Final Report: The SmartMirror

Chris Rectenwald, Teague Kohlbeck, and Benny Richmond
EE 41440- Senior Design 2
Professor Mike Schafer

Table of Contents

3. Introduction

Presentation and punctuality are two of the most valued qualities in modern society. However, it can be difficult to effectively prepare for the day while remaining knowledgeable about current affairs and still maintain a timely schedule. In the morning, it is imperative to prepare for the day in front of a mirror, which is often a slow process. Additionally, factors such as the current weather conditions can influence how a person prepares for the day. Finding an efficient way to check all the factors that can affect how a person prepares for the day while also not adversely affecting the tasks that are performed in front of a mirror can be a challenge. The goal of our project was to create a product that will provide quick and easy access to the time, news, and weather while simultaneously allowing a person to go through their morning routine. Our product should enhance productivity while providing a functional and enjoyable user experience.

3.1 Problem Statement and Proposed Solution

The world we live in today has become a place of the fiercest competition, whether it is in sports, entertainment, or the job market. In order to be the best, one needs to allocate an extraordinary amount of time to their goals with little distraction. However, the advent of information technology tends to act like a dual-edged sword when it comes to work productivity; sometimes one can use the ease of information to help them complete a task, but it can also provide significant distraction. Ultimately one strives to be their best, but the interruption of keeping up with the daily news, or preparing for incoming weather can hinder one’s progress. Taking time throughout the day for these various activities can be extremely distracting and greatly cut into performance.

Along with information, people greatly value their appearance, spending approximately an hour a day in front of the mirror during their morning and night routines. This is a significant amount of time where important things are taking place, but the mind is not working. It would be extremely useful to spend that time on the phone or computer completing any of the tasks mentioned above, but unfortunately it is difficult to do so while preparing for the day. A product is needed that can allow a person to efficiently complete everything they need to do to prepare for the day, all in one place and at the same time.

The goal of the SmartMirror is to provide a single easy to access location for a person to receive all the information that could affect how they prepare for the day. Through the

use of LCD displays and a two way mirror, weather, time and date, and news are available at a glance. Additionally, a user friendly interface, accessible from any WiFi enabled device, allows the user to easily setup the connection to their home WiFi, change the location from which they receive the weather, and select a source from which to receive the day's headlines. By building these features into a mirror, which most people will already be using in their morning routine, it is possible to present this information in such a way that it will seamlessly blend together with the task of morning grooming.

3.2 How Our Design Compares to Goals

In our final design, we have included the capability for up to four displays mounted behind the mirror. This allows the time, date, and weather to be displayed constantly when the mirror is on, along with a screen that scrolls through the headlines. By flipping a switch mounted on the side of the mirror and pressing the reset button, the user is able to toggle between the default mode, which will connect to WiFi and parse information from the internet, or the setup mode, which creates a hotspot that the user can connect to from any WiFi enabled device to switch the SSID, password, zip code, and news source. The electronics will go into sleep mode if there is no motion detected for 5 minutes, which helps to reduce light output and power consumption when nobody is using the mirror. The two-way mirror film coated plexiglas creates a durable and functional mirror and the built in Bluetooth speaker allows the user the option of playing their wake up jams directly from the mirror. Overall, the design that was created serves to create an environment that promotes efficient daily preparation, allow easy access to weather information and news headlines, and provides an user friendly and enjoyable mirror experience.

4. System Requirements

The microcontroller used to control the mirror and the LCD displays will need to be powered by a 5V power supply. This will not need to be portable as the mirror will be stationary after its initial installation. A 3.3V regulator will need to be included for some components. Therefore, a 5V wall wart power supply and an on chip voltage regulator are the best choices to meet the power requirements. The embedded system used in the mirror will need to be able to interface with a WiFi interface as well as multiple LCD displays. The WiFi will need to be able to work within a home, so it should be sensitive

enough to pick up the signal from a home WiFi router. The system will need to be able to take user input to program the user's location, initial SSID information, and switch what data is being displayed on the auxiliary screen. There will need to be communication from a motion sensor, which will need to sense motion up to about 10 feet in front of the mirror, to determine if a person is present and the system needs to be woken from its sleep state. The displays must be able to show all relevant text and icons, and refresh quickly enough to allow for regular information updates. Overall, all the code must be compact enough to fit and run on the microcontroller we plan to use. The mirror will be around 1.5’ by 1.5’, so all the hardware will need to fit within this profile and the displays will need to be properly scaled. The entire system will also need to be mounted on a wall, so the structure will need to be durable enough to support this weight. The hardware must be able to be mounted securely and neatly to the back of the mirror so that it will hold its position when mounted on the wall.

5. Project Description

5.1 Overall System

The overall system has been broken down into the 5 subsystems: The On/Off Subsystem, Internet Subsystem, User Interface Subsystem, Speaker Subsystem, and Display Subsystem. The connectivity of the entire system can be seen in the figure below. The User Interface Subsystem takes user input to determine the SSID, password, zip code, and a news source from which to display the headlines and writes it into nonvolatile memory. This information is read out of memory and is out of memory and passed to the Internet Subsystem. The Internet Subsystem connects to the internet via WiFi and parses the time, weather, and news headlines from the sources selected in the User interface. The data parsed by the Internet Subsystem is saved into arrays that can be printed onto the screens by the Display Subsystem. The On/Off Subsystem runs off a motion sensor and determines whether the system is in sleep mode or is actively pulling and displaying data to the screens. The Speaker Subsystem will be available to be connected to via Bluetooth whenever the overall system is turned on.

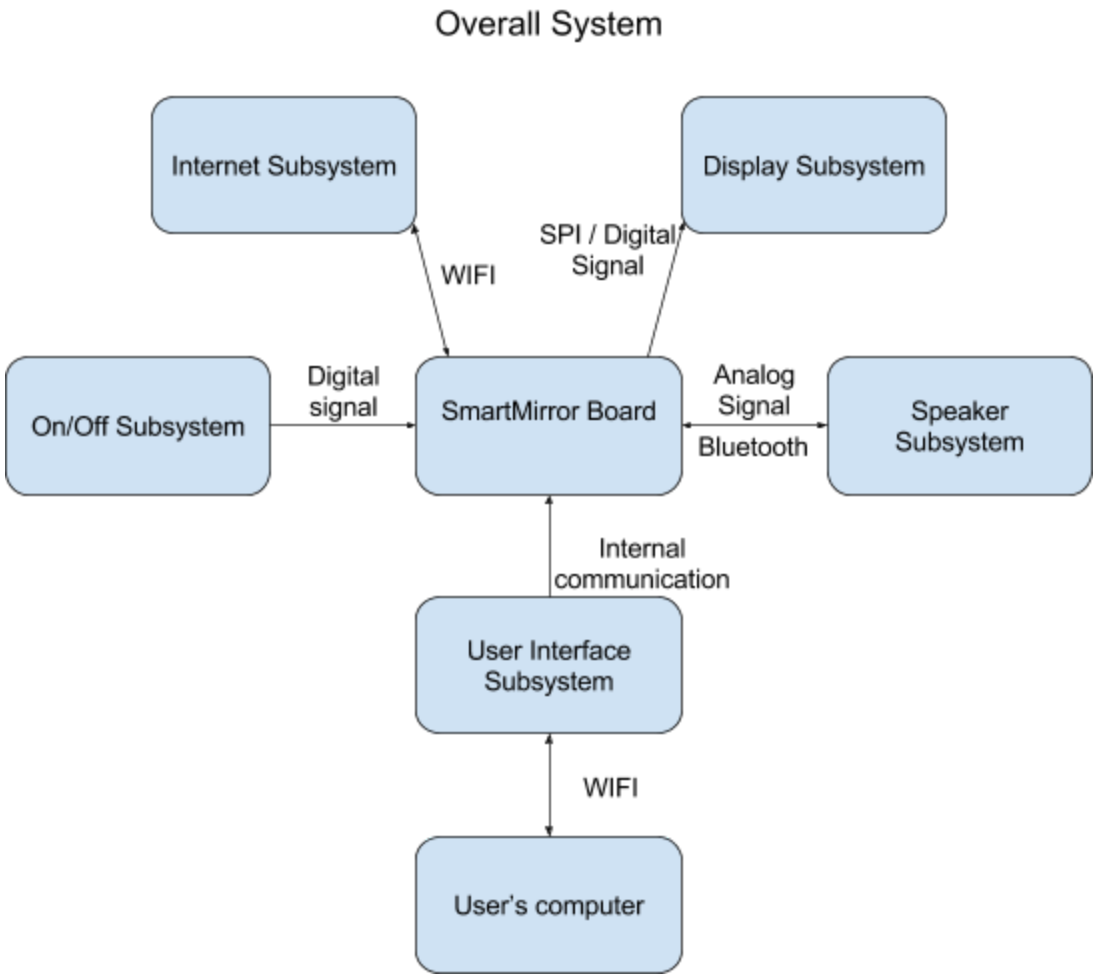


Figure: Overall System Flow Diagram

5.2 Operation and Design of On/Off Subsystem

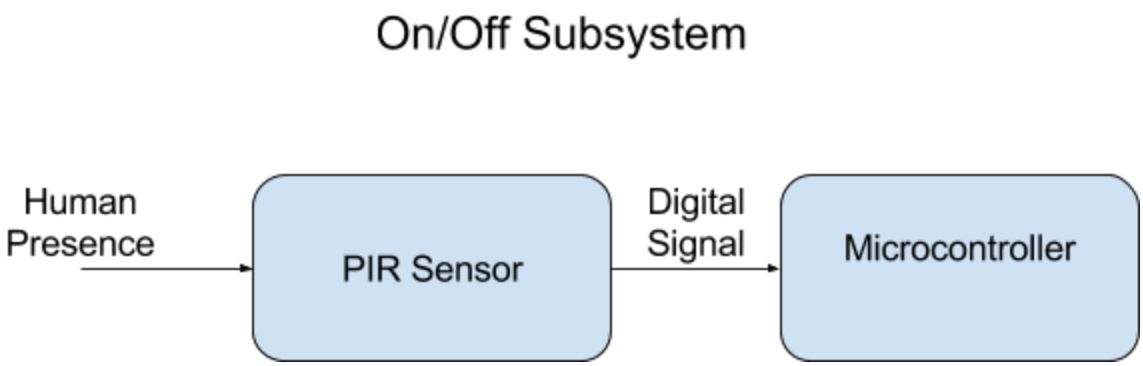


Figure: Flow Diagram of On/Off Subsystem

This system overall deals with determining whether or not there is a person in front of the mirror or not. As of right now the group plans to use HC-SR501 PIR Motion Detector to detect a human movement. We chose the PIR Motion Detector over other options such as a photocell because of a PIR Motion Detector is not too aesthetically displeasing, while triggering a high or a low voltage rather than current. We plan to interface the PIR Motion Detector with the Arduino Uno but could be interfaced with an ESP microcontroller. All of these choices would suffice as their are examples in the Arduino IDE to help us get through any potential roadblocks along the way.

