM eleven 16bit calibration coefficients are stored and used to compensate offset of temperature and pressure readings. As temperature is a factor for air pressure calculation, it has to be known to calculate the true pressure. Reading of the temperature can be done with the piezo-resistive sensor as well.

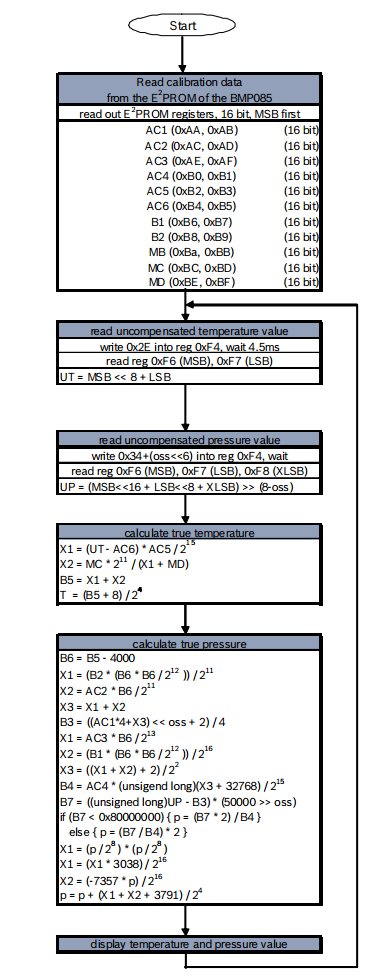


Figure 9 - Calculation Process of Pressure and Temperature for BMP085

A measurement sequence with the complete algorithm is shown in Figure 9. It has to be implemented to the MCU according to this order.

After the MC has sent a start sequence, the calibration data is requested from the E²PROM registers. The 16 constants need to be read out only once and are stored on the MCU. Then the MCU has to wait for uncompensated temperature (UT) and pressure (UP) readings from the sensors. The stored calibration data is now used to calculate temperature in °C and pressure in Pa. After applying the algorithms, the sensor waits for the next measurement command.

As additional factor for true pressure calculation, an oversampling mode can be chosen to set the internal sampling of the sensor for one measurement. Changing the mode to a higher resolution will increase the accuracy of a measurement but also has an impact on energy consumption, reaction time and the long term stability [17].

Table 4 gives an overview about the four oversampling setting modes. As the weather station is designed for long and continuous measurements, standard mode will be sufficient.

Figure 10 - BMP085 Temperature and Pressure Reading Process

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mode** | **Oversamping Setting** | **Internal Samples** | **Conversion Time (ms)** | **Noise (hPA)** |
| Ultra Low Power | 0 | 1 | 4.5 | 0.0.6 |
| Standard | 1 | 2 | 7.5 | 0.05 |
| High Resolution | 2 | 4 | 13.5 | 0.0.4 |
| Ultra High Resolution | 3 | 8 | 25.5 | 0.03 |

Table 4 - Measurement modes of the BMP085 and resolution effects

| **BMP085** | **Arduino Mega Pin** |
| --- | --- |
| SDA | 20 |
| SCL | 21 |
| XCLR | Not Connected |
| EOC | Not Connected |
| GND | GND |
| VCC | 3.3V |

As mentioned before, the BMP085 sensor is using the I2C protocol to communicate with the MCU and has to be connected to SDA and SCL pins of the Arduino Mega as shown below.

Table 5 - BMP085 Pin Connections to Arduino Mega

The altitude of the instrument has to be taken into account as well, because air pressure decreases with increasing height. For getting comparable values it has to be calculated to sea level pressure [6].

Working with the BMP085 should be done carefully, because it could be damaged by shocks or when getting in contact with water. It should be handled as Electrostatic Sensitive Device (ESD) [18] too. Therefore, it should be installed in a dry environment with constant temperature.

#### Precipitation

Rainfall is measured by a Tipping Bucket Rain Gauge included in the Weather Meters Kit [19] available from Sparkfun. Each time the self – emptying tipping bucket is emptied, a reed switch closes once. This corresponds to 0.2794 mm of rain fall [20]. The state change is caught by the MCU using Arduino’s Interrupt input.

A problem that frequently appears at trying to capture the state change of a switch is called bouncing problem. State change means that the input at a digital pin on the Arduino changes from high to low (equals from 5V to 0V) or vice versa. The problem is that each switching contains interferences in the signal before it reaches the level of 5V or 0V. This can lead to multiple counts from only one state switch. A visualization of the problem is shown in Figure 5.

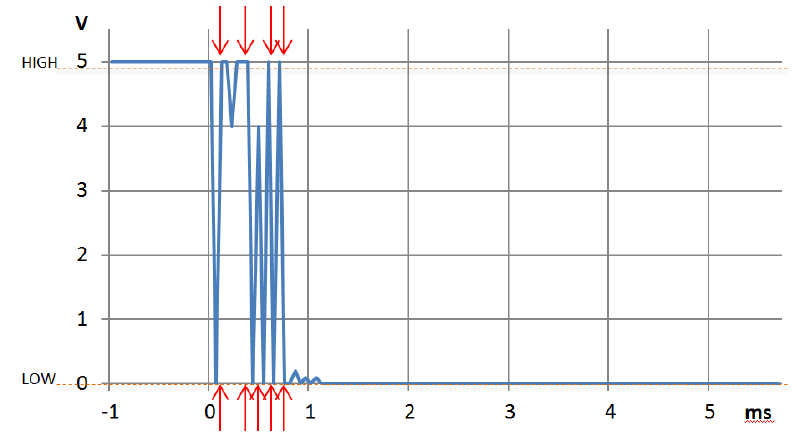


Figure 11 - Bouncing problem at a digital input. MCU will read nine state changes instead of one

To overcome this problem there are two solutions. The first is to integrate a capacitor to the circuit and the second is to define a software method with a debounce interval of some microseconds (μs), which is called very time a state change is recognized.

For the project the second method is used. The implementation is show below.

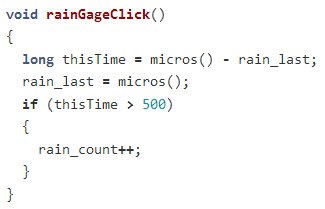


Figure 12 – Rain Reading And Switch Debouncing Implementation

The code above will ignore any interrupt that occurs within 500μs (0.0005s) of the previous one. This technique will limit the max rain this configuration can measure to a very high value that do not corresponds to an earthly value.

##### Wiring the Rain Gauge

The two wires coming from the rain gauge, through the RJ11 cable, have to be connected to the Arduino Mega as shown in the figure below.



Figure 13 - Connection of the Rain Gauge to the Arduino Mega

The red line must be connected to Ground while the green line is pulled up with a 10kΩ resistor to 5V. The output of the sensor has to be connected to digital pin 19 on the Arduino Mega.

A disadvantage of this method is that only rain fall can be measured. To include snow fall in the precipitation measurement a self heated device would be needed. Temperatures below 0° Celsius could also lock up the anemometer. This should be taken into account when analyzing the data of the rain gauge.

#### Wind Speed //TODO INCLUDE getGust NEW FUNCTION

The wind speed is measured with a cup – type anemometer included in Sparkfun’s Weather Meters Kit [19]. In principle, it works exactly like the rain gauge. At every turn, it closes a contact as a magnet moves past a switch, which is connected to a digital counter. 0.67 meter per second or 1.29651 knots causes the switch to close once per second [20].

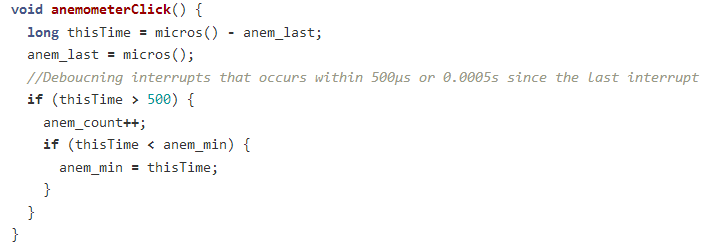
Switch debouncing is implemented for the anemometer exactly the same way as the rain gauge. The max wind speed this configuration can measure is limited to about 2,700 knots that does not affect the system since that kind of values are not earthly.

Figure 14 - Anemometer Reading And Switch Debouncing Implementation

##### Wiring the Anemometer

The anemometer is connected with a two wired cable to the wind vane. The two center wires (yellow and red) of the cable coming from the wind vane are used by the anemometer. Again, the red cable needs a pull-up resistor before connecting it to 5V. The yellow cable is connected to GND.



