Introduction

What follows is a description of an “Internet of Things” inspired, do-it-yourself, solar powered, weather station. The implementation described below, though certainly not unique, represents a balanced trade-off between cost and function. A high priority has been placed on using all "open-source" hardware and software. Avoiding proprietary, closed source software, as well as proprietary “cloud” services, allows for much greater flexibility. This flexibility facilitates ease of deployment and scale-out on a wide variety of open source platforms, and for a large variety of end user applications.

For example, the weather station might be in a location which only provides wireless connectivity to the Internet. In this case, the weather station would be configured to send data to a server physically located somewhere else. Alternatively, the weather station might be connected to the same local area network as the server hardware. In this case, the local network router may be configured to allow Internet access to the weather station web page hosted on the server. Many deployment options are possible.

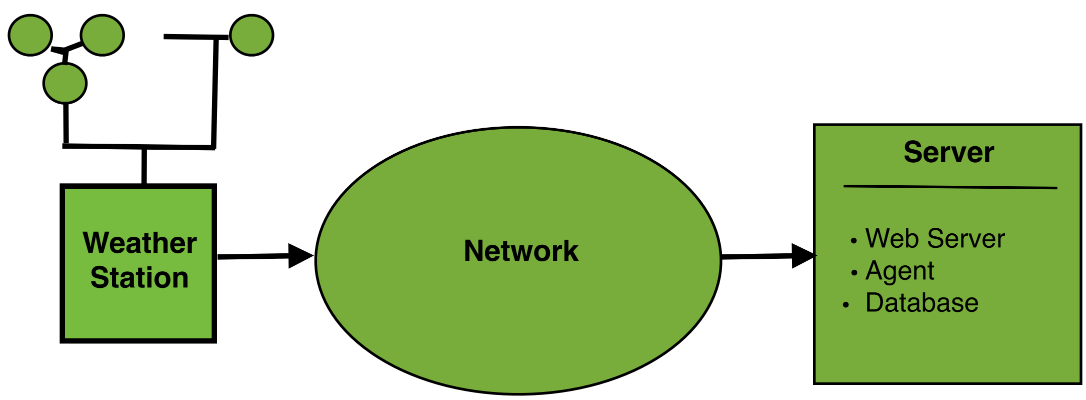

Figure 1. Overall conceptual view showing weather station, network, and server components.

This project plan uses the later model to provide an example of a “turn-key” solution that includes both the weather station, and the implementation of “back-end” services. Prerequisites for building and deploying the DIY Weather station include: soldering, ability to follow electrical hookup guides, knowledge of Arduino technology, Linux system administration, and some knowledge of the C and Python computer languages. The open source software provided in the Github repository should be sufficient, with only minor modifications, for most applications. The server side software has been designed around the file structure and system configuration found in most common Linux distributions. In most cases, the majority of modification will be in the HTML, customizing the look and appearance of the web page presentation to the end user.

DIY Weather System Overview

The DIY Weather system comprises three major components. Referring to figure 1, the weather station hardware, the network connecting the weather station to the server, and the server, itself. The following project description will focus on the weather station hardware and the server software. The network connecting the two together, including wireless access points, switches, routers, etc., will be considered background infrastructure beyond the scope of this discussion.

The weather station, itself, consists of a number of hardware components, as well as a software component running on the weather station micro-controller. The server hardware will be considered background infrastructure and not discussed, other than regarding software dependencies. The server hardware can be implemented in anything from a Raspberry Pi, running Raspbian, to a full up Internet cloud server running Linux. The server software consists of three components: a PHP script called remotely by the weather station, an agent for processing data received from the weather station, and HTML documents for displaying the data.

The weather station periodically sends data via HTTP through the network to the server. This weather data is embedded in the URL for the server. The URL calls a PHP script, which writes the data, embedded in the URL, to a file for use by the agent. The agent, a script implemented in Python, manages the data sent from the weather station. The agent converts and formats the data, updates a database, sends maintenance signals to the weather station, and makes the data available to the HTTP server. The agent provides the data to the HTTP server by writing to a Javascript object notation (JSON) formatted file, which Javascript, embedded in the HTML documents, can read and display in web pages.

Component Descriptions

Weather Station Hardware

Referring to figure 2, the weather station hardware consists of components for generating and managing power to the station electronics: a micro-controller, a weather shield, a wifi shield, and weather meters. A solar cell provides power, through a charge controller, to both the LiPo backup battery and the micro-controller.

Wind meters send out signals which the micro-controller interprets to determine wind speed and direction. The weather shield provides an electrical interface from the wind and rain fall sensors, to the micro-controller. The weather shield also contains the pressure and humidity sensors and an electrical interface for these two sensors. The pressure sensor measures both barometric pressure and temperature.