Moving forward with this system we will need to consider how to effectively put a current-draining resistor on the mirror in order to get rid of static voltages. We also need to consider the time-delay and sensitivity adjustments as seen on top of the figure below that are available with this PIR Motion Detector as well as a repeat or single trigger option that is in the red box in order to provide the most pleasant user experience.

So far we have tested the PIR sensor in the Arduino IDE by digitally reading the voltage level from the middle prong at the bottom of figure below and outputting whether there is motion detected accordingly. After seeing that the voltage level changed based off if one's hands had passed the capacitors on the PIR Motion Detector or not, we concluded that the PIR Motion Detector works.

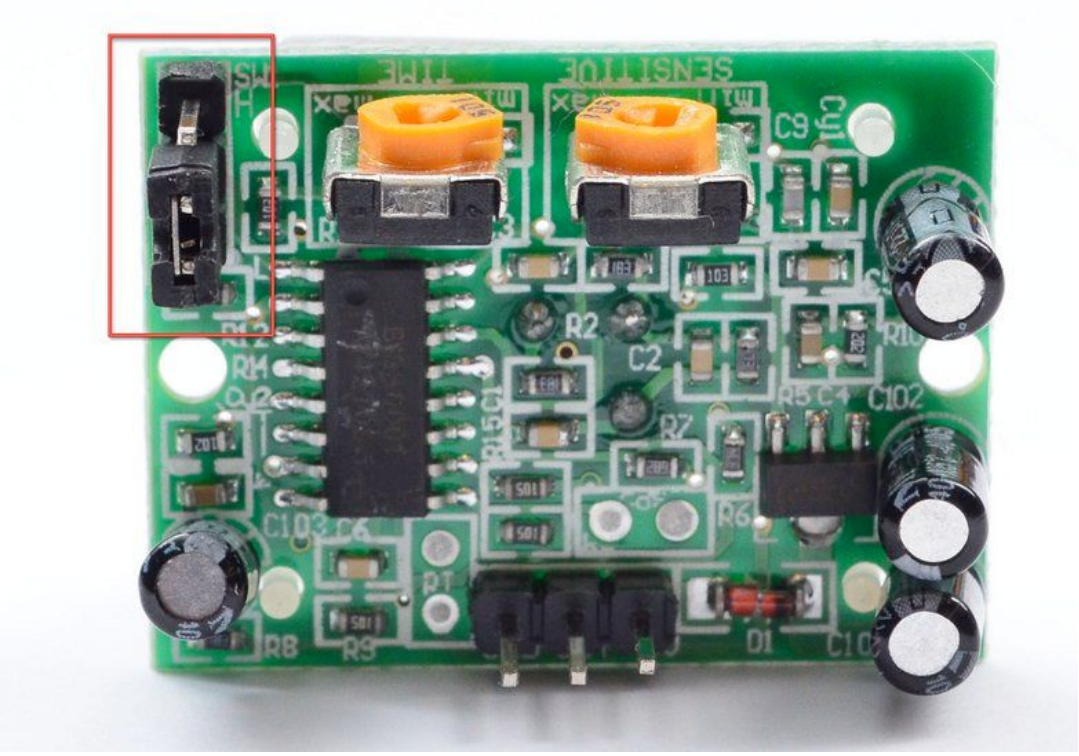


Figure: PIR Motion Detector

5.3 Internet Subsystem

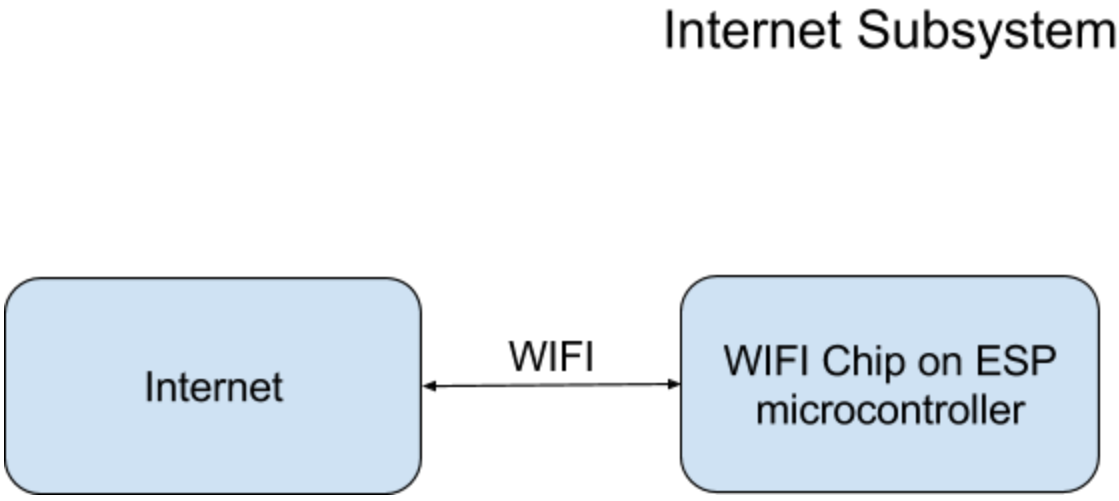


Figure: Flow Diagram of Internet Subsystem

The internet subsystem deals with being able to retrieve information such as weather, time, news or any other applicable information from the internet to be displayed on the

mirror. An ESP microcontroller or similar device will be the device pulling the data over a WiFi signal and communicate via I2C/SPI to the LCDs on the mirror.

Our choice for considering both of these device is because of their ability to connect to the internet while still have the capability of communicating with I2C/SPI to the mirror. We chose to use the Arduino IDE for this subsystem, as it is flexible in ways to retrieve information with either using get http functions or parsing with javascript. For instance in order to retrieve the time, we are able to connect to a Network Time Protocol that would give us the Greenwich Mean Time, and would translate from the time accordingly based on the user's location.

We also wanted to retrieve the local weather data, so we decided to use Wunderground's API for any location that is based off the user's inputted zip code. Wunderground provides a webpage of the data relevant to the user's needs including the location, time, and weather in javascript. Arduino allows us to use the "ArduinoJson.h" library that allows one to easily parse data from character arrays using javascript. Although ideally we plan for the zip code to be read from the user's setting on a website, to test the subsystem we hard coded the zip code to be one of South Bend's zip codes 46617. An illustration of the software program for this subsystem, Pull_Data.ino, can be seen in the Figure below.

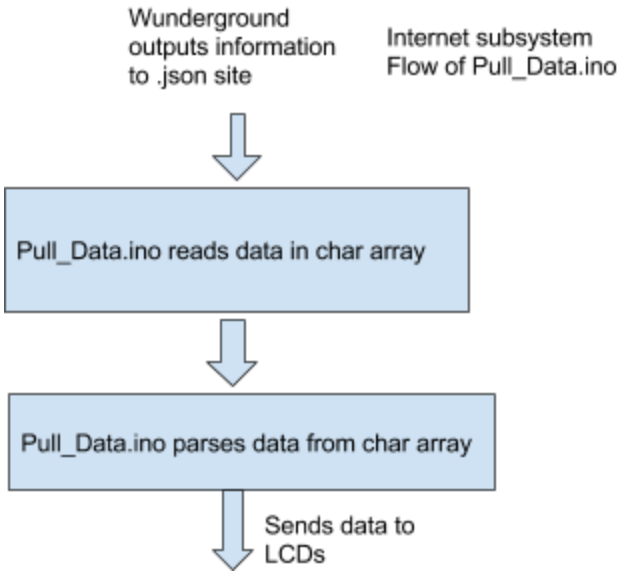


Figure: Flow Diagram of Software for Internet Subsystem

5.4 User Interface

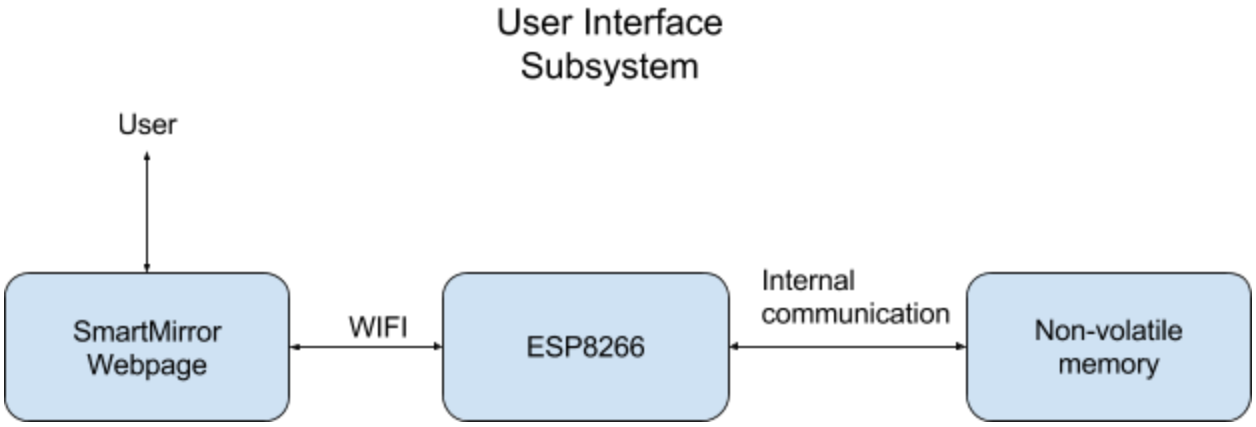


Figure: Flow Diagram of User Interface Subsystem

The user interface subsystem allows the user to update the settings of the mirror to connect to their home wifi and display data relevant to their geographical location. The ESP microcontroller must be configured to run as an access point server upon initial boot up as well as when the user desires to edit the settings of the mirror, as well as store the settings in the non-volatile memory of the chip.

The ESP microcontroller was chosen due to its ability to be configured as an access point as well as its ability to connect to the WiFi. The user is required to enter the ssid and password to allow the chip to connect with the WiFi as well as the user's zip code. The zip code was chosen as a parameter as it will allow for the weather and time for the user's location to be pulled by the internet subsystem and reduces the risk for the incorrect location to be input. The system is set to act as an access point upon powering the board on. After being powered on, it will check to see if there are valid credentials to connect to WiFi. If there are, then the chip will move on to the main function, if not, it will create an access point to input WiFi credentials. This was chosen as the mirror will be stationary and thus the information should remain the same after the initial boot up. If the location changes, the chip will not be able to connect to the network and will generate an access point to adjust the settings.

The ESP8266 was programmed using the Arduino IDE and utilizing the "ESP8266WiFi.h", "ESP8266WebServer.h", and "EEPROM.h" libraries. These were chosen as they are well documented and allow for custom html form data to be added. This enabled the for textboxes and dropdown menus to be added to allow the zip code, SSID, password, and additional information to be input. When access point mode is

initiated, a WiFi hotspot starting with “ESP” should become available. When prompted for a password, it is necessary to input the password “password” to connect to access point. Once logged on, the user interface will be available at the chips default IP address of 192.168.4.1. After entering data into the interface, the data will be saved as a string and then written into EEPROM. Additionally, these allow for a pleasant and user friendly interface to be accessed at the default ip of the chip, as shown below.