Figure 15 - Connection of the Anemometer to the Arduino Mega

#### Wind Direction

The wind vane of the Weather Sensor Assembly [19] contains eight switches, each of them connected to a resistor with different resistance values. A magnet switch can close two switches at once which allows to indicate 16 different positions. The voltage of the output cable can be measured at an analog input on the Arduino Mega. Directions for corresponding voltage output can be found

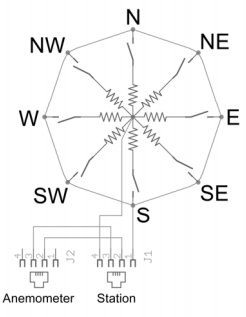
In table 6.

Figure 16 - Wind Vane resistor arrangement

|  |  |  |
| --- | --- | --- |
| **Direction (Compass)** | **Direction (Degrees)** | **Voltage Output** |
| N | 0 / 360 | 3.84 |
| N - NE | 22.5 | 1.98 |
| NE | 45 | 2.25 |
| E - NE | 67.5 | 0.41 |
| E | 90 | 0.45 |
| E – SE | 112.5 | 0.32 |
| SE | 135 | 0.90 |
| S – SE | 157.5 | 0.62 |
| S | 180 | 1.40 |
| S – SW | 202.5 | 1.19 |
| SW | 225 | 3.08 |
| W – SW | 247.5 | 2.93 |
| W | 270 | 4.62 |
| W – NW | 292.5 | 4.04 |
| NE | 315 | 4.78 (4.33) |
| N – NW | 337.5 | 3.43 |

Table 6 - Analog Voltage output with corresponding direction

Testing of the wind vane showed that voltage values at 315o is not 4.78V as mentioned at the datasheet. A 10kΩ resistor in series with a 64.9kΩ resistor, the voltage drop across the vane will be 4.33V.

For better power management, the wind vane is powered by the digital pin VANE\_PWR right before it is used. A 100ms delay is inserted after the power pin is switched to HIGH just to let things settle down.

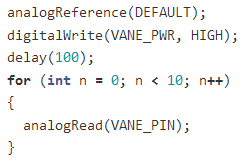


Figure 17 - Wind Vane Powering Up

##### Wiring the Wind Vane

The green and black outer wires of the cable from the wind vane have to be connected to the Arduino Mega as shown in Figure 8.

The wind vane and the anemometer shares the same RJ11 cable.



Figure 18 - Connection of the Wind Vane to the Arduino Mega

As mentioned before, the incoming voltage can be measured with an analog pin on the Arduino Mega. The values, which are read out at the analog input, have to be converted from 10 bit (0-1023) to a scale between 0 and 5V (0 to 5000). The resulting voltage can then be used to grab the corresponding value from a list according to Table 6.

### Prototype Architecture

//TODO: Physical setup description (mounts, containers etc.)

A schematic overview, created on Fritzing [21], of the wiring is show in figure 17.

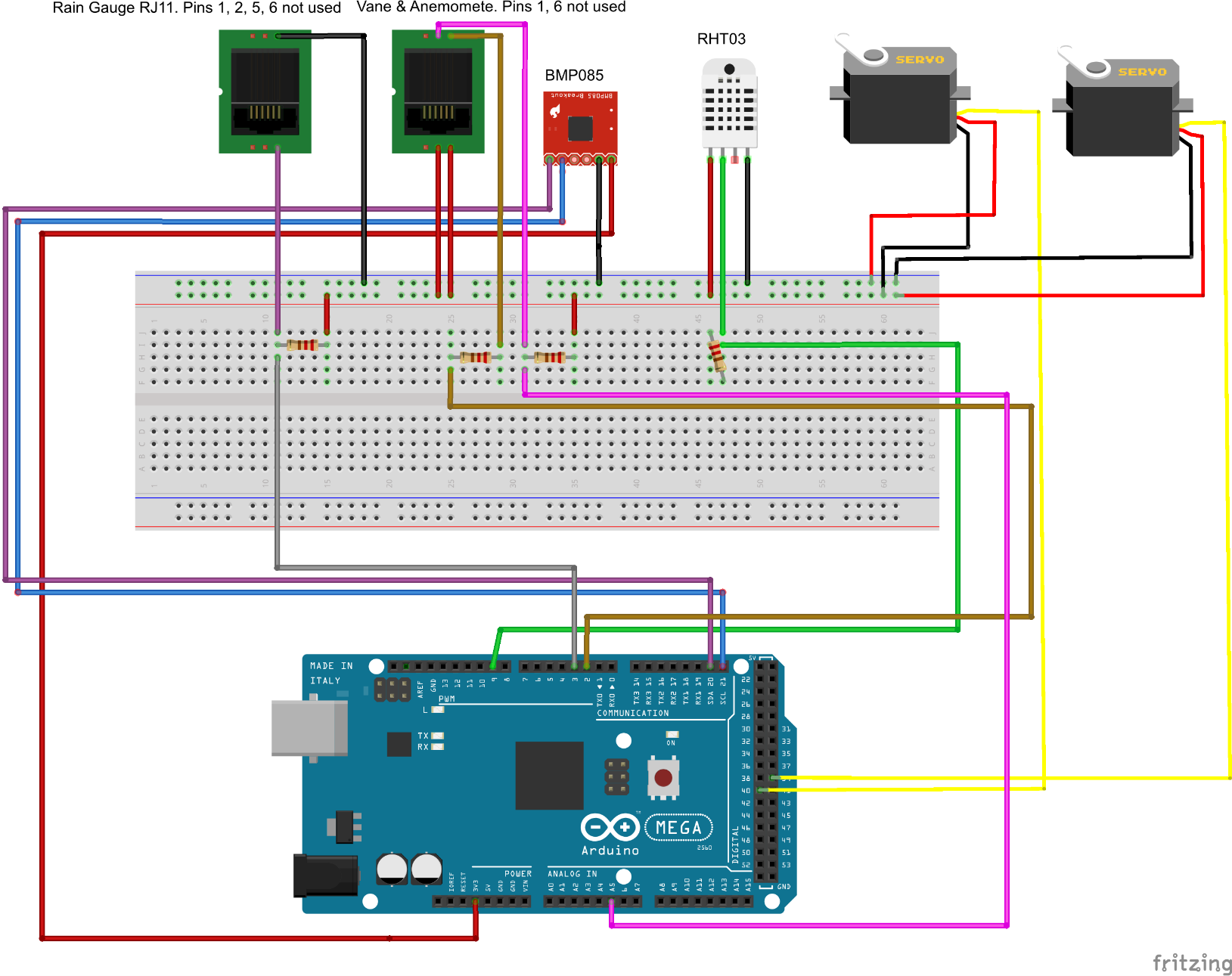


Figure 19 - Schematic wiring of all components

## In Depth Project Analysis

Following the Hardware components review, the Software analysis will dive deep into the main parts of the code that was used for the project to work.

This section will be split into four parts:

* Arduino Sketch
* Server – Side PHP scripts
* Android Application
* Time Lapse Video
* Web Site

### [Arduino Sketch](https://github.com/mKontakis/Sensebox/blob/master/WeatherMeters/ver11/ver11.ino) [23] //TODO WATCHDOG

//TODO: include watchdog timer

The Arduino Sketch is the most basic part of the project. It is the beginning. If this does not work perfectly nothing else can be done.

The Sketch consists of three major parts.

* The sensors values readings
* The HTTP Requests
* The Pan – Tilt camera mount control

#### Sensors Values Readings

Each sensor has its own style for reading its value.

##### RHT03

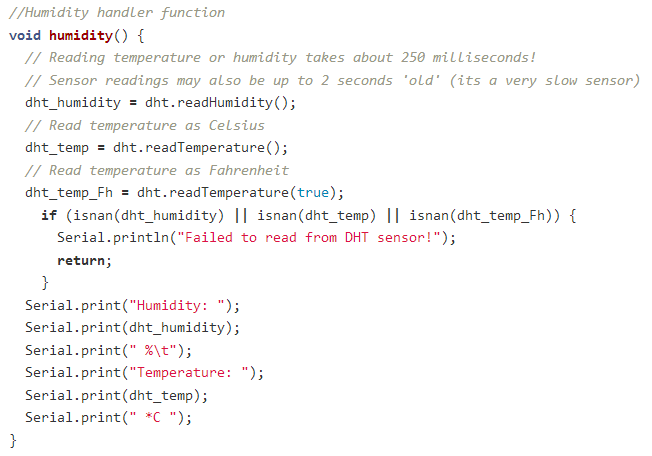
To integrate RHT03’s functionality to the project the library written by Adafruit Industries [22] is used.