The micro-controller, the heart of the weather station, consists of a single, Arduino “Uno” type micro-controller. Arduino “shields” are various Arduino compatible devices that plug into the Arduino circuit board header connectors. The wifi shield is plugged into (or stacked on top) of the Arduino board. The weather shield is plugged into (or stacked on top) of the wifi shield. This arrangement or “stack” requires that board and shields all be compatible with each other, both electrically and at the data layer.

The solar cell generates power that must be conditioned to both charge the backup, LiPo battery, and power the station micro-controller and attached shields. Solar cell current and voltage varies with the amount of solar energy falling on the solar cell. Clouds reduce the amount of solar energy falling on the cell, and, obviously, at night the solar cell generates no power. The charge controller provides conditioned voltage and current for both charging the backup battery and powering the micro-controller stack. The LiPo battery provides power during times when the solar panel is not generating enough power to run the micro-controller stack.

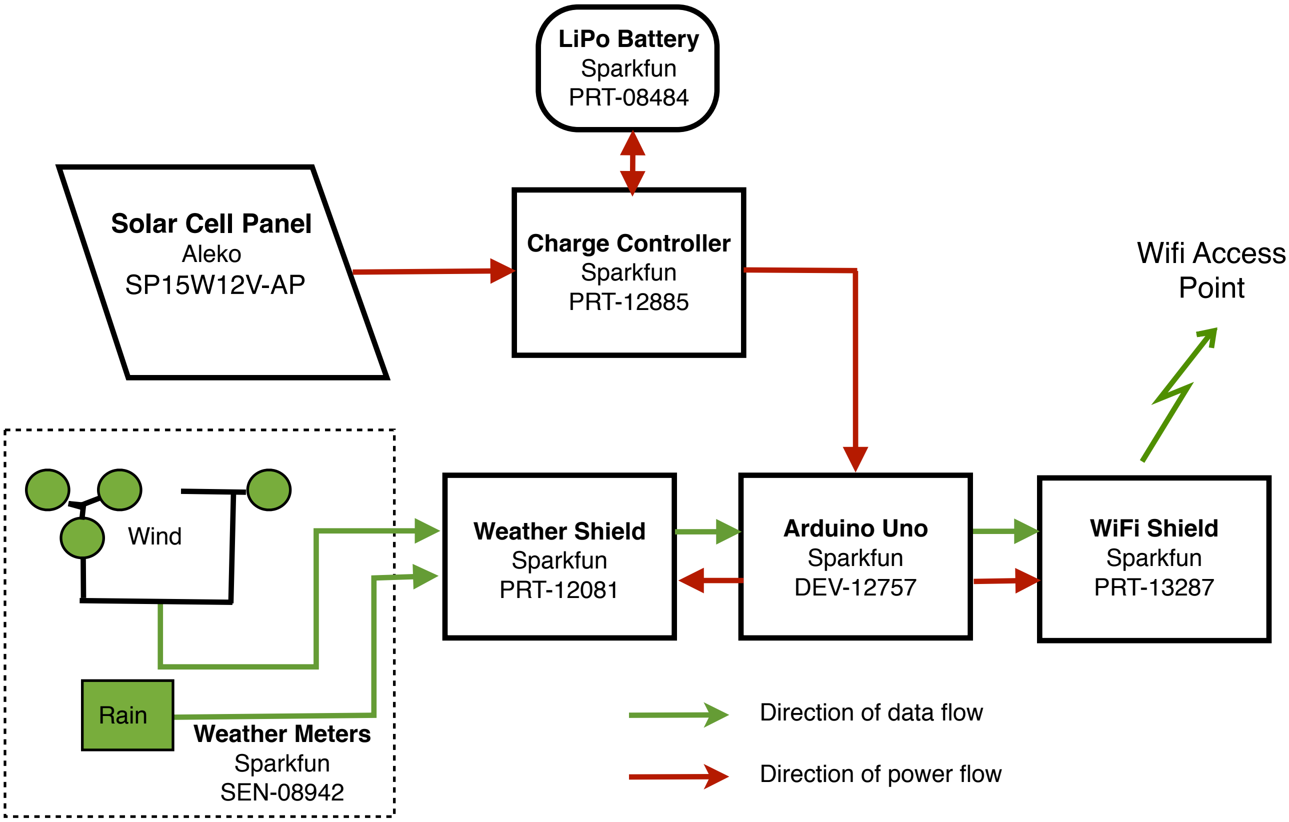

Figure 2. Block diagram of weather station hardware showing individual electronic components.

Weather Station Software

The software for the weather station micro-controller - the heart of the weather station - consists of an Arduino "sketch", along with several ancillary libraries that must be included at compile time. Referring to figure 3, the main module, or sketch, may be viewed as a "stack" made up of three layers. The top most includes the setup routine which runs once at boot up time, and the main loop that contains all the routines which run forever after initial setup.