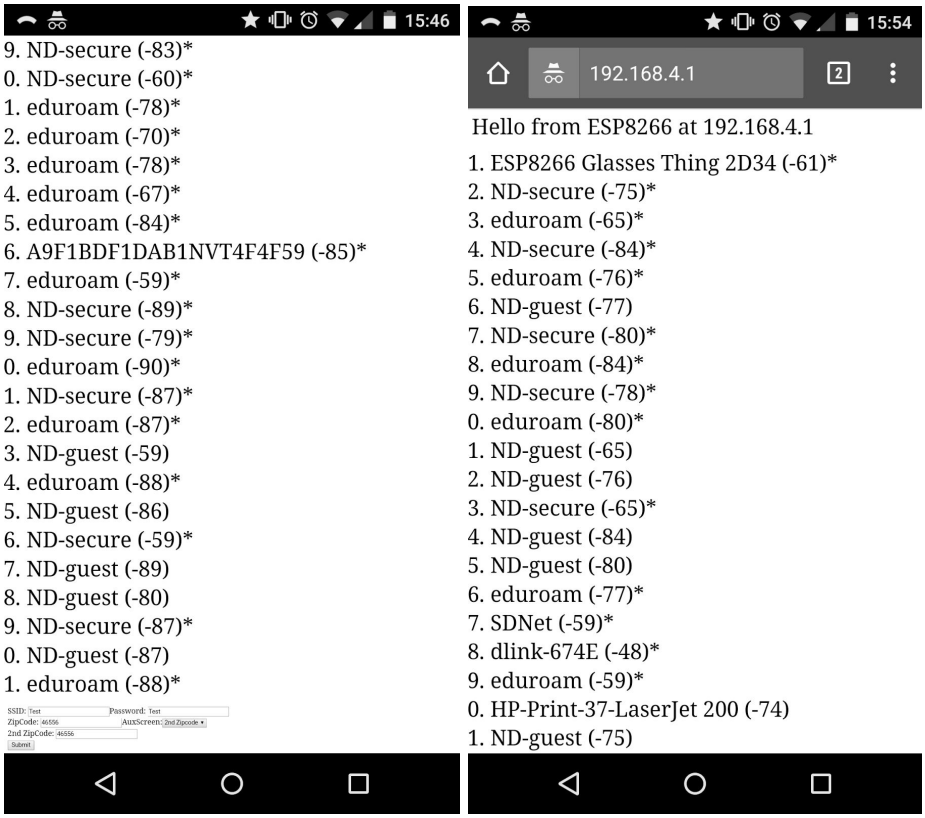


Figure: User Interface Web page

To test the system, the code was uploaded to the chip and the web page was accessed from a mobile device. After the after the data was input, as the data was being input into EEPROM, what was being written into EEPROM was printed to the monitor. The chip was then powered off and then powered back on and the string was read out of EEPROM. To ensure the correct data was saved, the strings from EEPROM were printed to the monitor.

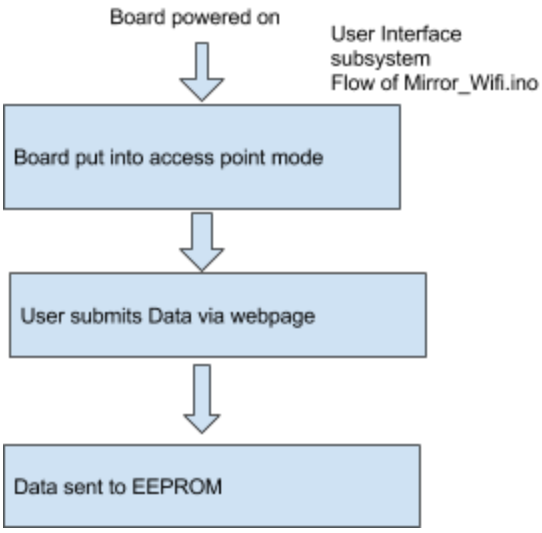


Figure: Flow Diagram of Software for User Interface Subsystem

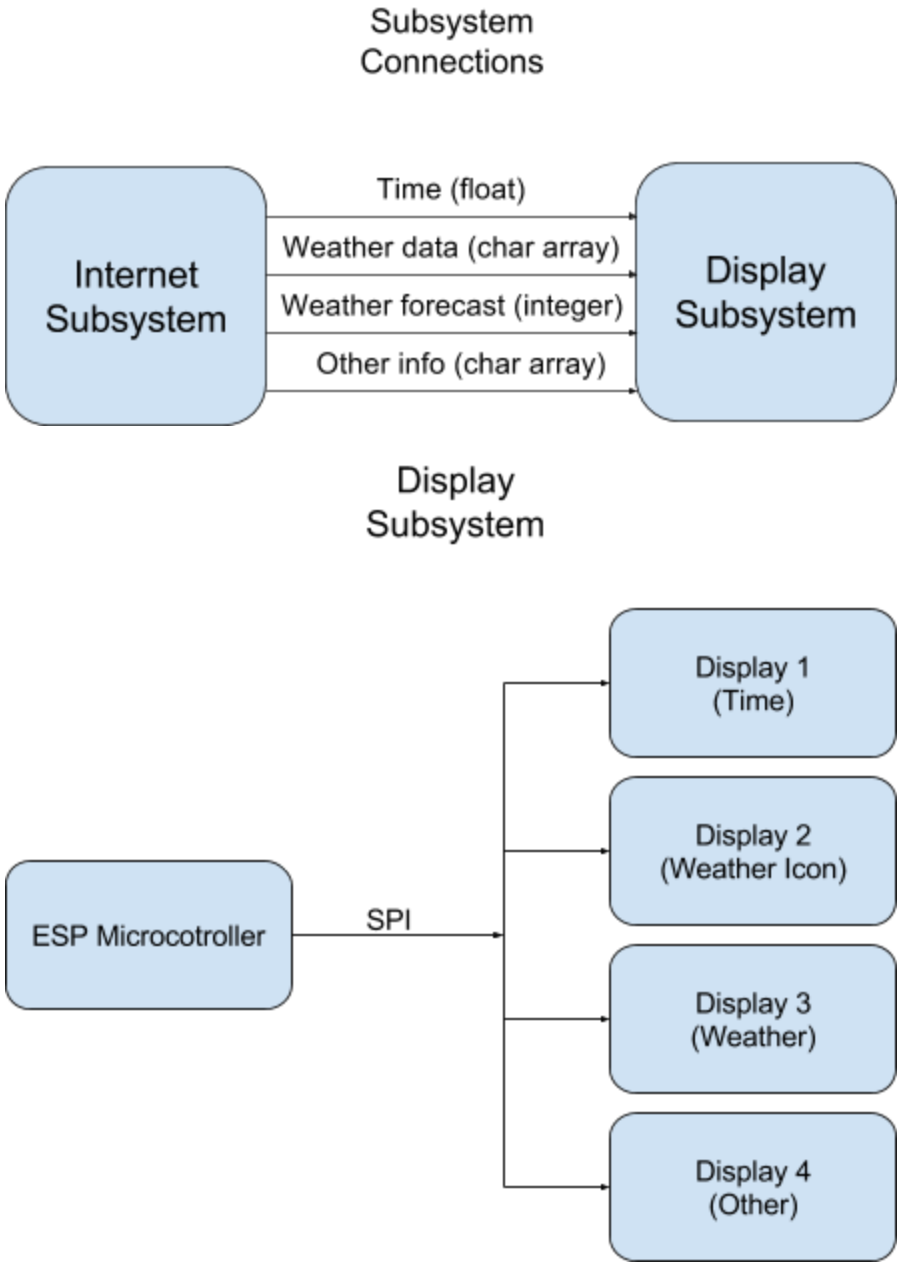
5.5 Display Subsystem:

This subsystem takes information retrieved by the Internet Subsystem, creates the appropriate textual outputs, and sends the desired information via SPI to each of the tft LCD screens. The settings used to organize the display of data will be based on the user's last input to the user interface subsystem. The system will continually exchange data with the internet subsystem to update the displays regularly.

There will be four display screens in this subsystem. All are Adafruit brand tft LCD displays, chosen for their ease of use and flexibility. The first screen is a 1.8" display, and will be used to show the current time. The second screen will be another 1.8" display that will show an icon relating to the current forecast. The third screen will be a 2.8" display, and will be used to show the daily weather. The fourth screen will be another 2.8" display, and will be used to show further information. One of the 1.8" displays was purchased with a breakout board and header pins included for ease of use, but the other three displays will only have surface mount cables.

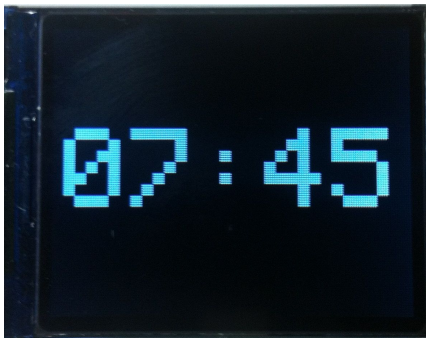
The displays will be controlled from the ESP microcontroller. Connection with the display hardware will be done with the ST7735 library and ILI9341 library for the 1.8" and 2.8" displays respectively, and graphics functionality will be achieved with the Adafruit GFX library. All three libraries have been modified to reduce the memory requirements of the microcontroller, as many of the functions included are not used. Each display has five wires running to it: power, ground, chip select, MOSI, and clock. The only unique wire for each is the chip select, all others are shared. The

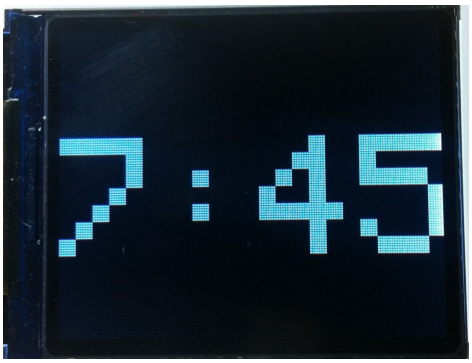
microcontroller sets which board will be communicated with via the chip select pin. Diagrams of the subsystem connections are included below.