First of all, the library must be imported and a DHT object is created.

#include <DHT.h>

DHT **dht**(DHTPIN, DHTTYPE);

To get RHT03’s readings the library does the heavy job.



The code above is very simple, since DHT library undertakes to deal with all the heavy job. At the beginning, temperature and humidity values are stored at the global variables ***dht\_humidity, dht\_temp***. Then an if else statement checks if the values read are numbers by calling **isnan** function (Is Not A Number). This check is being done to ensure that everything works well in the sensor.

##### BMP035

Collecting BMP035’s values was a challenging task, since the algorithm described in Figure 9 had to be implemented in the MCU.

The first thing during the development of the sketch, was to make sure that I can actually communicate with the sensor. For many digital sensors, once the communication is established it is halfway done.

First of all, the functions used to read 8 and 16 bit values from the BMP085 were implemented.

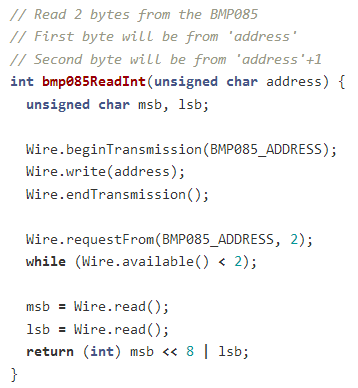
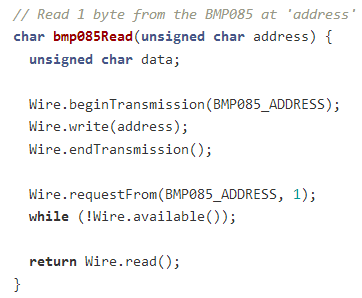


Figure 20 - Function to read 1 byte from the bmp085

Figure 21 - Function to read 2 bytes from the bmp085

According to sensor’s datasheet, the 176 bit E2PROM memory is partitioned in eleven words of 16 bit each. These contain eleven calibration coefficients. Before the first calculation of temperature and pressure, the master reads out the E2PROM data (The function at figure 20 is responsible for this task. It is being called only once during the setup process). The data communication can be checked by checking that none of these words has the value of 0 or 0xFFFF. The parameter association with the Registers Addresses can be seen in the Table 7.

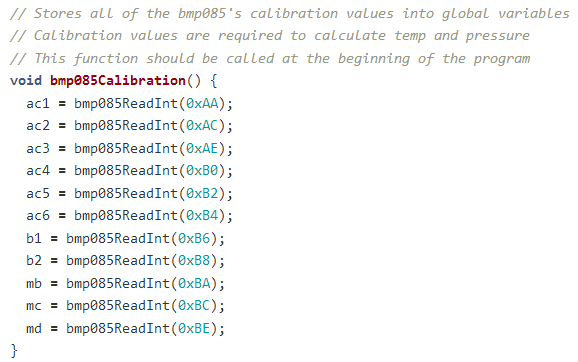
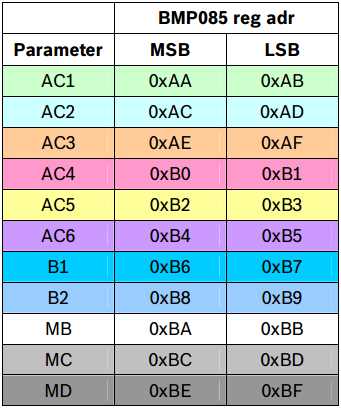


Figure - BMP085 Calibration

Table - BMP085 Calibration Coefficients

Once we have read the calibration values, we just need two more variables in order to calculator temperature and pressure. ‘UT’ and ‘UP’ are the uncompensated temperature and pressure values. Every time the temperature or pressure is measured, first ‘UT’ and ‘UP’ values must be read. They are the starting point for calculating the actual temperature and pressure values. The uncompensated temperature is an unsigned 16-bit integer number while uncompensated pressure is an unsigned 32-bit long number. The two functions can be seen at figures 21, 22.

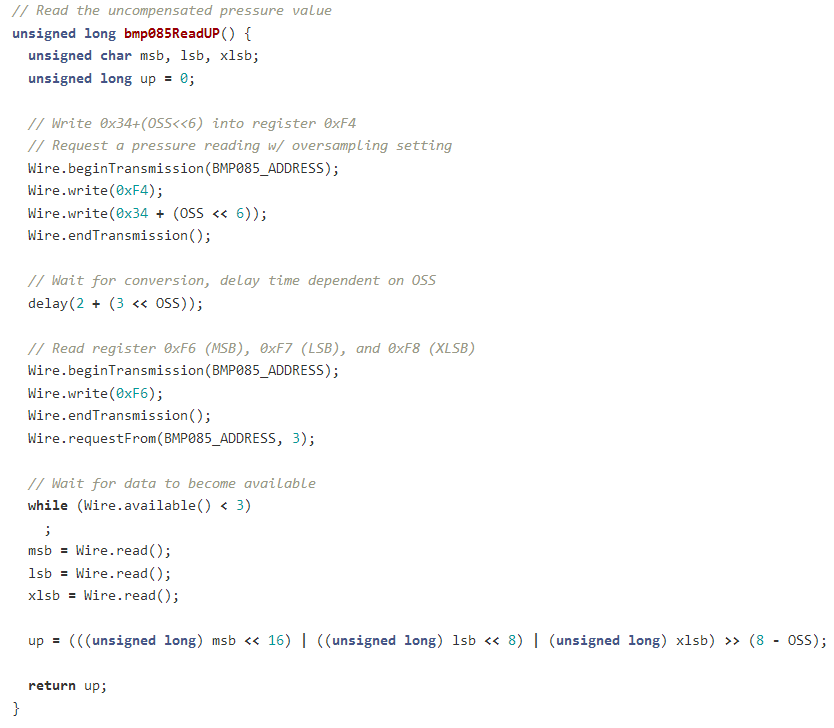
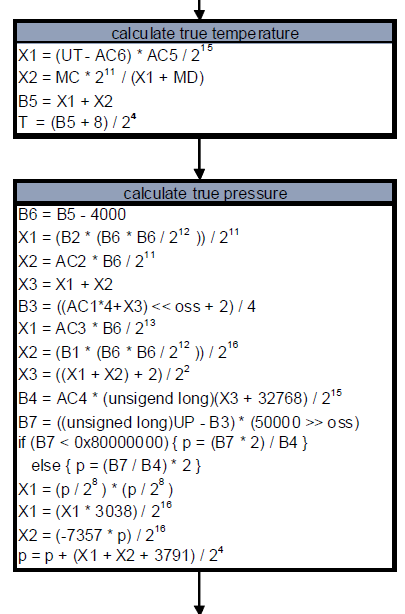


Figure 23 - BMP085 Read Uncompensated Pressure Function



Figure 24 - BMP085 Read Uncompensated Temperature Function

For last part of the calculation process, two functions are needed to take care of the following:



The temperature calculation is rather a simple task, but the pressure ones are more complicated.

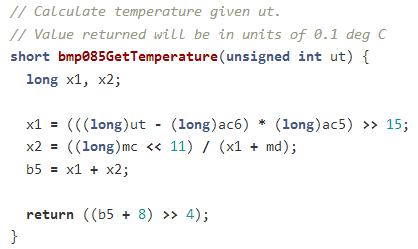


Figure 25 - BMP085 Final Temperature Calculations

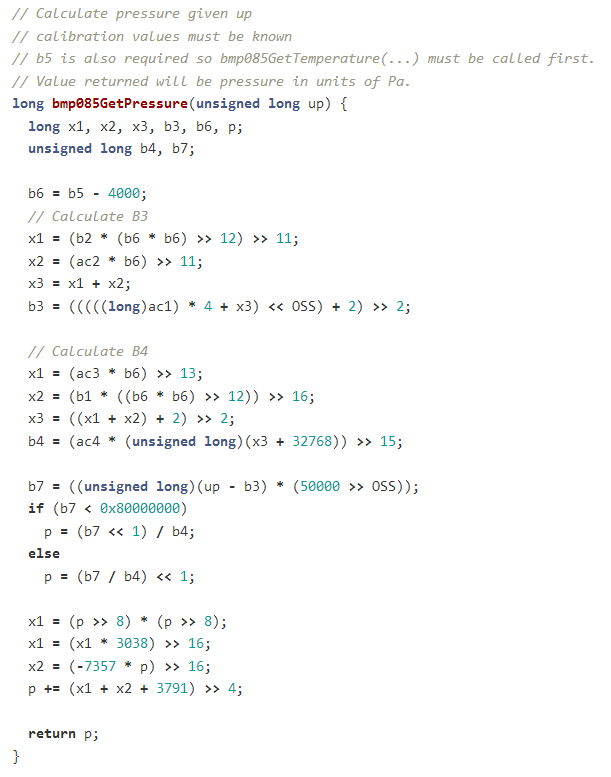


Figure 26 - BMP085 Final Pressure Calculations

##### Anemometer and Rain Gauge

Gathering the data from the anemometer and the rain gauge was an almost similar procedure, this is why they are analyzed together. The concept behind is the Interrupts mentioned at 4.1.1.1, 4.1.2.3, 4.1.2.4.