Initial setup runs routines that initialize the weather sensors and the wifi interface. The main loop runs the 2nd level routines referred to in figure 3 as *Do Every Second* and *Do Every Minute*. *Do Every Second* triggers 3rd layer events that happen at intervals determined by one second marks, which includes *Update 1 Second Wind Gust*, *Update 2 Minute Wind Average*, and *Calculate and Report Weather Data*. Similarly, *Do Every Minute* triggers events that happen at one minute marks, which includes *Update 10 Minute Wind Gust* and *Update 1 Hour Rain Fall*.

Update 1 Second Wind Gust runs every second and compares the current wind speed with the wind speed one second earlier. If the current wind speed is greater, then both the wind gust speed and direction values get updated with the current values. The wind gust speed (and direction) is stored in a ten element array. After a minute elapses, the array pointer gets incremented, thus each element of the array contains the highest wind speed measured over a one minute interval. The largest wind speed value in the array determines the speed of the highest wind gust in the previous ten minutes.

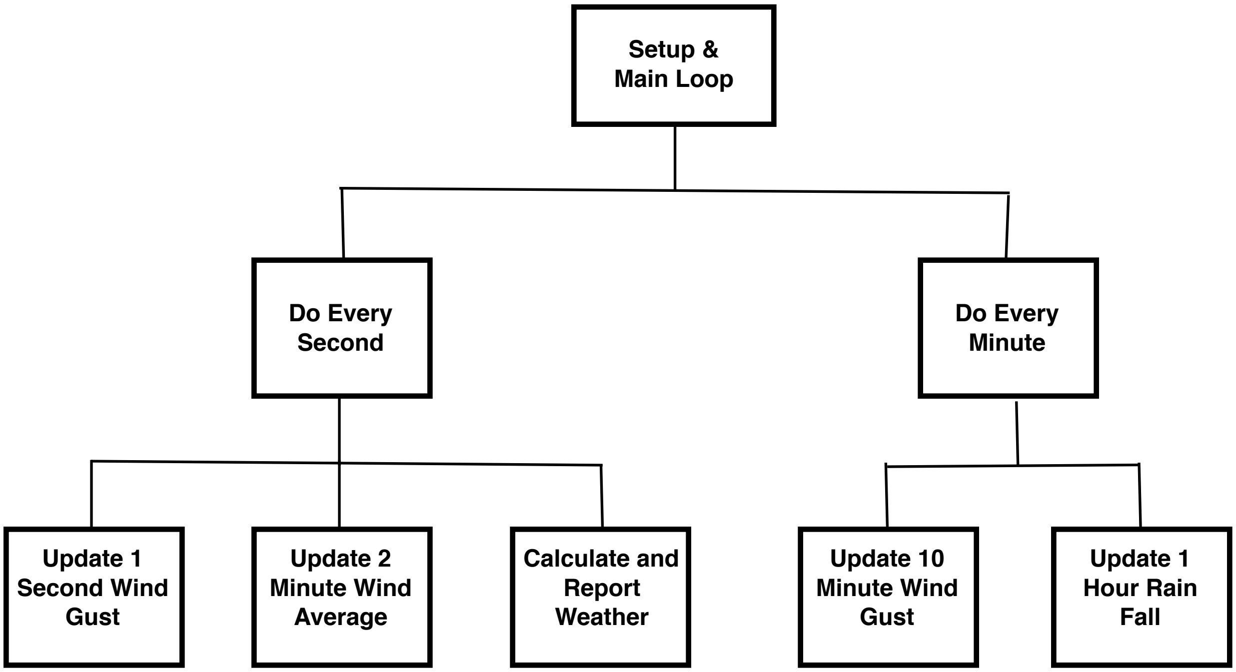

Figure 3. Micro-controller software block diagram showing high level components.

Update 2 Minute Wind Average runs every second and stores the instantaneous wind speed and direction in the 2 minute average wind array. When *Calculate and Report Weather* gets called the 2 minute average wind array is used to calculate average wind speed and direction for the 120 second period prior to when *Calculate and Report Weather* gets called.

Calculate and Report Weather, triggered periodically, calculates wind and rain, retrieves current readings from sensors, and sends weather data to the server. The data reported includes the highest wind gust for the day, the highest wind gust in the last ten minutes, the average wind for the previous 120 seconds, and the total rainfall for the previous 60 minutes. The sensor data reported includes the humidity (percent), barometric pressure (Pascals), and temperature (Fahrenheit).

Update 10 Minute Wind Gust, triggered every minute, increments a pointer to the ten minute wind gust array. Each element of the array contains the highest wind speed measured over a one minute period, the entire array covering a ten minute period. When the pointer reaches the bottom of the array, it gets “rolled over” back to the top of the array, thus implementing a “round robin” data structure. This method works since only the maximum of the array is needed, not the value of any individual element.

Update 1 Hour Rain Fall, triggered every minute, stores the amount of rain fall measured during that minute in the rain fall array and then increments the array pointer. Similar to the above, when the array pointer reaches the bottom of the array, it gets rolled over to the top of the array. Again, this method works since only the total amount of rain over the last sixty minutes is needed.