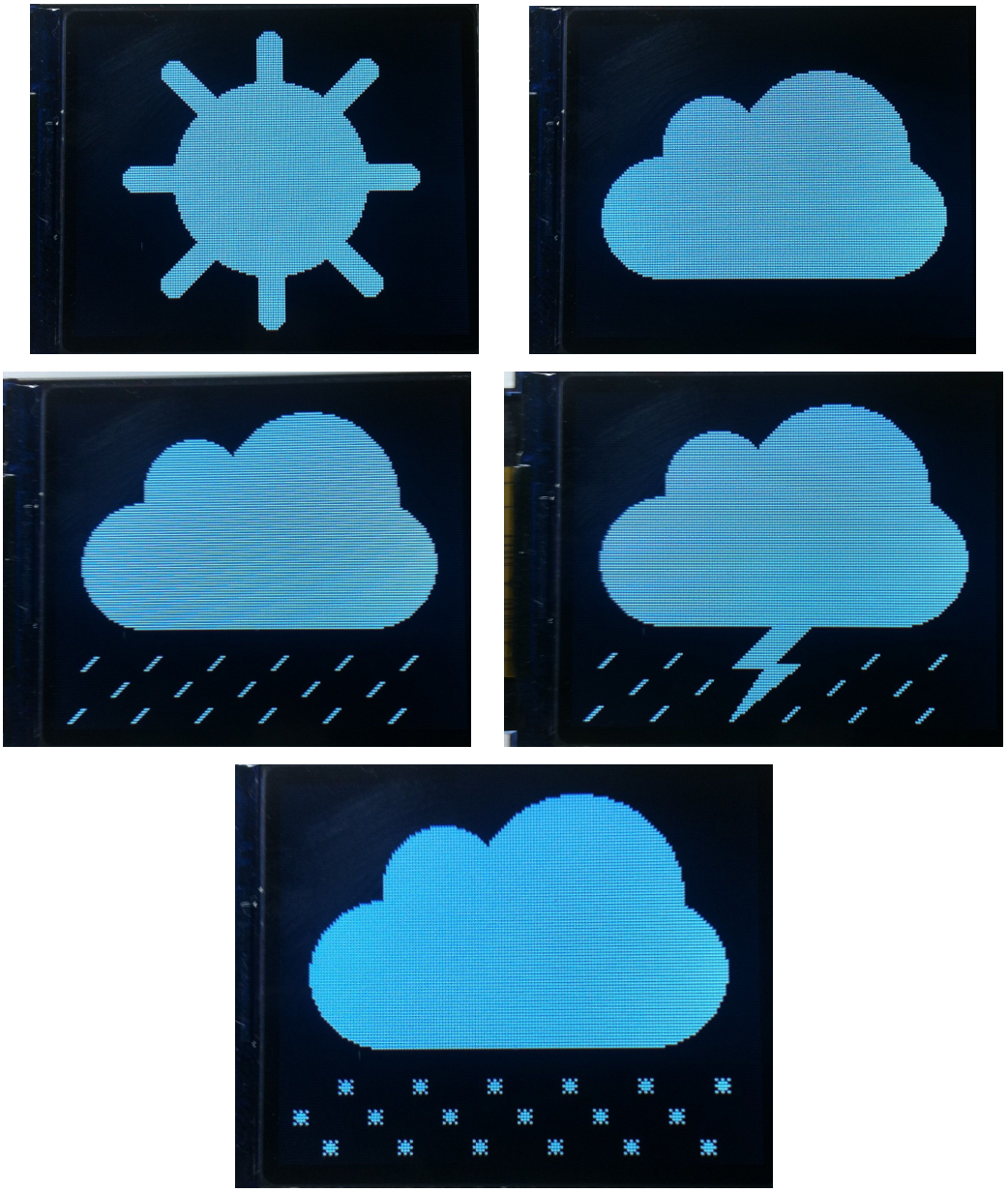
Display 1 is set up to receive a float variable of the current time from the internet subsystem. The float is but in a useable buffer, and a colon is added to make the time easier to understand.The time will be available in either a 12 or 24 hour format, depending on the setting the user chose during setup. Time will always come into the

subsystem in 24 hour format, but if 12 hour format is required the system will update the value accordingly. Additionally, when the time is only three digits long, the text will automatically be scaled up and repositioned to allow for the largest and most visible information possible. Shown below are displays of a full 4 digit time, a three digit time with padding, and a three digit time with size increased.





Display 2 is set up to receive an integer from 1 to 5, where each number represents a current weather condition. The possibilities are sunny, cloudy/overcast, raining, storming, and snowing. An icon depicting the condition will be shown on display 2. To save space on the microcontroller and increase execution speed, the icons were drawn manually instead of being loaded from bmp files. This was done by positioning various elementary shapes into buffers using the adafruit gfx library. Circles, triangles, lines, and individual pixels were all utilized. Examples of each icon can be seen below (take note that the coloring looks slightly off in photos, but the images are pure black and white).

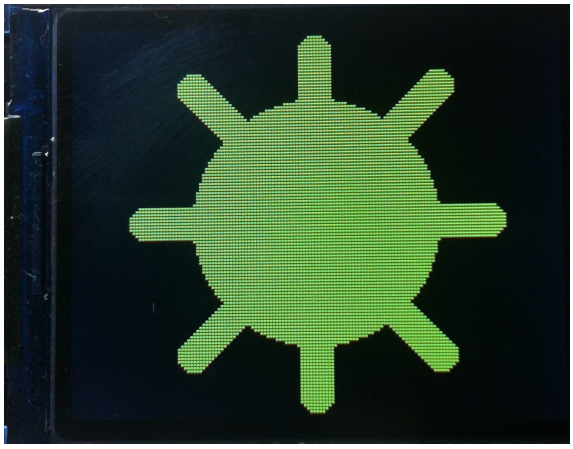


Display 3 is set up to receive text information from the internet subsystem in the form of character arrays. This information will be put up on the display as it comes in, and will be updated as much as possible. Because this is a larger screen, it allows for more information to be shown at once. At a minimum, the temperature, humidity, and the daily forecast will be shown. The top left and top right will show the temperature and humidity respectively. If the forecast is the only other information to be shown, it will be along the bottom of the screen. If there is additional information to be shown, it will scroll along the bottom of the screen, entering from the right and exiting to the left. The scrolling feature is achieved with reasonable speed by only updating ‘cells’ on the display where

changes occur. The character array is processed into small sections, which are sent to the display backed with black coloring so that they can overwrite current characters. This makes for smooth scrolling and readable text.

Display 4 is able to receive and display any extra information as needed from the internet subsystem. Based on the style of information the text can be large or small, scrolling or motionless, and static or changing. Some information that might be displayed includes sports scores, stock information, an rss feed, or a news ticker.

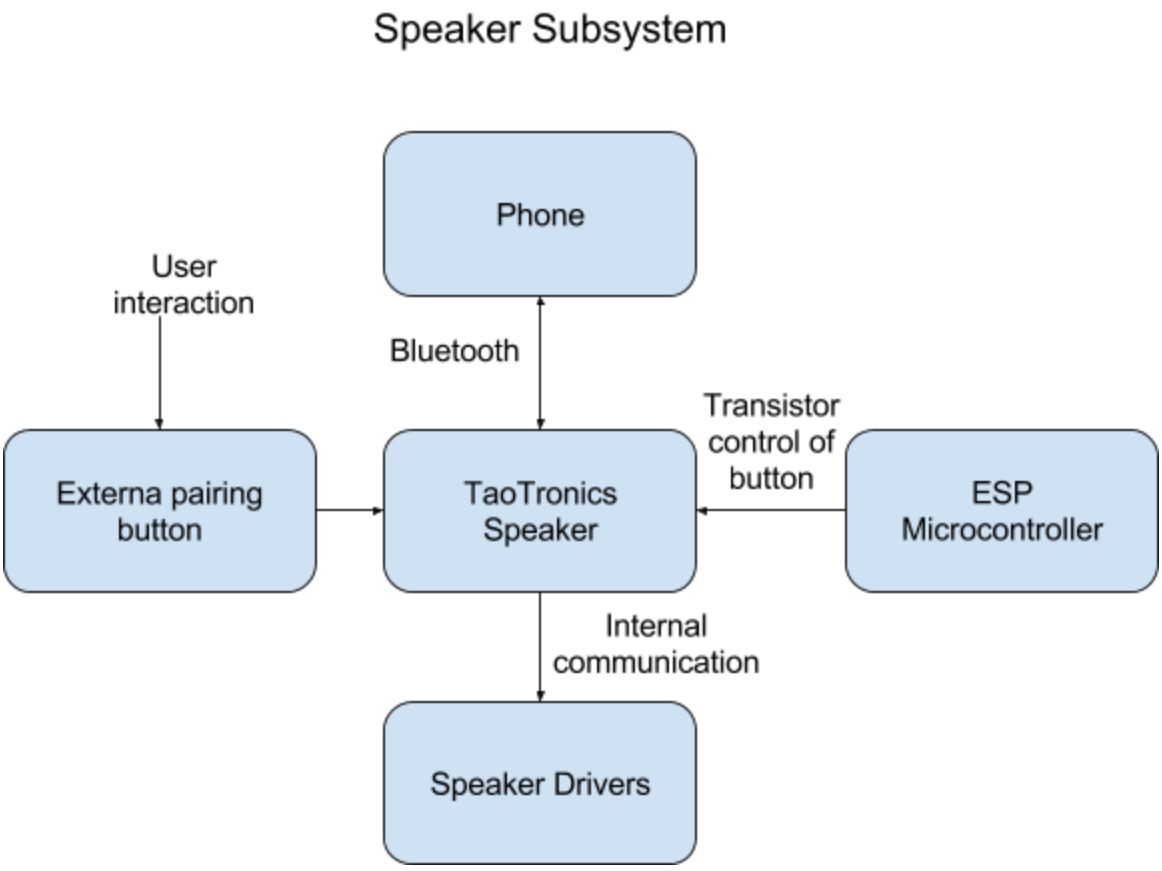
It is possible for all displays to show the text and images in color. It is unlikely that the user will want this functionality, as color does not show through the mirror as well, but it will nonetheless be available. Examples can be seen below, where the sun has been colored yellow and the stormcloud has been colored grey, blue, and yellow for the cloud, rain, and lightning respectively.



5.6 Speaker Subsystem

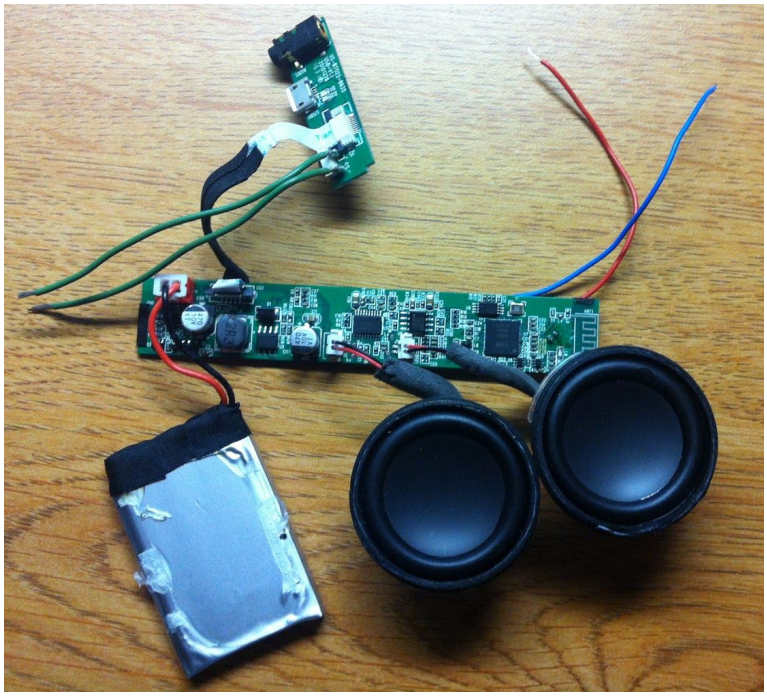
The speaker subsystem allows for the user to be able to play music through the mirror via a bluetooth connection. Once the mirror has connected to the bluetooth device (likely a phone), it will continuously receive data from the device and will output music accordingly.