Firstly, pins and interrupts must be configured. Interrupts are set to FALLING mode, for when the pin goes from HIGH to LOW (Figure 25).

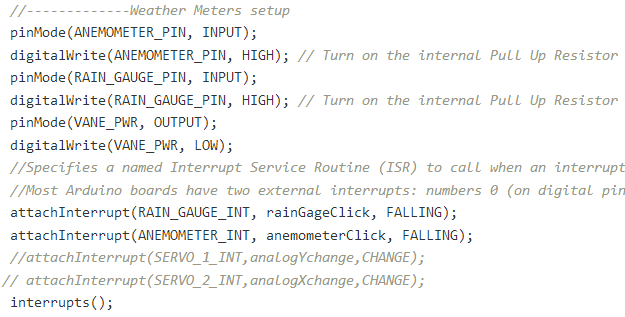


Figure 27 - Weather Meters Configuration

The next step is to implement rainGageClick (Figure 10) and anemometerClick (Figure 12) functions that are called whether an interrupt occurs. These functions are very simple. They just count how many times an interrupt occurred.

Finally, the counters values are used to calculate the final wind and rain depending on each sensor’s factor. (i.e. For Rain each interrupt occurred corresponds to 0.2794mm of rain, instead for wind it is 2.4 miles per hour).

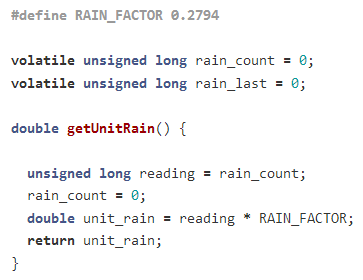


Figure - Rain Final Calculation

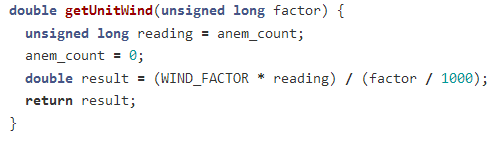
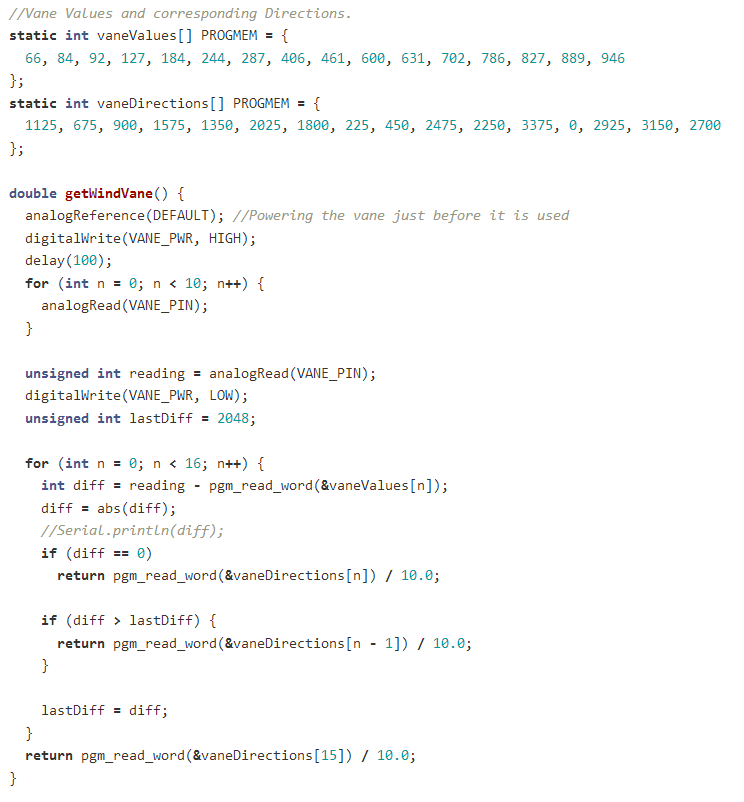


Figure 29 - Wind Final Calculation

##### Wind Direction

Wind Direction’s function compares the ADC value to the ideal values in vaneValues array and returns the corresponding wind direction in degrees.



To save SRAM memory, PROGMEM [25] is used to store data in flash instead of SRAM. For the purpose of the project it is not really useful since memory is enough, but it is a programming practice worth mentioning, especially when referring to the domain of MCUs that memory is valuable.

The pgm\_read\_word function [24] pulls the values out of FLASH and into SRAM when needed but only works with integers, so the actual wind directions are stored as 10x their value.

#### HTTP Request

The HTTP Request sends the sensors data to the MYSQL database. The request method used is GET [26]. GET can be used to send small amounts of data to the server through URL Encoding.

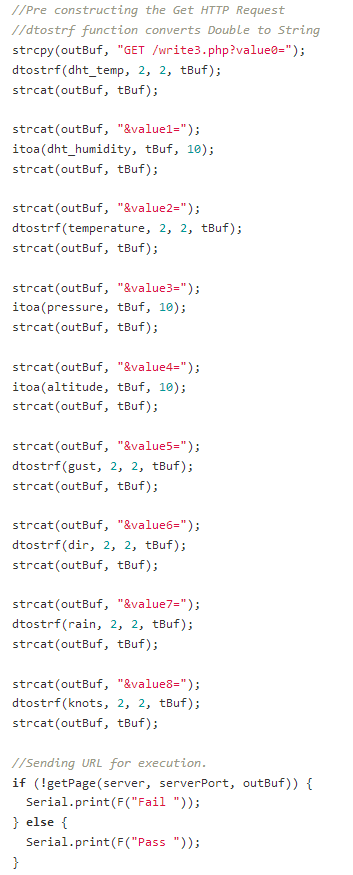
The construction of the URL (figure 28) is being done before the HTTP request process, so Arduino can send the data in one packet using the write function of Ethernet library.

Figure - URL contruction

When the URL assembly process is done, it is passed to getPage( ) (figure 29) function.

This function connects to the server and executes the HTTP Request with the data included within the URI.

The getPage function also has a connection status control. If the connection fails or the server stalls, the Arduino code shows a “Timeout” message after ten seconds and continue to attempt the connection every a time threshold we set.

Dtostrf function explained

Dtostrf (double val, signed char width, unsigned char prec, char\* s)

converts the double values passes into an ASCII representation that will be stored under s. The function returns the pointer to the converter string s.



Figure - HTTP Request Execution

#### Pan Tilt Camera Mount Control

The Remote Camera Control part of the Arduino sketch, controls the Pan Tilt [27] Bracket remotely. Arduino and Ethernet shield serves up a web page that allows the control of the servos that rotate the bracket. When a client connects to the Arduino server, it sends back to the client’s browser the HTML page (figure 30, 31).

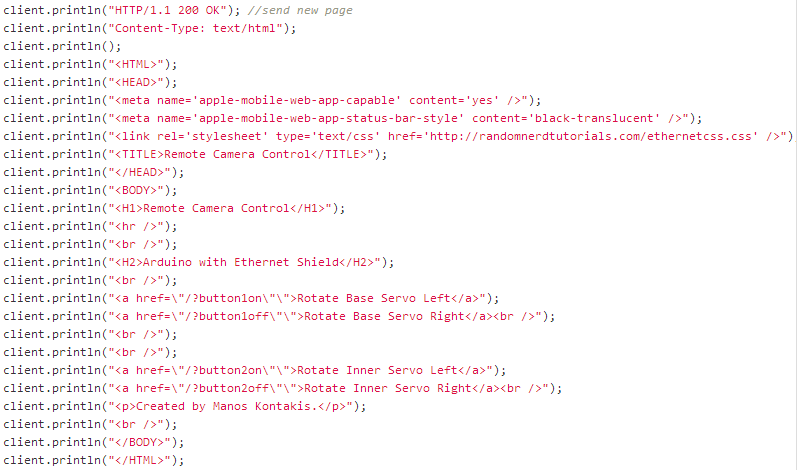


Figure - HTTP Response

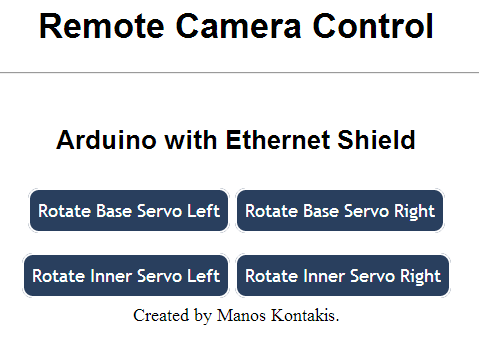


Figure 33 - Remote Camera Control

When the user clicks on a button, the browser generates a HTTP Request with the button name attached to the URL. Arduino reads the HTTP Request and checks which button triggered the request.