The above description provides an outline summary of the micro-controller program. The program, itself, involves quite a bit more: interrupt handlers for wind speed and rain fall sensors, functions for calculating instantaneous wind speed and direction, and a function for processing maintenance commands from the server.

Server Software

Referring to figure 4, the server software consists essentially of two components: an HTTP server component and an agent component. The HTTP server component includes a PHP script along with HTML documents containing embedded Javascript. The PHP script processes HTTP requests sent from the weather station. The agent, a Python script, manages data conversion and re-formatting, updating a database, and routine weather station maintenance.

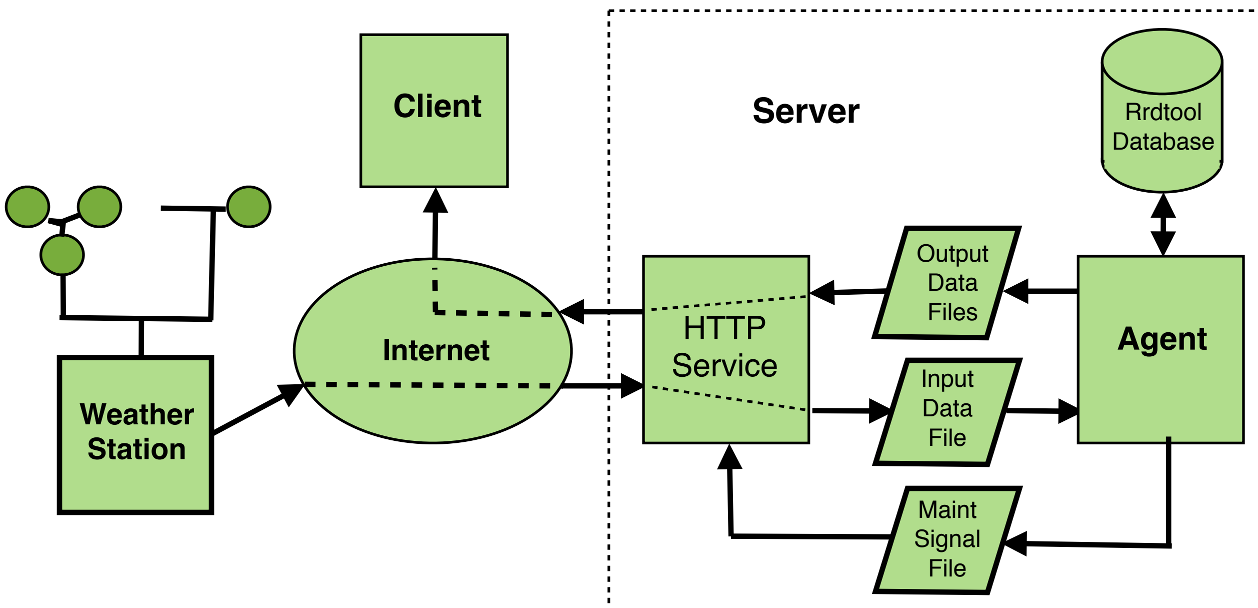

Figure 4. Block diagram of server software components showing data i/o, agent process, HTTP service, and database elements.

Events flow in the following manner. The weather station periodically sends data, via the network, to the HTTP service. Below is a typical example of the URL sent to the server by the weather station.

```
http://remoteServer/~user/weather/submit.php?wea=$,ws=10.0,wd=270,ws2=7.0,
wd2=270,gs=25.0,gd=180,gs10=12.0,gd10=270,h=51.0,t=76.8,p=101269.3,r=1.00,dr=
5.00,b=4.3,l=2.4,#
```

The PHP *submit.php* script, referenced in the URL, writes the data embedded in the URL to the *Input Data File*. The string between “$” and ‘#’ contains the embedded weather station data. The agent periodically reads the input data file and processes the data in that file. The agent converts specific data items to other physical units where required and formats the data for other services. Selected data items also get written to a round-robin database for permanent storage. Besides these functions, the agent manages generation of graphic charts for display in HTML documents, as well as sending periodic “reset” signals to the weather station.

The reset signal, sent exactly at midnight, resets the daily total rain fall and maximum wind gust back to zero. The agent writes the reset signal to the *Maintenance Signal File*. When the weather station sends data to the HTTP service, the *submit.php* script reads the maintenance signal file and sends the signal back to the weather station as part of its HTTP response message. Other maintenance features include the ability to remotely change wifi configuration and server URL.

After formatting the weather data, the agent writes the data to the *Output Data File* for use by HTML documents. When a client browser requests the weather HTML document, Javascript embedded in the document reads the output data file and displays this data in an HTML document. Although not shown in figure 4, the agent, as mentioned earlier, generates graphic charts for display in HTML documents.   The graphic charts are stored as image files which Javascript in the HTML documents can display in the weather web page.