To enable a simpler and cheaper mirror, a premade bluetooth speaker was used. Instead of running the bluetooth connection and control information through the microcontroller, the speaker handles the majority of it, leaving little for the microcontroller. The TaoTronics Wireless Pocket Boombox was chosen for its price and performance. The speaker has been disassembled to allow for modifications, and to ensure it would fit behind the mirror glass. A diagram of subsystem connections is included below.



There were two main modifications to the TaoTronics speaker. First, the speaker drivers themselves were removed in order to have more flexibility in mounting position. Testing was done to see if larger drivers would produce better sound, but it was determined the internal amplifier was not powerful enough. The second modification was adding hardware to make the system controllable by the microcontroller. To do this, wires were added across the contacts of the various surface mount buttons on the speaker. In this way, either the ESP microcontroller can control the features or the buttons can be run externally (allowing for better buttons and more straightforward construction). An image

of the disassembled device can be seen below (green wires are for power, red/blue are for pairing).



For control of the buttons, either the microcontroller can automatically connect the wires, or a button can be added externally for user access. When the microcontroller is used, it must be enabled for a precise amount of time to achieve the desired function, rather than reset the device. The following times were determined for successful operation: pairing button: 1-1.5 seconds, on button: 1.2-2 seconds, off button: 3-3.5 seconds.

One additional consideration that needs to be made for the speaker subsystem is whether a flexible membrane needs to be added to allow pressure in the system to equalize. Three such membranes were included in the speaker enclosure, but it is as of now unknown whether this will be necessary or not. If it is included, the existing membranes will simply be attached to the frame built for the mirror.

5.7 Interfaces

The User Interface subsystem will take user input to determine what the SSID and password for the WiFi, as well as other relevant data which will be saved to EEPROM. When this information is read out of EEPROM, it will need to be passed along to the

Internet subsystem in order to connect to the internet and pull weather and news data from the correct location. The User Interface subsystem and Internet subsystem interface need to be able to reliably exchange this information.

The Internet subsystem will need to interface with the Display subsystem in order to pass along the correct information to be displayed. The Internet subsystem needs to be capable of reading relevant data from a website and parsing it into strings which can be transmitted via SPI to the different displays mounted behind the mirror.

The On Off subsystem will interface with all of the Display subsystem as well as the Speaker subsystem. When the On Off subsystem times out, it will cause the displays and speaker to turn off and when the subsystem is triggered, it will signal for the displays to be turned back on and to display data, as well as signal for the Bluetooth speaker to turn on.

6. System Integration Testing

6.1 Subsystem Integration

To test the subsystem integration, the subsystems were combined by integrating two subsystems at a time. The User Interface and Internet subsystems were combined first and tested on the ESP8266 Dev Thing. A set of if statements was created to determined whether the ESP8266 would boot as an access point, or to connect to the internet. In order to test the compatibility of the subsystems, a variable was created to simulate input from a from a pin indicating which position it was in. If it was placed in an up state, the chip would boot into as an access point. From there, an SSID, password, and Zip code were input, and news source was selected from the dropdown menu. It was ensured that this data was being input to EEPROM by printing the strings to the monitor as the strings were written to nonvolatile memory. Next, the board was powered off and the position of the variable was changed to a down state to boot the system to connect to the internet and the board was powered back on. The data was read out of EEPROM, which was checked by printing the variables to the monitor. The chip then ran the Internet subsystem to ensure the variables read out of EEPROM provided the variables as the correct type. The weather data and news headlines were printed to the monitor to ensure that the user selections were what was found and parsed by the system.

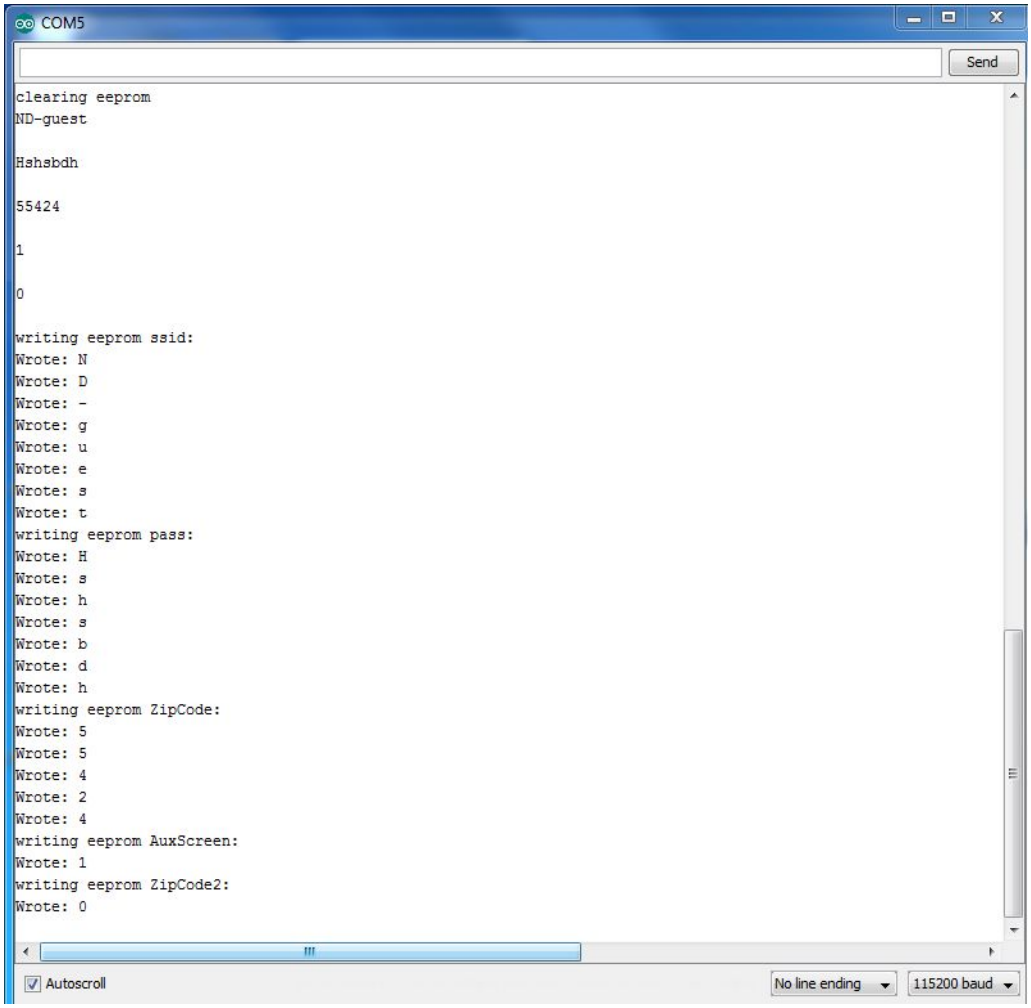


Figure: ESP8266 Booted as Access Point, Data Written to EEPROM

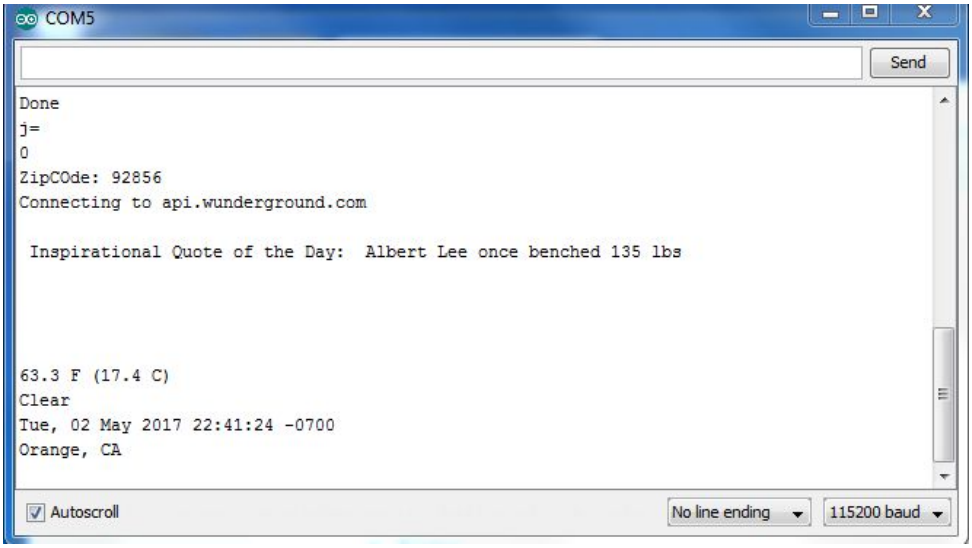


Figure: ESP8266 Booted to Connect to and Parse Data from Internet

After the user interface and internet subsystems were integrated, the next subsystem to integrate was the display subsystem. This required the main program file to utilize the multiplexer in order to display the information on each LCD display that the user selected in the User Interface. To test this, one display was added at a time, starting with one of the 1.8” displays. An inspirational quote was printed out on the screen, as well as the time, and the weather icons that were created in our code in order to see if the we could successfully communicate with the LCD. This process demonstrated that the 1.8” displays would work with the board hardware and software that was created. Next, a 2.8” display was added to the second display port. The same information that was sent to the 1.8” display was also sent to the 2.8” display. When both displays were run initially, there was problem with the multiplexer pins floating and causing the displays to prematurely clear or have crossfeed between the signals. To solve this problem, pull down resistors were added to the design. This stopped the floating of the multiplexer pins from adversely affecting the displays. Unfortunately, while the display was powered on and receiving information, the displays were showing inverted flashing and the images were inverted. This meant that the hardware was capable of supporting the multiple screens, but they were not able to be fully integrated.

Next, the On/Off subsystem was added to the system. To test this, the PIR sensor was connected to the board. This was set up to turn the entire system on when it detected motion and started a timer to keep the mirror on for 10 minutes when triggered before being reset and checked for motion. To test that the system was working, “On” was printed to the serial monitor when the PIR sensor was triggered. The sensor was initialized as high so that the system would go through its set up when it was initially turned on.

Finally, the Speaker subsystem was added onto the overall system. Since it was less expensive to dismantle a prebuilt Bluetooth speaker to connect to the system than to add a Bluetooth onto the chip, all that was required for integration was connecting the speaker power to the board and run out a button to sync the speaker with a device.