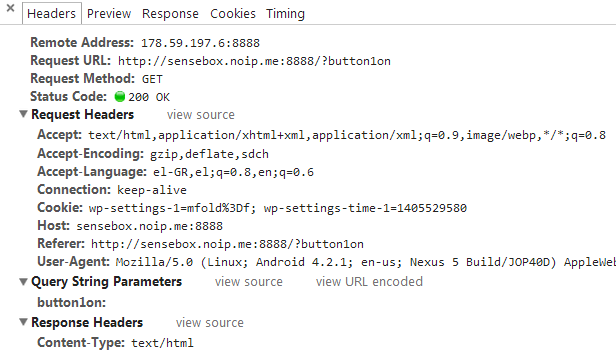
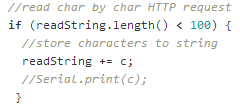


Figure 34 - Generated HTTP Request on button click

indexOf locates a character or String within another String. Returns the index of the desired String within the String or -1 if not found.

The rest of the code rotates the servo.

Figure

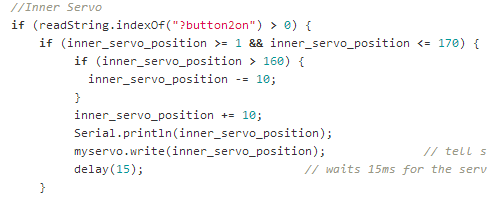


Figure - Servo Control Example

#### Watchdog Timer

A watchdog timer (WDT; sometimes called a computer operating properly or COP timer, or simply a watchdog) is an electronic timer that is used to detect and recover from computer malfunctions. During normal operation, the computer regularly restarts the watchdog timer to prevent it from elapsing, or "timing out". If, due to a hardware fault or program error, the computer fails to restart the watchdog, the timer will elapse and generate a timeout signal. The timeout signal is used to initiate corrective action or actions. The corrective actions typically include placing the computer system in a safe state and restoring normal system operation.

The ATmega328P that powers the Arduino Mega provides a programmable Watchdog Timer with a separator on-chip 128kHz Oscillator. The WDT gives an interrupt and / or a system reset when the counter reaches a given time-out value. In normal operation mode, it is required that the system uses the WDR - Watchdog Timer Reset - instruction to restart the counter before the time-out value is reached. If the system

Does not restart the counter, an interrupt or system reset will be issued.

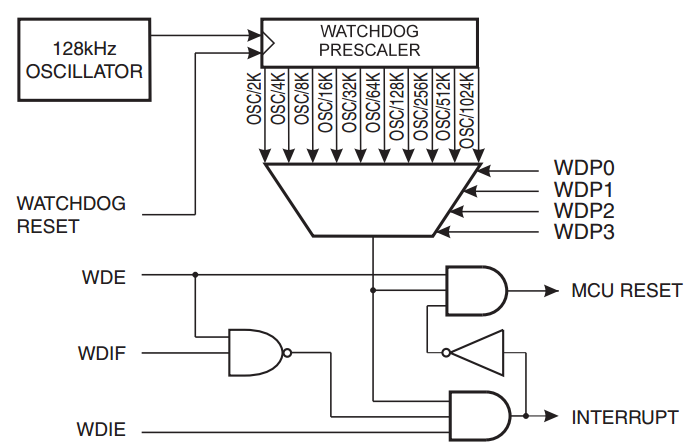
**In Interrupt mode** (WDIE), the WDT triggers an interrupt when the timer expires. This interrupt can be used to wake the device from sleep-modes, and also as a general system timer. **In System Reset mode**, the WDT gives a reset when the timer expires. This is typically used to prevent system hang-up in case of runaway code. **The third mode**, Interrupt and System Reset mode, combines the other two modes by

Figure - Watchdog Timer Schematic

first giving an interrupt and then switch to System Reset mode. This mode will for instance allow a safe shutdown by saving critical parameters before a system reset. This is the mode used in the project.

### Android Application [28]

The Android Application processes the weather data and presents them in different formats:

* Charts
* Current Weather Conditions Report

Figure - Android App Icon

* Min / Max Report
* Data presentation in Table Layout



The Charts are generated using the GraphView [28] library. GraphView is a library for Android that programmatically creates flexible and elegant diagrams. The reason why I picked this library is because it is easy to understand, to integrate and to customize. The library supports two different types of diagrams. Line and Bar Charts.

As soon as the app is launched, the Splash Screen appears. During the Splash Screen, the weather data is fetched, in JSON format, and parsed to generate the 24 Hours Charts (First Screen). If the device is connected to the internet with a decent connection, the Splash Screen appears for around 5 seconds. The slowest the connection is the longest the Splash Screen lasts, because the device has to connect to the server and fetch the Weather Data. This is the most time – consuming part. Parsing the fetched data and drawing the Charts takes only some milliseconds.

Figure - Splash Screen

Figure 37 - Start Screen shows the start screen of the application. Seven charts (only two are shown in the figure) are generated. Swiping from left to right or clicking the Action’s Bar text will display the menu, implemented using Navigation Drawer [29] element //TODO NAVDRAWER FIGURE REFERENCE .

Figure - Start Screen

#### Charts Creation Procedure

To generate the Charts the following steps are needed:

* Construct the URLS
* Execute HTTP Requests and fetch Weather Data from Database in JSON format
* Parse JSON Objects
* Draw the Charts

##### URLs Construction

The makeURL method constructs the URLs using the Builder Class (BuildString) and saves them in a String Array. One URL corresponds to one chart.

A URL built with that function can be found in Figure 41 - URL example

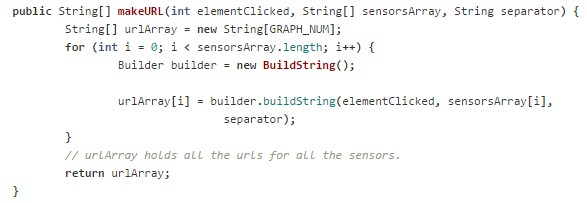


Figure 41 - Method to construct URLs

As soon as this procedure is done, the application is ready to proceed to the next step and execute the HTTP requests.

##### Execute HTTP Requests

The URLs one by one are executed and the content of the response is handled in Figure 40. There the BufferedReader reads the response and appends the characters of the response one by one in a StringBuilder object and finally when it is done it converts it to a **JSONobject**.

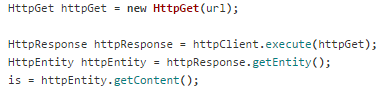
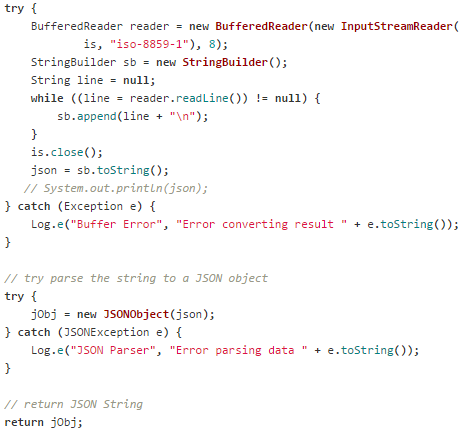
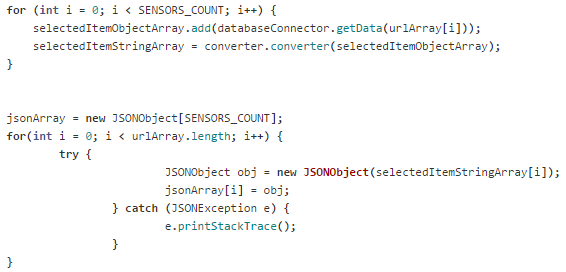


Figure - Android HTTP Request



**Figure 43 - HTTP Response Handling**

At Figure41, a [JSONObject](http://developer.android.com/reference/org/json/JSONObject.html) array is being filled with JSON data. Each index holds data for each sensor.