Building the DIY Weather Station

Parts List

To build the weather station, see the list below for recommended parts and materials. Consult the supplier’s web site for detailed instructions on how to hookup and use individual parts.

Description	Quantity	Supplier	Part #
Red Board (Arduino Uno equivalent)	1	Sparkfun	DEV-13975
Weather Shield	1	Sparkfun	DEV-13956
ESP8266 Wifi Shield	1	Sparkfun	WRL-13287
Weather Meters	1	Sparkfun	SEN-08942
Sunny Buddy MPPT Solar Charger	1	Sparkfun	PRT-12885
Polymer Lithium Ion Battery 6Ah	1	Sparkfun	PRT-13856
Arduino Stackable Header Kit	3	Sparkfun	PRT-11417
RJ-11 6 Pin Connector	2	Sparkfun	PRT-00132
JST Jumper 2 Wire Assembly	1	Sparkfun	PRT-09914
Solar Panel 15 Watt	1	Aleko	SP15W12V-AP

Description	Quantity	Supplier	Part #
Solar Radiation Shield (housing)	1	Ambient	SRS100LX
Raspberry Pi 3, Model B+ (optional, if you need a server)	1	Adafruit	3775

Supplier Websites
Ordering information, detailed specifications, and hookup guides are available online at

- Sparkfun: sparkfun.com
- Ambient LLC: ambientweather.com
- Adafruit: www.adafruit.com
- Aleko: alekoproducts.com

Some additional hardware will be required, such as: a plastic card on which to mount the Arduino and shield assembly, solar charger, and LiPo battery. Also the weather meters, solar shield housing, and battery will need to be mounted on some sort of mast. (The author used an inexpensive camera tripod, available from amazon.com.)

For construction ideas and more details see the excellent weather station tutorial at https://learn.sparkfun.com/tutorials/weather-station-wirelessly-connected-to-wunderground.

Assembly Notes

1. The weather shield should be plugged into the wifi shield, and the wifi shield should be plugged into the Arduino board. This arraignment allows for the most free air flow around the pressure and humidity sensors and distances the temperature sensor from possible heat sources on the Arduino board and wifi shield.
2. The Arduino-wifi-weather-shield stack should be mounted, along with the solar charger and battery, on a plastic card such that the entire assembly can be easily slid into the solar shield housing. See the Sparkfun weather station tutorial for details.
3. If enabled, the Arduino sketch can make use of the ESP8266 wifi shield's deep sleep mode for energy conservation. In order to use this mode a jumper must be installed on the wifi shield between the 'XPD' pin and 'RST' pin.
4. The power output leads coming from the charge controller should be connected directly to the 5 Volt and GND pins on the weather shield header. (See figure 5.) While the Sparkfun Redboard, weather and wifi shields will all work powered from a 3.8 volt LiPo battery, the power must be connected directly to the "5 Volt" rail. Powering through the VIN pin requires an input voltage of 8 to 16 volts, which the LiPo battery cannot provide.
5. Average daily solar energy falling on a solar cell varies greatly with latitude and climatic conditions. The selection of a 20 Watt solar cell is based on the climatic conditions of western Oregon, where periods of heavy overcast skies can last for days. During such times typical solar cells produce as little as 5% of their rated output. The solar cell must be sized to provide adequate output to both power the

electronics and keep the battery charged. Desert areas subject to daily clear skies may require only a 5 Watt (or less) solar cell.