6.2 Testing Relevance to Overall System

When the systems were integrated, it was possible to show that the system met the design requirements that were set at the beginning of the project. The Internet and User Interface subsystems are able to be switched between, meaning that the user can easily switch the settings when they want and have the system connect to the internet and parse the data to display when the mirror is in normal operation. Although the 2.8” displays were not able to be fully integrated, the testing illustrated that the hardware had

the capability to support all the screens and run multiple screens at the same time as well as run screens of different sizes. The On/Off subsystem testing demonstrated that the power to the display was able to be controlled by movement of the user, allowing the mirror to turn itself on when a person is standing in front of it. By connecting the Bluetooth speaker to the board and running out a button to allow the speaker to sync, the system requirement of having a Bluetooth speaker built into the mirror.

7. User's Manual/Installation Manual

7.1 Installation

To install the mirror, please mount in your desired location. It is important to remember that there mirror must be positioned in a place that provides access to a power outlet, as for smart functionality the 5V power supply must be plugged in, otherwise the product will simply function as a mirror. Additionally, ensure that the wall you have selected to mount the mirror on is able to support the weight of the mirror, approximately 5 lbs. Be advised, the entire mirror is about 1" to 2" deep.

7.2 Mirror Setup

To set up the mirror WiFi and Location:

1. Plug the power supply into an outlet
2. Before powering the mirror on, ensure that the mode select switch is set to up position to enter "SETUP MODE"
3. Power on the mirror
4. On any WiFi enabled device there should now be an access point starting with "ESP". Select this access point
    a. If prompted for a password enter: "password"
5. Once connected use an internet browser to go to "192.168.4.1"
6. On this page you should see all available WiFi access points listed as well as the following text boxes:

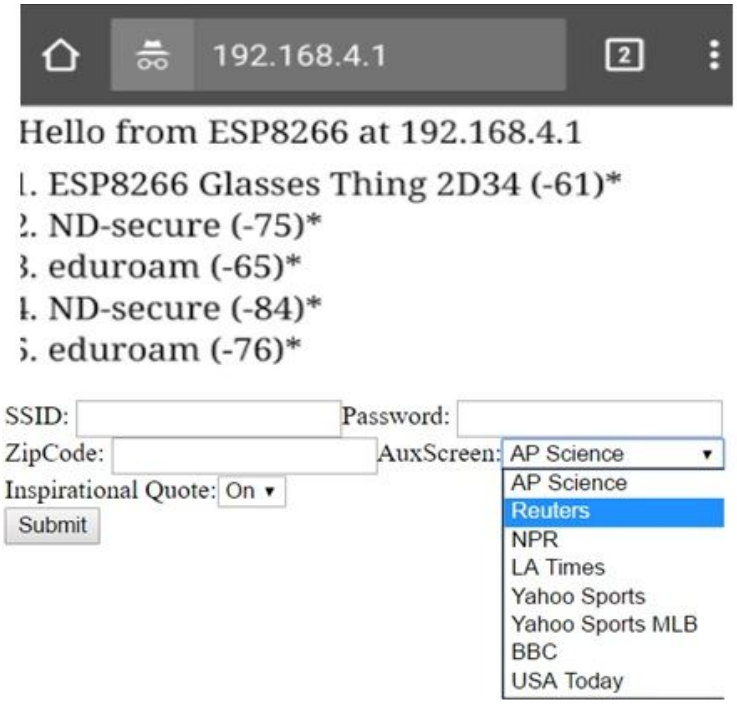


Figure: User Interface

7. Input your WiFi SSID, password, and Zip code and select a news source from which to receive headlines.
    a. All text boxes must be filled in to successfully submit
8. Once you have pressed the submit button you should see a message prompting you to power off the mirror. Power off the mirror.
9. Place the mode select switch in the down position to enter “DISPLAY MODE” and power on the mirror
    a. If the mirror was successfully setup, the time and weather for the Zip code that you input as well as the news headlines should appear on the mirror displays.

To set up speaker connectivity:

1. Power on the mirror
2. Press the sync button and look for the device to appear in Bluetooth devices
3. Select the device to connect
    a. The speaker should beep when connected

Congratulations, you have successfully setup your Smart Mirror. You can now begin to enjoy a more productive mirror experience.

7.3 How to Tell if Your Smart Mirror is Working

If your Smart Mirror is working properly, you should see the following:

1. If you are currently “SETUP MODE”:
    a. A hotspot starting with “ESP” should appear as a potential access point of WiFi enabled devices
    b. When connected to the hotspot starting “ESP”, you should be able to access the user interface by going to “192.168.4.1” and should be able to change the selected settings by inputting data and hitting submit
        i. All options should be selected to what is desired to submit a change
2. If you are currently in “DISPLAY MODE”:
    a. The screens should be lit up and displaying the time, weather, and news headlines
    b. The screens should automatically turn off 5 minutes (INSERT ACTUAL TIME) after you walk away from the mirror
        i. If the displays turn off while you are still at the mirror, move around to turn the trigger the motion sensor
    c. The Bluetooth speaker should be available to sync your Bluetooth enabled device with

7.4 Troubleshooting

If you are not currently seeing anything displayed on the mirror:

1. Make sure that the mirror is plugged in and powered on
2. Go to into “SETUP MODE” and re-input SSID, password, Zip code, and select the your news source
    a. Ensure that you have not made any spelling or capitalization errors in any of your inputs
    b. Make sure that you see a message prompting you to restart the mirror after submitting your information. If you do not see this, or see an error message instead, you have not successfully submitted your information
        i. All options must be input to successfully change information
    c. Some of the news sources will supply more headlines than others, so if you are unsatisfied with the number of news headlines, try using a different source
    d. Once you have submitted your information, turn off the mirror, enter “DISPLAY MODE”, and power the mirror on

If you cannot find the “ESP” access point:

1. Make sure that the mirror is plugged in and power on
2. Check that the mode select switch is in the up position

If you mirror is not displaying the correct news or weather source

1. Place the mirror into “SETUP MODE”
2. Re-enter the zip code and select the desired news source
    a. All options on the user interface must have data entered in order to submit settings change

If you cannot sync with the Bluetooth speaker:

1. Make sure that the mirror is plugged in and powered on
2. Make sure that Bluetooth is enabled on your device

General troubleshooting:

1. Make sure the mirror is plugged in
2. Reset the mirror by either powering it off and on, or pressing the reset button

8. To-Market Design Changes

While the prototype developed was able to accomplish all the requirements set forth for the project, the product would require upgrades to the design before it would be ready to be sold as a completed product. Additional time, as well as the knowledge gained from the development process, would allow the following improvements to be included in the Smart Mirror.

Firstly, a wooden from could be added to the mirror. This would allow for the mirror to become more aesthetically pleasing and create a more completed look. A well constructed frame would also assist in positioning the switches, keeping the hardware mounted behind the mirror, and provide a sturdy point by which to mount the mirror.

It would be beneficial to upgrade the PCB and code to work using the ESP32 instead of the ESP8266. Upgrading to the ESP32 would give the board more processing power and provide more I/O pins to improve the display functionality. This would also improve the memory capabilities of the board, allowing for a more robust code to be created and programmed to the board.

The user interface could be improved by creating a more aesthetically pleasing and polished design. Changing the name of the hotspot provided by the mirror to a constant name would allow easier access to the user. Similarly, changing the location to a more intuitive address name, rather than the default device IP of 192.168.4.1 would facilitate a more straightforward user experience. Combined with improvements to the Internet subsystem, additional options for what to display on the LCD screens could be developed, such as syncing email or calendars to the mirror.

Finally, additional sizes for the mirror could be offered. This could allow for more flexibility on the amount of information that could be shown on the mirror at any given time. Also, a larger mirror size may allow for more screens to be placed behind the mirror, or larger displays to be installed, which could reduce the frequency with which the displays need to be refreshed.

9. Conclusions

The Smart Mirror designed in this project will provide the user with an enhanced mirror experience. By making use of multiple displays, the user can stay updated on the time, weather, and news headlines while preparing for the day in with the fully functional Smart Mirror. Although there are other smart mirror technologies that are available, the Smart Mirror created in this project stresses saving cost and flexible usage. Through an easy to use interface, the mirror can be easily setup to display data that conforms with their desires. The mirror is able to connect to the internet and parse the proper data to display. The PIR sensor ensures that the mirror will always turn on when a person steps up to use it. Building in a Bluetooth speaker means that the user is also able to play their music directly from the mirror. While the Smart Mirror will need to be more polished and have a few changes made before it can be a viable product to be sold, but the Smart Mirror made in this project meets all the design goals set forth before the project and has all the elements that would be needed for a fully functional Smart Mirror product.