So at this point of time, that all the data from all the sensors are stored into the jsonArray (JSONObject[]) we can proceed on parsing all these data and finally generate the charts.

**Figure 44 - Filling JSONobject Array, one index per sensor**

##### Parse / Resolve JSON Objects

**JSON** or **JavaScript Object Notation**, is an [open standard](http://en.wikipedia.org/wiki/Open_standard) format that uses [human-readable](http://en.wikipedia.org/wiki/Human-readable) text to transmit data objects consisting of [attribute–value pairs](http://en.wikipedia.org/wiki/Attribute%E2%80%93value_pair). It is used primarily to transmit data between a server and web application, as an alternative to [XML](http://en.wikipedia.org/wiki/XML).

Parsing the JSON Objects and extracting the weather information from them is a little complex procedure. In the cases of monthly and quarter charts/data presentation these functions are intense for the device.

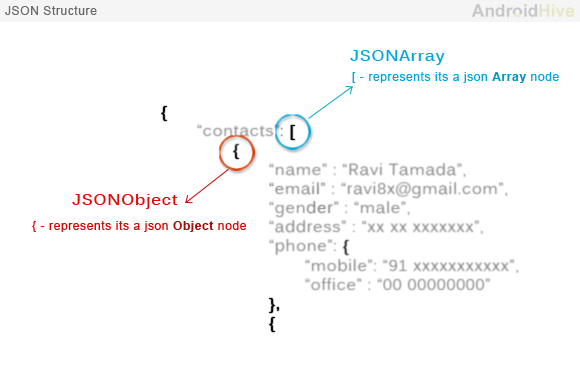
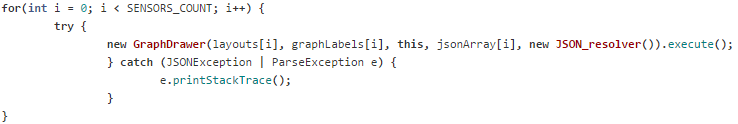
In general all the JSON nodes will start with a square bracket or with a curly bracket. The difference between [ and { is, the square bracket ([) represents starting of an **JSONArray** node whereas curly bracket ({) represents **JSONObject**. So while accessing these nodes we need to call appropriate method to access the data.

Figure - Example of JSON Elements

At the for loop below, a new instance for each sensor of the class that builds and draws the charts is created. One of the parameters that this method gets as input is one index, per loop, of the jsonArray and also a new JSON\_resolver object in order to fulfil the parsing of the object. More details on GraphDrawer class can be found in the next section.



The resolve method in Figure 43 extracts the data from the JSONObject passed as parameter in the constructor of GraphDrawer class. The resolve procedure is being done according to the Figure 42.

The method stores the Dates and the Data in two separate Array Lists data structures so they can be easily used to generate the charts.

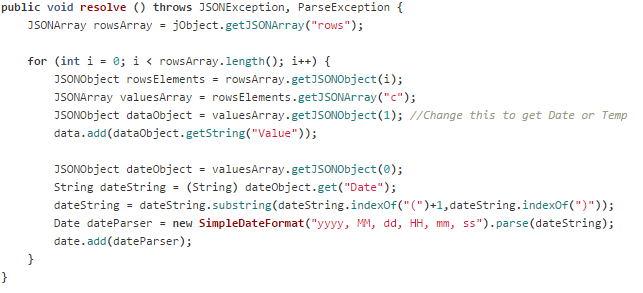


Figure 46 - JSON resolver method

##### Drawing the Charts

As soon as the Data and the Dates is extracted from each JSONObject the charts are very easy to generate. The procedure is a straightforward one thanks to the intuitive library Android Graph – View [29].

[GraphDrawer class](https://github.com/mKontakis/Sensebox/blob/master/SenseBox.Project/src/com/graduational/sensebox/GraphingClasses/GraphDrawer.java) is responsible to create and draw the charts to UI. It extends AsyncTask so each chart creation process is executed in a separate thread. AsyncTask fits just perfect in our situation since it performs HTTP Requests and JSON parsing to background and publish the results on the UI thread as soon as each chart is ready.

First of all, a GraphViewData Array is created, to keep all the samples for the charts. Its constructor takes two parameters, the X value (Date) and the Y value (Sensor Value) as shown in line 95. To add dates to X axis they must be converted in [Unix Time](http://en.wikipedia.org/wiki/Unix_time) format. This is what getTime() method does in line 81. This method returns how many milliseconds have passed since January 1, 1970, 00:00:00 GMT.

Then, a GraphViewSeries class is instantiated that holds the data, description and styles. The code at line 103 adds the series of data to the chart.



Figure 47 - Method to handle graph drawing

#### Current Weather Conditions Report

Current weather conditions report is a very simple unique top – level view that displays the last reading from the sensors. As shown in Figure 45, the time and date of the last reading is displayed on the top of the screen. Arduino reads the sensors every six minutes. The last reading of this screenshot was at 16:25 and current time was 16:29 (top right corner) so this is confirmation that when the screenshot was taken everything worked smoothly.



For the creation of this view a very similar process with the creation of the charts is followed. URL construction (with different flag as show in the example of Figure 47 - Dynamic SQL Query Construction), execution of HTTP Request, JSON resolving and finally instead of generating charts it displays the data in the TextViews.

Figure - Current Weather Conditions Report

#### Min / Max Report

This is another unique top – level view for displaying the minimum and the maximum reading of each sensor in daily, weekly, monthly and quarter basis.

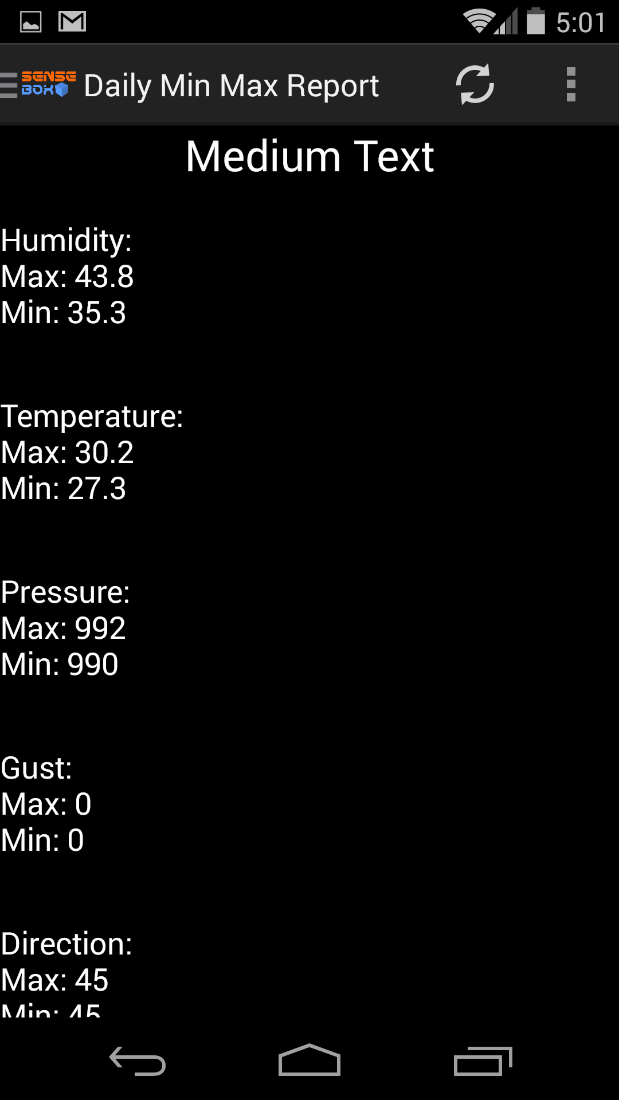
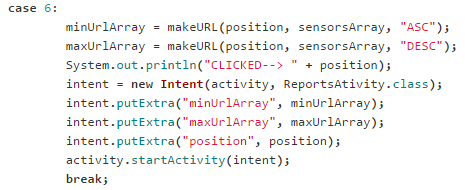
Like Current Weather Conditions Report, the creation process is similar with the creating of the charts. The difference, over the flag, that worth mentioning is that the URL construction and HTTP Requests execution processes happen twice. One for the Min and one for the Max. What separates these two is the **separator** variable that is attached to the URLs (Figure 49 - URL example). When the separator takes ASC value it refers to Min values and consequently DESC refers to Max values.