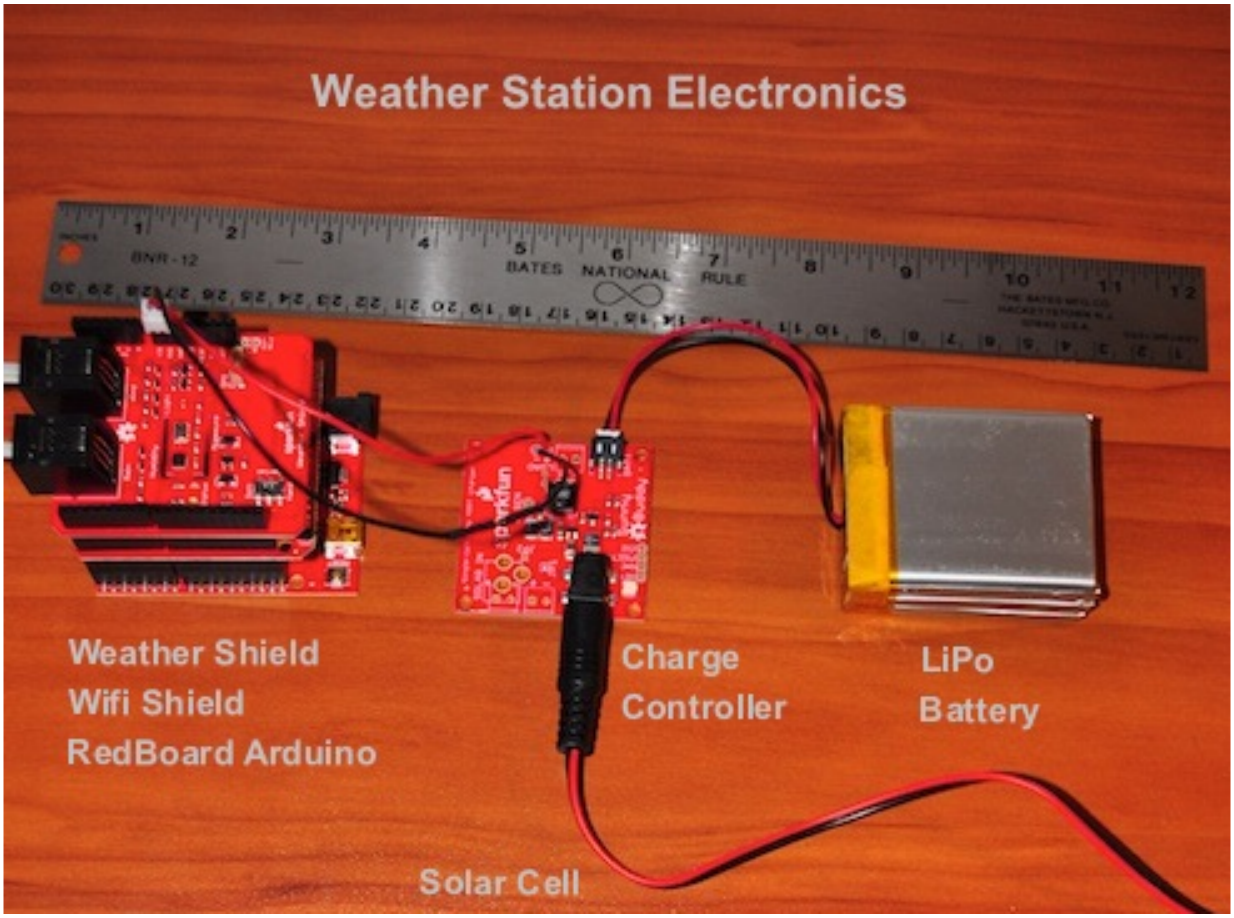

Figure 5. Photograph showing weather station electronic components. Note how the weather shield and wifi shields are plugged into the Arduino board on the bottom.