10. Appendices

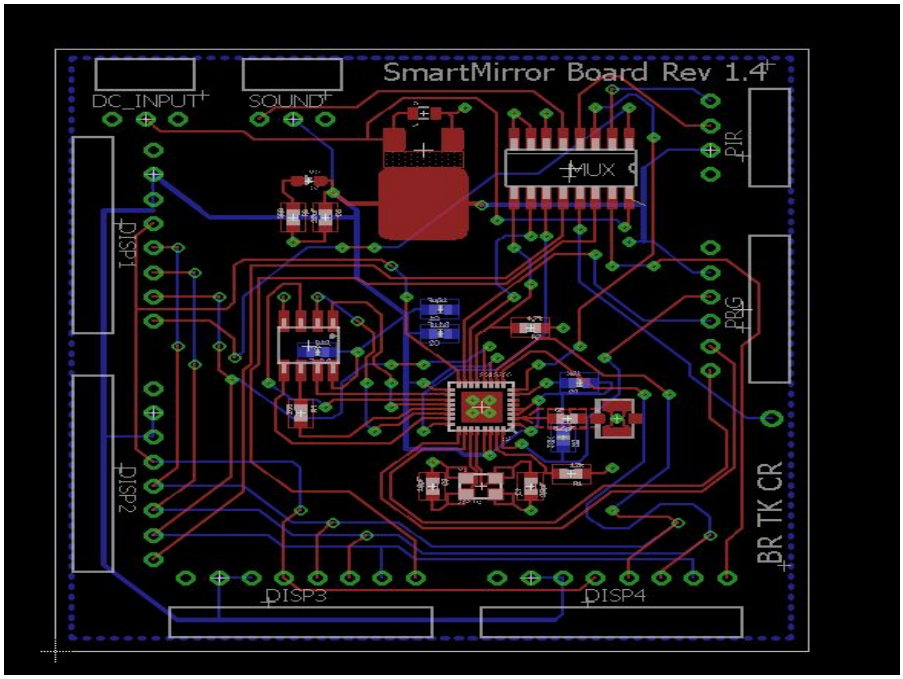
Appendix A:
Relevant Spec Sheets

https://nurdspace.nl/images/e/e0/ESP8266_Specifications_English.pdf

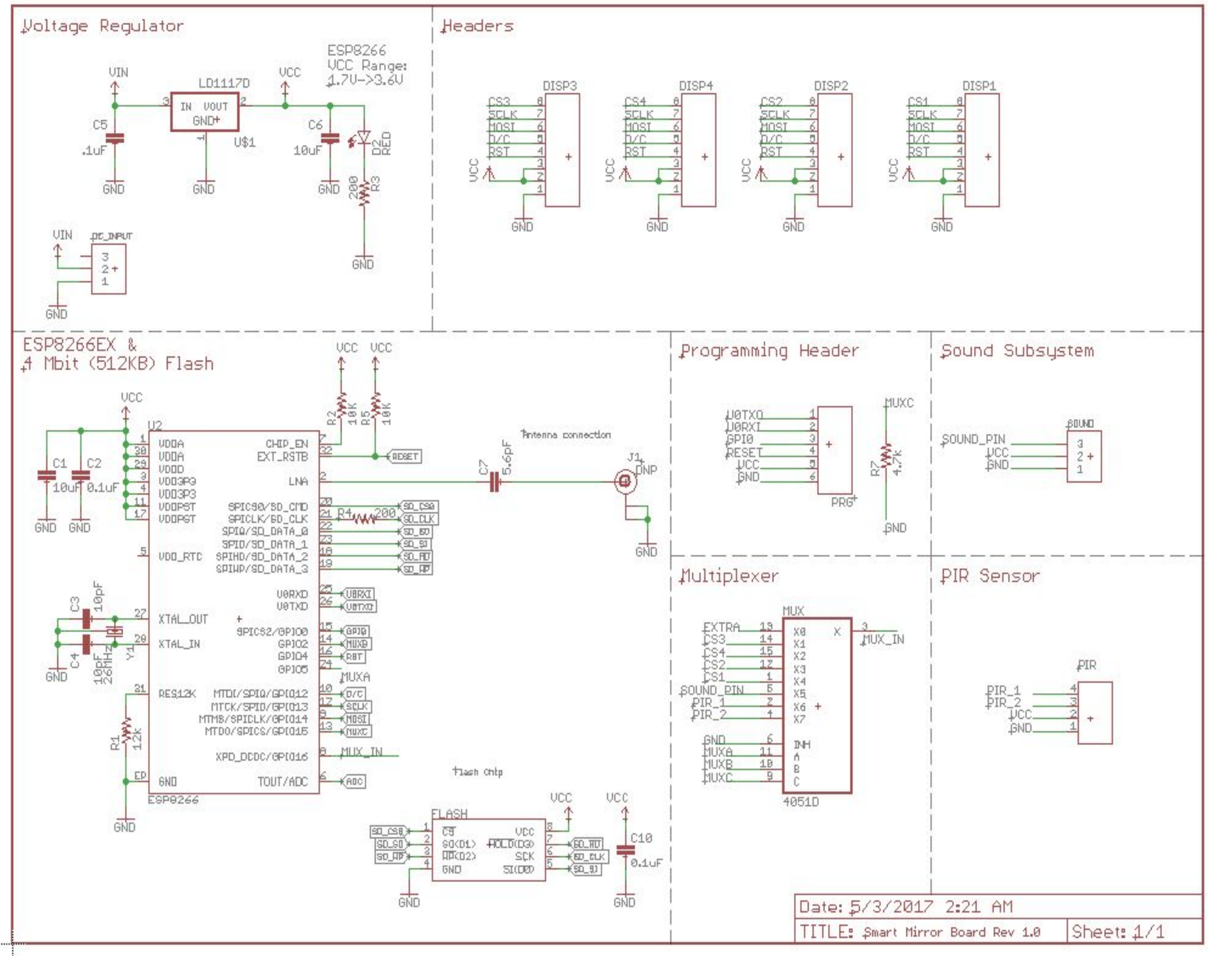
https://cdn-learn.adafruit.com/downloads/pdf/adafruit-2-8-tft-touch-shield-v2.pdf

https://sakailogin.nd.edu/access/content/group/SP17-EE-41440-01/Spec%20Sheets/PIR%20sensor.pdf

Appendix B:
Final Board Eagle File



Appendix C: Final Board Schematic



Appendix D:

<u>PIR On/Off Testing Code</u>

```
/*
 * PIR sensor tester
 */

int ledPin = 0;                // choose the pin for the LED
int inputPin = 13;
//-+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++            // choose the input pin (for PIR
sensor)
int pirState = LOW;             // we start, assuming no motion detected
int val = 0;                    // variable for reading the pin status
int count=0;
int i;
void setup() {
  pinMode(ledPin, OUTPUT);      // declare LED as output
```

```
  pinMode(inputPin, INPUT);      // declare sensor as input

  Serial.begin(9600);
}
 /*
void loop(){
  val = digitalRead(inputPin);  // read input value
  if (val == HIGH) {            // check if the input is HIGH
    digitalWrite(ledPin, HIGH); //turn LED ON
    delay(100);// turn LED ON
    if (pirState == LOW) {
      // we have just turned on
      Serial.println("Motion detected!");
      // We only want to print on the output change, not state
      pirState = HIGH;
    }
  } else {
    digitalWrite(ledPin, LOW); // turn LED OFF
      if (pirState == HIGH){
      // we have just turned of
      Serial.println("Motion ended!");
      // We only want to print on the output change, not state
      pirState = LOW;
    }
  }
}*/

void loop(){
val=digitalRead(inputPin);
while (i==0){
if (count < 5500){
  val=digitalRead(inputPin);
      if (val == HIGH){
        digitalWrite(ledPin, HIGH);
        count =0;
        Serial.println(count);
    } else if (val == LOW){
     digitalWrite(ledPin, HIGH);
     delay(1);
     count++;
     Serial.println(count);
    }
}
```

```
 else {
  digitalWrite(ledPin, LOW);
  } //Turn LED Off
}
}
```

Appendix E:

Time/Weather Parser Testing Code:

```
#include <ESP8266WiFi.h>

#include <ArduinoJson.h>
#include "RSSReader.h"
const char SSID[]     = "ND-guest";
const char PASSWORD[] = "";
String WU_API_KEY ="2df07ee2d9774288";
String WU_LOCATION = "46617"; //read from EEPROM
//String GET = "/api/" + WU_API_KEY + "/conditions/q/" + "WU_LOCATION" + ".json";



// 5 minutes between update checks. The free developer account has a limit
// on the  number of calls so don't go wild.
#define DELAY_NORMAL    (5*60*1000)
// 20 minute delay between updates after an error
#define DELAY_ERROR     (20*60*1000)

#define WUNDERGROUND "api.wunderground.com"
#define NUMBER_OF_FEEDS (sizeof(rssFeedURLs) / sizeof(char *))
int rssFeedIndex;
int QuoteIndex;

// Create RSS reader instance with 3 second timeout
RSSReader reader = RSSReader(300);

// Array of feed URLs
const char *rssFeedURLs [] = {
  "https://sports.yahoo.com/mlb/rss.xml", // A Few
  "https://sports.yahoo.com/top/rss.xml",
  "www.npr.org/rss/rss.php?id=1001", // 2 headlines
  "http://feeds.bbci.co.uk/news/rss.xml", //Not working
  "http://hosted.ap.org/lineups/SCIENCEHEADS-rss_2.0.xml?SITE=OHLIM&SECTION=HOME",
// A few
  "http://www.latimes.com/rss2.0.xml", // works
```

```
  "http://feeds.reuters.com/reuters/topNews",
  "rssfeeds.usatoday.com/usatoday-NewsTopStories",//doesn't work
};

//Necessary for RSSReader
void setTitleCallback(pt2Function titleCallback);
void setDescCallback(pt2Function descCallback);
void setPubDateCallback(pt2Function dateCallback);


void setup() {
  // Initialize serial interface and allow it to stablize
  Serial.begin(115200);
  delay(1000);
  Serial.print(F("Connecting to "));
  Serial.println(SSID);

  // Attempt WiFi network connection
  WiFi.begin(SSID, PASSWORD);
  //serial.println(GET)

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(F("."));
  }
  Serial.println();
  Serial.println(F("WiFi connected"));
  Serial.println(F("IP address: "));
  Serial.println(WiFi.localIP());


  // Setup callbacks for title and pubDate tags in RSS XML
  reader.setTitleCallback(&titleCallback);
  reader.setPubDateCallback(&pubDateCallback);

  // Initialize feed index to first entry in feed list
  rssFeedIndex = 0; //Read from EEPROM
  /////////
  QuoteIndex= 0; //Read from EEPROM
  //Initialize Zip Code
  // HTTP request
}
```

```
static char respBuf[4096];

void loop() {
  //const int httpPort = 80;
  // Short delay to keep networking happy
  int j=0;
  for (j =0; j<1 ; j++){
  delay(10);

  // Get URL of RSS feed to display
  const char *url = rssFeedURLs[rssFeedIndex];
  Serial.println(url);
  // Read and parse the RSS feed
  bool result = reader.read(url);
  Serial.println("j= ");
  Serial.println(j);
  delay(3000);
  }
  String aa= "api.wunderground.com";
  String a= "GET /api/"+WU_API_KEY+"/conditions/q/"+WU_LOCATION+".json HTTP/1.1\r\n";
  String b= "User-Agent: ESP8266/0.1\r\n";
  String c= "Accept: */*\r\n";
  String d= "Host: "+aa+"\r\n";
  String e= "Connection: close\r\n";
  String f= "\r\n";
  String g= a+b+c+d+e+f;


  WiFiClient httpclient;
  const int httpPort = 80;
  if (!httpclient.connect(WUNDERGROUND, httpPort)) {
    //Serial.println(F("connection failed"));
    delay(DELAY_ERROR);
    return;
  }
 // This will send the http request to the server
  httpclient.print(g);
  httpclient.flush();
  Serial.print(F("Connecting to "));
  Serial.println(WUNDERGROUND);

//Inspirational Quotes part
const char *quote= InspirationalQuotes[QuoteIndex];
```

```
Serial.print("\n Inspirational Quote of the Day:  ");
Serial.println(quote);
Serial.print("\n \n \n \n");

//Collect http response headers and content from WEather Underground
// HTTP headers are discarded
// The content is formatted in JSON and is left in respBuf
  int respLen= 0;
  bool skip_headers = true;
  while (httpclient.connected() || httpclient.available()) {
    if (skip_headers) {
      String aLine = httpclient.readStringUntil('\n');
      //Serial.println(aLine);
      //Serial.println("^aLine");
      // Blank line denotes end of headers
      if (aLine.length() <= 1) {
        skip_headers = false;
      }
    }
    else {
      int bytesIn;
      bytesIn = httpclient.read((uint8_t *)&respBuf[respLen], sizeof(respBuf) - respLen);
      //Serial.print(F("bytesIn ")); Serial.println(bytesIn);
      if (bytesIn > 0) {
        respLen += bytesIn;
        if (respLen > sizeof(respBuf)) respLen = sizeof(respBuf);
      }
      else if (bytesIn < 0) {
        //Serial.print(F("read error "));
       // Serial.println(bytesIn);
      }
    }
    delay(1);
  }
  httpclient.stop();

  if (respLen >= sizeof(respBuf)) {
     Serial.print(F("respBuf overflow "));
    Serial.println(respLen);
    delay(DELAY_ERROR);
    return;
  }
  // Terminate the C string
```

```
  respBuf[respLen++] = '\0';
  //Serial.print(F("respLen "));
  //Serial.println(respLen);
  //Serial.println(respBuf);
  if (showWeather(respBuf)) {
    delay(DELAY_NORMAL);
  }
  else {
    delay(DELAY_ERROR);
  }

}


void titleCallback(char *titleStr) {

  char buffer[TITLE_BUFFER_SIZE];

  Serial.print("\nTitle: ");
  Serial.println(titleStr);
  // Make buffer empty
  buffer[0] = '\0';
  strcpy(buffer, " ");
  strcat(buffer, titleStr);
  strcat(buffer, "    ");
}


bool showWeather(char *json)
{
  StaticJsonBuffer<3 * 1024> jsonBuffer;

  // Skip characters until first '{' found
  // Ignore chunked length, if present
  char *jsonstart = strchr(json, '{');
  //Serial.print(F("jsonstart ")); Serial.println(jsonstart);
  if (jsonstart == NULL) {
    //Serial.println(F("JSON data missing"));
    return false;
  }
  json = jsonstart;

  // Parse JSON
```

```
  JsonObject& root = jsonBuffer.parseObject(json);
  if (!root.success()) {
    //Serial.println(F("jsonBuffer.parseObject() failed"));
    return false;
  }

  // Extract weather info from parsed JSON
  JsonObject& current = root["current_observation"];
  const char *temp_string = current["temperature_string"];
  Serial.println(temp_string);
  const char *weather = current["weather"];
  Serial.println(weather);
  const char *observation_time = current["local_time_rfc822"];
  Serial.println(observation_time);
  const char *place = current["display_location"]["full"];
  Serial.println(place);

  return true;
}

// Callback called every time a pubDate tag is found in the RSS XML
void pubDateCallback(char *dateStr) {
  Serial.println(dateStr);
}
```