Figure - Min / Max Report View

Speaking in SQL what exactly the separator does is to order the returned table by the sensor value in ascending or descending order and take the first result each time. This technique returns the Max/Min value.

#### Data Presentation in Table Layout

Table Layout is a complex and tricky layout that arranges its children into rows and columns. It consists of a number of [TableRow](http://developer.android.com/reference/android/widget/TableRow.html) objects, each defining a row.

Again, to generate the tables the first three steps are familiar. URLs construction with corresponding flag, HTTP Requests execution and JSON resolving. The next step is to add the Array Lists with the values of each sensor to an Array List of Array Lists (Figure 47 - ArrayList that holds ArrayLists).



Figure 50 - ArrayList that holds ArrayLists

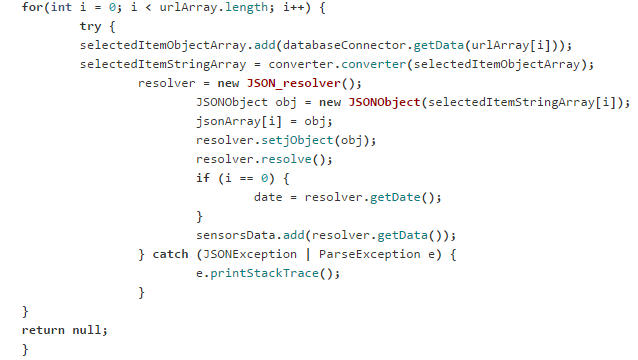


Figure 51 - Table Layout Creation

The last step of this procedure is to create the Table Rows object and dynamically add them to the Table Layout.

Figure 49 pictures the Table Layout. At the top – right corner options the range of the time results can be selected. Default is daily (24 Hours). 2 Days, Weekly and Monthly can be selected. Also, there is a Refresh Button D:\Dropbox\GraduationalProject\SenseBox\SenseBox.Project\res\drawable-hdpi\ic_menu_refresh.png to include the latest readings to the table.

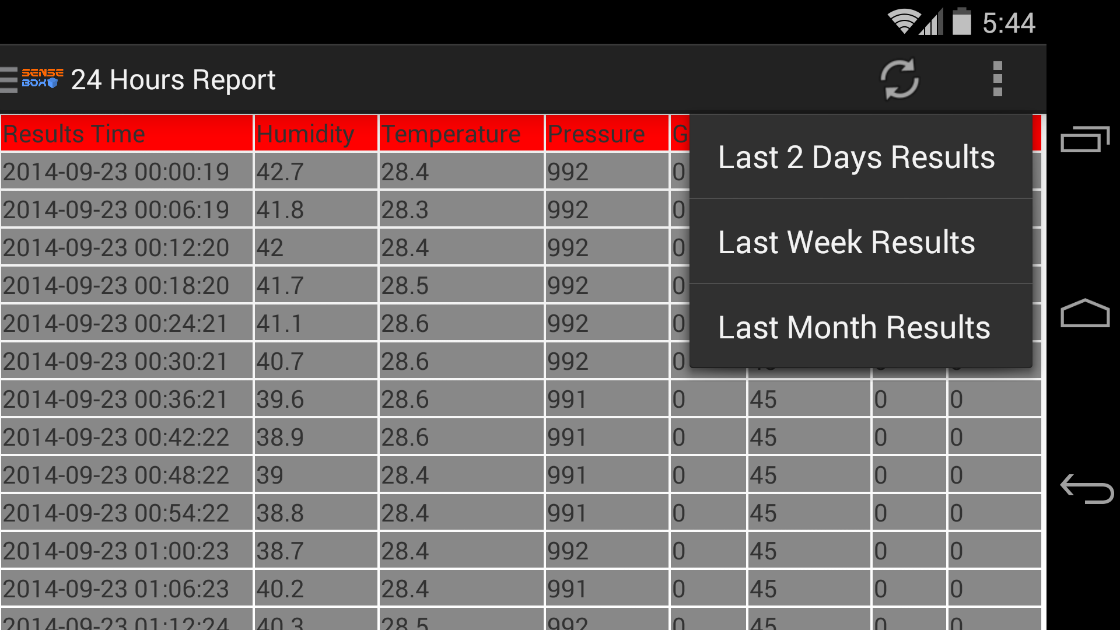


Figure 52 - Table Layout View

As shown in Figure 50 - Adding Data to Table Layout first a TableRow is instantiated. Then a TextView is instantiated and some parameters are passed to it. This TextView (in the outer for loop) is for the dates. In the inner for loop seven TextViews are created per one loop of the outer loop. These TextViews correspond to the seven sensors.

Generally, the number of the Text Views that are added to a Table Row corresponds to how many cells each row has.



Figure 53 - Adding Data to Table Layout

Finally, each TableRow that now includes the Date and the value of each sensor – eight in total - at that point of time, is added to the Table Layout with some Layout Parameters.

### Time Lapse Video Making

The procedure of making the daily Time Lapse video is mainly consists of Bash scripts and Cron [34] Jobs.

### Web Site

### Server – Side PHP Scripts

PHP is a server – side scripting language designed mainly for web development but also as a general – purpose programming language. A PHP interpreter is usually responsible to process PHP, which is usually implemented as a web server’s native module. After the PHP code is interpreted and executed, the web server sends resulting output to its client. This is exactly how PHP is used in the project.

#### Storing Data in **Database**

In *Figure 28 - URL contruction* the URL is constructed and data are URL encoded. The script in Figure 37, extracts the data from the URL, establishes a connection to the MYSQL database and stores data. For debugging purposes, it returns to the client the data values stored.

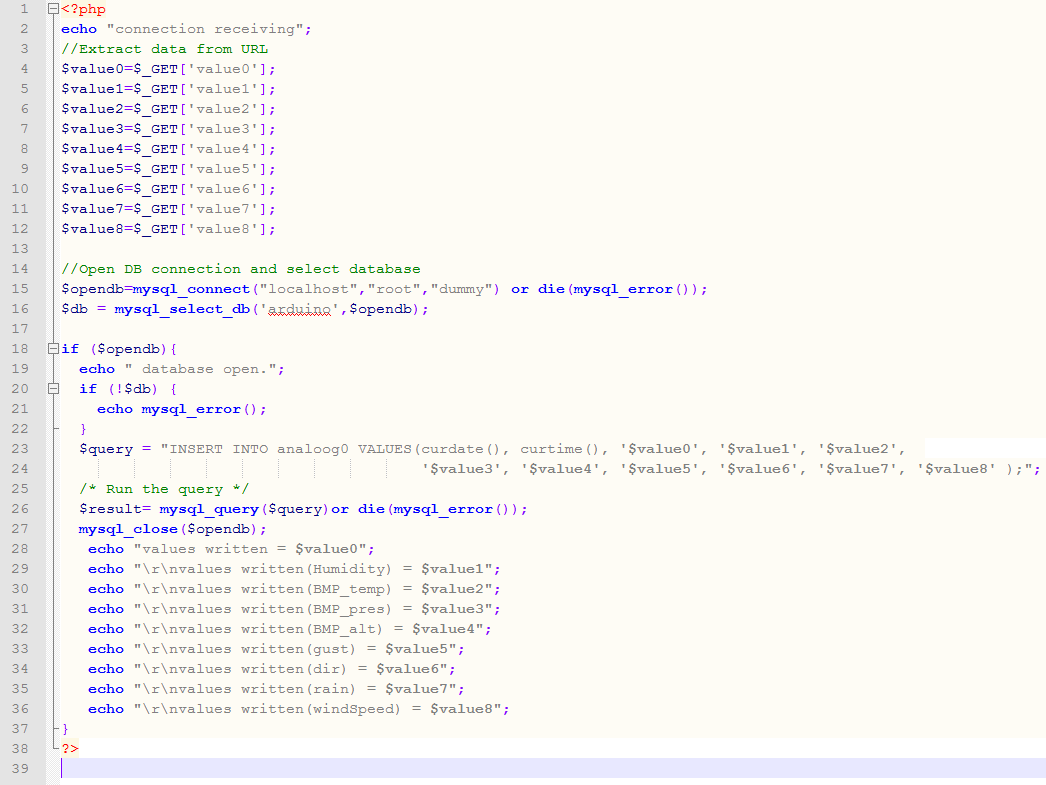


Figure 54 - Script to store data to database

//TODO: Fix google chart’s fetching data PHP script (generalize – optimize) and make report

#### Android Application Fetching Script

This PHP script constructs the corresponding SQL Query depending on three variables attached on the URL, the Sensor, the Flag and the Separator.