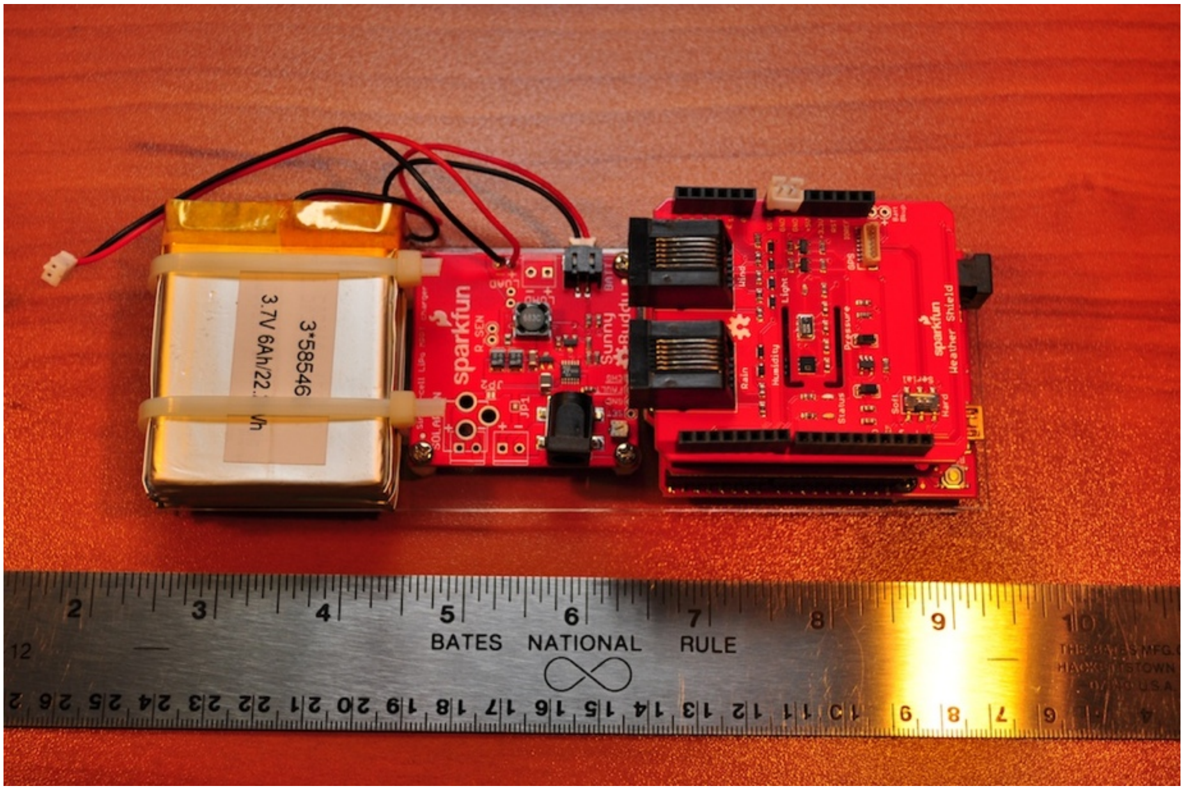

Figure 6. Photograph showing weather station electronic components. Note how the weather shield and wifi shields are plugged into the Arduino board at the bottom.

Figure 7. Photograph showing how weather station electronics fit inside the Ambient solar radiation shield assembly.

Software Installation

1. Get the latest version of the Arduino IDE from https://www.arduino.cc/en/Main/Software. The IDE is available for Mac, Windows, and Linux operating systems.
2. Before the weather station sketch can be compiled, three device driver libraries must be installed. When you run the Arduino IDE for the first time, the IDE creates a folder **Arduino/sketch/libraries**. Look for this folder in your home documents folder. Download the two sensor libraries from Sparkfun's web site at https://cdn.sparkfun.com/assets/b/5/9/7/f/52cd8187ce395fa7158b456c.zip. Unzip the downloaded folder and copy the HTU21D humidity sensor and MPL3115A2 pressure sensor library folders into your **Arduino/sketch/libraries** folder.
3. Download the ESP8266 wifi shield device driver library from https://github.com/sparkfun/SparkFun_ESP8266_AT_Arduino_Library/archive/master.zip. Unzip the downloaded folder, and place the unzipped folder in your **Arduino/sketch/libraries** folder.
4. If you are using the Sparkfun Arduino Redboard as the micro-controller, consult Sparkfun's web site for details on using the Redboard with the Arduino IDE. If you are installing from a Mac or Windows, you may need to install some custom USB drivers from Sparkfun.
5. Using a USB cable connect the computer running the Arduino IDE to the Arduino's USB port. *Be sure to disconnect the power leads from the charge controller before connecting the USB cable*.
6. Follow the Arduino instructions for connecting the Arduino to your computer and configuring the IDE for the Uno. Normally this means navigating to the Tools->Board menu item and selecting the Uno, and going to the Tools->Port menu item and selecting USB port to which the Uno is connected.
7. Before the **WeatherStation** sketch can be compiled and downloaded to the Arduino, the Arduino EEPROM must first be initialized with your wifi configuration and server URL. From the Arduino folder in the github project folder, download the **SetupEeprom** Arduino sketch and open it in the Arduino IDE.
8. Look for the preprocessor defines at the top of the file. These defines will need to be edited. Look for the following three lines

   ```
   #define DEST_SERVER "your server URL - see instructions"
   #define SS_ID "your wifi access point's SSID"
   #define WPA_PASSWD "your wifi access point's WPA password"
   ```

   SS_ID and WPA_PASSWD are self-explanatory, however, DEST_SERVER requires some additional explanation. The DEST_SERVER string should look something like

   "myServer.com/~user/weather/submit.php"

   where "/weather/submit.php" is dependent on how you install the software on your server. If you follow the instructions in this document, then you should replace only "myServer.com" with the name of your server and "{user}" with your user account. *The destination server string should NOT be prefixed with "http://".* It should be exactly as shown, with your server name replacing "{myServer.com}" and your user name replacing "user". An IP address can be used instead of a qualified domain name. For example,

10.10.168.155/~user/weather/submit.php

Note that at a minimum the server name (or IP address) and PHP script have to be in the string. For example, if you install the html documents and the submit.php script in the default /var/www folder, the following would be a valid destination server string

myServer.com/submit.php

9. Compile and download the sketch. If the sketch is running properly, in the IDE serial monitor window you should see status messages followed by your destination server, wifi ssid and password.
10. From the Arduino folder in the github project folder, download the WeatherStation Arduino sketch, and open it in the Arduino IDE.
11. Compile the sketch and download it to the Arduino. Now complete the next section “Installing the Server Software”, and then return step 12 below.
12. Once the server software is installed, you can verify if the sketch is running properly by opening up the Arduino IDE’s serial monitor window. You should see the startup message followed periodically by the data string the Arduino sends to the server.

This completes the weather station software installation.

Installing the Server Software

Dependencies

- The server software should be installed on a recent Linux distribution such as Debian or Ubuntu. (The author has successfully developed and installed the software on a Raspberry Pi running the Raspbian operating system.)
- Apache2 should be installed and configured to allow serving HTML documents from the user’s public_html folder.
- PHP should be installed and configured to allow running user PHP scripts from the user’s public_html folder.
- Rrdtool should be installed - type **sudo apt-get install rrdtool**
- Python 2.7 usually comes pre-installed in virtually all Linux distributions. Type “python” at a command line prompt to verify Python has been installed.

Software Inventory

The software items that need to be installed on the server are, from the github project site **client_model** folder

HTML folder:

- weather.html
- index.html
- submit.php
- maintsig
- static/chalk.jpg

Agent folder:

- createWeatherRrd.py
- weatherAgent.py
- startwea
- stopwea

The above software items may be found in the respective folders on the project web site.

Installation
Note that the following installation procedure assumes that weather HTML documents will reside in the user's public_html folder. Typically the full path name to this folder will be something like "/home/user/public_html".

1. In the public_html folder, use **mkdir** to create a folder **"weather"** to contain the weather HTML and PHP files.
2. In the weather folder created in step 1, use **mkdir** to create a folder **"dynamic"** to contain dynamic content used by html documents.
3. From the html folder in the github project folder, download all files and folders to the folder created in step 1.
4. The input and output data files (see figure 4) get overwritten frequently; similarly the graphic image files get overwritten frequently. On SD card systems, such as a Raspberry Pi, it is inadvisable to do frequent writes to any file system mounted on the SD card. To use the ram based temporary file system (tmpfs) to store these files, continue with step 5. Otherwise, to write dynamic content to the disk drive, continue with step 7.
5. Assure that the server /tmp folder gets mounted to the temporary file system. This can be done by adding the following line to the **/etc/fstab** file

   tmpfs /tmp tmpfs nodev,nosuid,size=50M 0 0

6. The HTTP service cannot freely access files and folders outside of the public_html document root folder. Therefore, the dynamic folder must be bound to the temporary file system by running the following commands

   mkdir /tmp/weather
   chmod g+w /tmp/weather
   sudo mount --bind /tmp/weather /home/USER/public_html/weather/dynamic
   sudo chown :www-data /tmp/weather

   The above four commands may be placed in a startup shell script and run at boot up time by launching the startup script with the **su** command from **/etc/rc.local**, e.g.,

   (su - pi -c "bin/startup.sh")&

7. Change the group ownership and permissions of the **maintsig** file by running

    sudo chown :www-data maintsig
    chmod g+w maintsig

    Note that although requiring superuser permissions to do so, the above provides a far more secure environment than merely enabling write permission for everyone. Entries in this file can change the weather station's wifi configuration and/or destination server, potentially making it totally unreachable. Once the weather station becomes unreachable, it can only be reconfigured by physically connecting to the Arduino and working through the steps described in the previous section.
8. If it does not already exist, use **mkdir** to create a folder named "bin" in the user home folder. For example, the full path name should look like "/home/user/bin'.
9. From the bin folder in the github project folder, download the files into the bin folder created in step 8. In most Linux installations the user's bash profile will automatically add the user's bin folder to the command search path. If such is the case, then the agent can be started up by simply typing **weatherAgent.py** followed by ENTER.
10. In the user home folder create a folder named "database". In the bin folder run the python script **createWeatherRrd.py**. Running this script creates an empty round robin database file where the agent will store weather data as it arrives from the weather station. This script should be run once and then kept in a secure place. Running it accidentally at some future date will result in *total loss of all previously stored data*.
11. In the user's home folder create a folder named "log". For convenience two scripts have been provided to make it easy to turn the agent on and off. The **startwea** script starts up the agent and causes all diagnostic output and error messages to be written to a log file in the log folder. The **stopwea** stops the agent from running.

This completes installation of the server software.

References and Resources

The Sparkfun hookup guides provide a wealth of detailed application notes for Sparkfun products. The following hookup guides should be consulted before building the weather station:

- Weather Shield https://learn.sparkfun.com/tutorials/arduino-weather-shield-hookup-guide-v12
- ESP8266 Wifi Shield https://learn.sparkfun.com/tutorials/esp8266-wifi-shield-hookup-guide
- RedBoard https://learn.sparkfun.com/tutorials/redboard-hookup-guide
- Sunny Buddy Solar Charger https://learn.sparkfun.com/tutorials/sunny-buddy-solar-charger-v13-hookup-guide-
- Weather station tutorial https://learn.sparkfun.com/tutorials/weather-station-wirelessly-connected-to-wunderground

The following tutorials are useful for more in depth understanding of programming Arduino:

- C programming https://www.gnu.org/software/gnu-c-manual/
- Arduino programming https://www.arduino.cc/en/Tutorial/Foundations

The following tutorials are useful for more in depth understanding of the server software:

- Javascript http://www.w3schools.com/js/default.asp
- PHP http://www.w3schools.com/php/default.asp
- HTML http://www.w3schools.com/html/default.asp
- Rrdtool http://oss.oetiker.ch/rrdtool/
- Python http://greenteapress.com/thinkpython/thinkpython.html