The Sensor variable declares the sensors whom data will be fetched. The Flag informs the script which selection was clicked from the Navigation Drawer menu while the Separator variable separates Min and Max values.

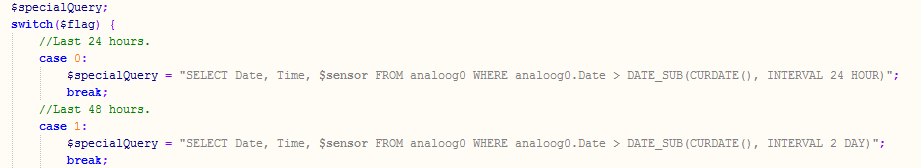


Figure 55 - Dynamic SQL Query Construction

*http://sensebox.noip.me/dynamicQueryExecutor.php?sensor=BMP\_temp&flag=6&separator=ASC*

Figure - URL example

After the SQL Query is constructed it is executed. The script returns the weather data in JSON format to Android Application. The construction of the JSON is shown in Figure 39

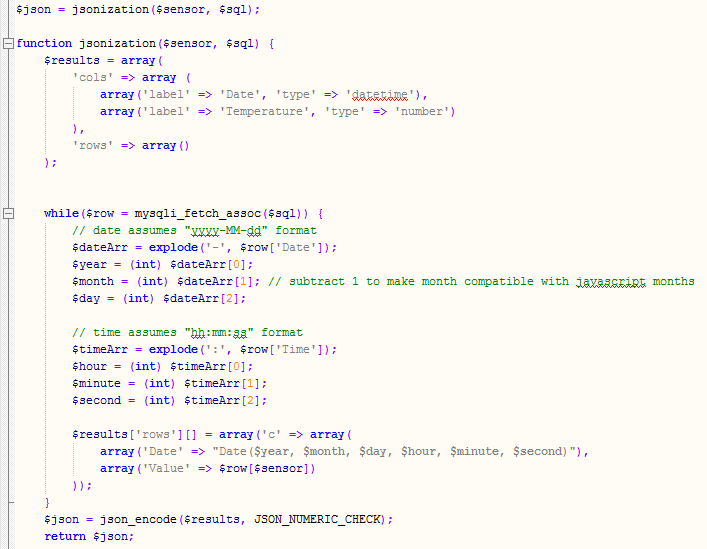


Figure 57 - SQL Table to JSON

Something that worth mentioning here are the cases of the Weekly, Monthly and Three Months Charts. In these three cases high resolution of the charts are of no importance. Also generating seven charts with so many samples (about 22000 for Three months, 7333 for one month per chart) makes the application too heavy for a mobile device. To fix this problem one row per two rows are selected from the table for the weekly graph, one per 8 for the monthly and one per 10 for the 3 months chart. The SQL Query for the Monthly chart is shown below.

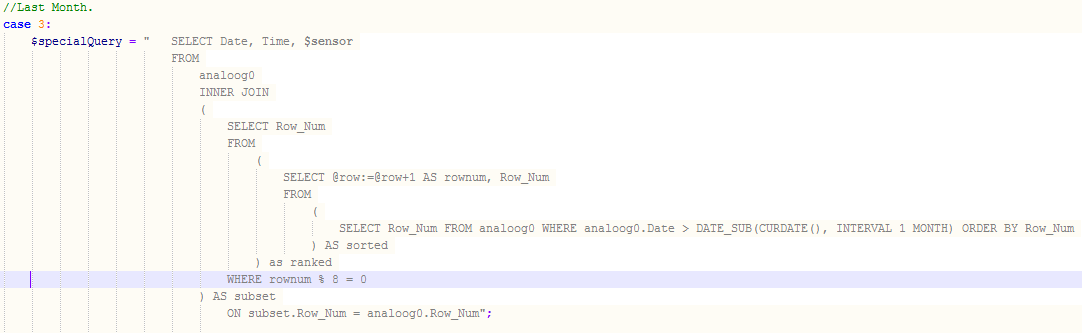


Figure 58 - SQL Query to select every n-th row

# References

1. Open Source Initiative: The Open Source Definition.

<http://opensource.org/docs/osd>

1. Open Source Hardware Association: OSHW Definition.

<http://www.oshwa.org/definition/>

1. Definition of Free Cultural Works: *Open Source Hardware.* [http://freedomdefined.org/OSHW](http://freedomdefined.org/OSHW%20)
2. Arduino: Homepage

<http://arduino.cc/en>

1. Arduino: *Reference.*

[http://arduino.cc/en/Reference/HomePage](http://arduino.cc/en/Reference/HomePage%20)

1. World Meteorological Organization, 1992: *International Meteorological Vocabulary*. Second edition, WMO-No. 182,Geneva.
2. Apache License, Version 2.0

<http://www.apache.org/licenses/LICENSE-2.0.html>

1. GNU General Public License, Version 2.0

<http://www.gnu.org/licenses/gpl-2.0.html>

1. System on a chip

<http://en.wikipedia.org/wiki/System_on_a_chip>

1. Google Charts API

<https://developers.google.com/chart/>

1. Arduino Mega 2560 R3 Web page.

<http://arduino.cc/en/Main/ArduinoBoardMega2560>

1. Arduino Ethernet Shield Web page.

<http://arduino.cc/en/Main/ArduinoEthernetShield>

1. Arduino Ethernet Library Web page.

<http://arduino.cc/en/reference/ethernet>

1. Arduino SD Library Web page.

<http://arduino.cc/en/Reference/SD>

1. RHT03 Official Datasheet.

<http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Weather/RHT03.pdf>

1. Bosch Sensortec: Pressure Sensors.

<http://goo.gl/SEpmgY>

1. BMP085 Datasheet.

<http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Pressure/BST-BMP085-DS000-06.pdf>

1. Electrostatic Sensitive Device

<http://en.wikipedia.org/wiki/Electrostatic-sensitive_device>

1. Sparkfun Electronics: Weather Meters Kit

<https://www.sparkfun.com/products/8942>

1. Sparkfun Electronics: Weahter Meters Datasheet

<https://www.sparkfun.com/datasheets/Sensors/Weather/Weather%20Sensor%20Assembly..pdf>

1. Fritzing

<http://fritzing.org/home/>

1. Adafruit Industries

<https://www.adafruit.com/>

1. Arduino Sketch Repository Link

<https://github.com/mKontakis/Sensebox/blob/master/WeatherMeters/ver11/ver11.ino>

1. <AVR/pgmspace.h>: Program Space Utilities:

<http://www.nongnu.org/avr-libc/user-manual/group__avr__pgmspace.html>

1. PROGMEM

<http://arduino.cc/en/Reference/PROGMEM>

1. HTTP Request: GET

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.3>

1. Pan – Tilt Bracket

<https://www.sparkfun.com/products/10335>

1. Android Application Repository

<https://github.com/mKontakis/Sensebox/blob/master/SenseBox.Project>

1. Android GraphView

<http://android-graphview.org/>

1. Android Navigation Drawer

<https://developer.android.com/design/patterns/navigation-drawer.html>+

1. Raspberry PI

<ww.raspberrypi.org>

1. Apache HTTP Server

<http://httpd.apache.org/>

1. WordPress

<https://wordpress.org/>

1. Cron

<http://en.wikipedia.org/wiki/Cron>

1. ATmega328P Data sheet

<http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf>

# Appendix

## A.1 Why do ISPs prefer dynamic IP

When ISPs were first starting, everyone connected to the Internet over a modem. And most people used the Internet for a few minutes to a few hours per week. Assigning a static IP to every subscriber would have been very expensive, for something that most people used just a few minutes a week.

As broadband connections have become more common, the practical reasons for not assigning a static IP have become much less noticeable, as now the majority of connections are "always-on"--even when nobody is (actively) using the Internet.

So there's a bit of a historical reason not to use static IPs--customers are already accustomed to using dynamic IPs.

When modern ISPs enforce dynamic IPs these days, it may be in part to distinguish between "consumer" and "professional" services--by reserving static IPs for customers who pay more, it gives customers who need that feature an incentive to upgrade their service level.

It can also serve as a deterrent for people abusing their consumer-grade service. Many ISPs, for instance, explicitly prohibit running "servers" on a home Internet connection. If every home user had a static IP, they'd be more inclined to abuse such terms of service.

It's also less of a management problem to assign customers dynamic IPs. If you move across town (but within the same ISP's service area), there's no need to re-assign how your static IP is routed; you'll just get a dynamic IP that exists in the new neighborhood.