Appendix F:

News Parser Testing Code and Header Files:

```
#include <ESP8266WiFi.h>
#include "RSSReader.h"
#define NUMBER_OF_FEEDS (sizeof(rssFeedURLs) / sizeof(char *))
int rssFeedIndex;


// Create RSS reader instance with 3 second timeout
RSSReader reader = RSSReader(300000000);

const char *WIFI_SSID = "ND-guest";
const char *WIFI_PASSWORD= "";

// Array of feed URLs
const char *rssFeedURLs [] = {
 "www.npr.org/rss/rss.php?id=1001",
```

```
 "http://feeds.bbci.co.uk/news/rss.xml",
 "http://hosted.ap.org/lineups/SCIENCEHEADS-rss_2.0.xml?SITE=OHLIM&SECTION=HOME",
 "http://www.latimes.com/rss2.0.xml",
 "http://feeds.reuters.com/reuters/topNews",
 "rssfeeds.usatoday.com/usatoday-NewsTopStories",
};

//Necessary for RSSReader
void setTitleCallback(pt2Function titleCallback);
void setDescCallback(pt2Function descCallback);
void setPubDateCallback(pt2Function dateCallback);
void titleCallback(char *titleStr) {

  char buffer[TITLE_BUFFER_SIZE];

  Serial.print("\nTitle: ");
  Serial.println(titleStr);
  // Make buffer empty
  buffer[0] = '\0';
  strcpy(buffer, " ");
  strcat(buffer, titleStr);
  strcat(buffer, "    ");
}
void setup() {
  // Initialize serial interface and allow it to stablize
  Serial.begin(115200);
  delay(1000);
  Serial.print("\n\nConnecting to: ");
  Serial.println(WIFI_SSID);

  // Attempt WiFi network connection
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Setup callbacks for title and pubDate tags in RSS XML
  reader.setTitleCallback(&titleCallback);
  reader.setPubDateCallback(&pubDateCallback);

  // Initialize feed index to first entry in feed list
```

```
  rssFeedIndex = 5;
}

// Callback called every time a pubDate tag is found in the RSS XML
void pubDateCallback(char *dateStr) {
  Serial.println(dateStr);
}
void loop() {

  // Short delay to keep networking happy
  delay(1);

  // Get URL of RSS feed to display
  const char *url = rssFeedURLs[rssFeedIndex];
  // Read and parse the RSS feed
  bool result = reader.read(url);
}
```

RSSReader.cpp

```
/*
 *  RSSReader.cpp - RSSReader Class Functions
 *
 *  Concept, Design and Implementation by: Craig A. Lindley
 *  Last Update: 11/06/2015
 */

#include <ESP8266WiFi.h>
#include "RSSReader.h"

#define HTTPPORT 80

// Connection for DST pushbutton
#define DST_SW 5

// States of the Finite State Machine (FSM) used to parse tags out of RSS XML
enum STATES {INIT, CHK_START, TEST_FIRST_CHAR, CHK_DATE, CHK_TITLE,
CHK_DESC, GET_DATE, GET_TITLE, GET_DESC};
enum STATES state;

const char *date = "pubDate>";
const char *title = "title>";
const char *desc  = "description>";
```

```
// Finite state machine for parsing XML tags of interest
void RSSReader::runStateMachine(char ch) {

  switch (state) {
    case INIT:
      // Make buffers empty
      memset(titleBuffer, 0, sizeof(titleBuffer));
      memset(descBuffer, 0, sizeof(descBuffer));
      memset(dateBuffer, 0, sizeof(dateBuffer));

      // Next state
      state = CHK_START;
      break;

    case CHK_START:
      if (ch == '<') {
        state = TEST_FIRST_CHAR;
      }
      break;

    case TEST_FIRST_CHAR:
      if (ch == date[0]) {
        dateIndex = 1;
        state = CHK_DATE;
      }
      else if (ch == title[0]) {
        titleIndex = 1;
        state = CHK_TITLE;
      } else if (ch == desc[0]) {
        descIndex = 1;
        state = CHK_DESC;
      } else  {
        state = CHK_START;
      }
      break;

    case CHK_DATE:
      if (ch != date[dateIndex]) {
        state = CHK_START;
        break;
      }
      dateIndex++;
```

```
  if (ch == '>') {
    dateIndex = 0;
    state = GET_DATE;
  }
  break;

case CHK_TITLE:
  if (ch != title[titleIndex]) {
    state = CHK_START;
    break;
  }
  titleIndex++;
  if (ch == '>') {
    titleIndex = 0;
    state = GET_TITLE;
  }
  break;

case CHK_DESC:
  if (ch != desc[descIndex]) {
    state = CHK_START;
    break;
  }
  descIndex++;
  if (ch == '>') {
    descIndex = 0;
    state = GET_DESC;
  }
  break;

case GET_TITLE:
  if (ch != '<') {
    titleBuffer[titleIndex++] = ch;
  } else  {
    titleBuffer[titleIndex++] = '\0';
    if ((strlen(titleBuffer) != 0) && (titleCallback != NULL)) {
      titleCallback(titleBuffer);
    }
    state = CHK_START;
  }
  break;

case GET_DESC:
```

```
      if ((ch != '<') && (ch != '&')) {
        descBuffer[descIndex++] = ch;
      } else  {
        descBuffer[descIndex++] = '\0';
        if ((strlen(descBuffer) != 0) && (descCallback != NULL)) {
          descCallback(descBuffer);
        }
        state = CHK_START;
      }
      break;

    case GET_DATE:
      if (ch != '<') {
        dateBuffer[dateIndex++] = ch;
      } else  {
        // Just return the date not the time
        dateBuffer[17] = '\0';
        if ((strlen(dateBuffer) != 0) && (dateCallback != NULL)) {
          dateCallback(dateBuffer);
        }
        state = CHK_START;
      }
      break;
  }
}

// Class Constructor
RSSReader::RSSReader(int _timeoutInMS) {
  // Save incoming timeout value
  timeoutInMS = _timeoutInMS;

  // Setup DST pushbutton switch
  pinMode(DST_SW, INPUT_PULLUP);
}

// Assign the RSSReader's pubDate callback function
void RSSReader::setPubDateCallback(pt2Function _dateCallback) {
  dateCallback = _dateCallback;
}

// Assign the RSSReader's title callback function
void RSSReader::setTitleCallback(pt2Function _titleCallback) {
  titleCallback = _titleCallback;
```

```
}

// Assign the RSSReader's description callback function
void RSSReader::setDescCallback(pt2Function _descCallback) {
  descCallback = _descCallback;
}

// Parse URL and return result
// If successful, host and path storage areas have
// components of the URL.
bool RSSReader::parseURL(const char* url) {

  int index;
  const char *ptr;

  // Clear string storage
  memset(host, 0, sizeof(host));
  memset(path, 0, sizeof(path));

  // Is there a protocol specified ?
  // If so skip it
  if ((ptr = strchr(url, ':')) != NULL)  {
    // Yes there was a protocol
    ptr += 3;
  } else  {
    // No protocol specified
    ptr = url;
  }

  // Search for host separator
  char *endPtr = strchr(ptr, '/');
  if (endPtr == NULL) {
    return false;
  }
  // Copy host string to host storage
  index = 0;
  while (ptr != endPtr) {
    host[index++] = *ptr++;
  }
  // The remainder of the string is the path
  strcpy(path, endPtr);

  // All went well
```

```
  return true;
}

// Make the connection to the RSS source and parse the returned
// results calling the callback function when a title or description
// element is found in the XML.
// Returns true if pushbutton switch was pressed to terminate feed; false otherwise.
bool RSSReader::read(const char *url) {

  WiFiClient client;

  // Parse the URL passed in
  if (! parseURL(url)) {
    return false;
  }

  // Initial state machine state
  state = INIT;

  // Use WiFiClient class to create TCP connections
  Serial.println("Connecting to host");
  while (! client.connect(host, HTTPPORT)) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Host connection succeeded");

  // This will send the request to the server
  client.print(String("GET ") + path + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: keep-alive" + "\r\n\r\n");

  // Read all the chars run them through state machine
  unsigned long timeoutMS = timeoutInMS + millis();
  while (timeoutMS > millis()) {
    int count = client.available();
    while (count--) {
      char ch = client.read();
      //      Serial.print(ch);
      runStateMachine(ch);
      timeoutMS = timeoutInMS + millis();

      // Check pushbutton
```

```
      if (digitalRead(DST_SW) == LOW) {
        // Switch was active; Terminate read
        // Wait till button was released
        while (digitalRead(DST_SW) == LOW) {
          delay(10);
        }
        return true;
      }
      yield();
    }
    delay(250);
  }
  client.stop();

  Serial.println("\n\nDone");
  return false;
}
```

RSSReader.h

```
#ifndef RSSREADER_H
#define RSSREADER_H

#define TITLE_BUFFER_SIZE  256
#define DESC_BUFFER_SIZE   512
#define PUBDATE_BUFFER_SIZE 80

// A function pointer for callback
typedef void (*pt2Function)(char *);

class RSSReader {

  public:
    RSSReader(int _timeoutInMS);

    void setTitleCallback(pt2Function _titleCallback);
    void setDescCallback(pt2Function _descCallback);
    void setPubDateCallback(pt2Function _dateCallback);

    bool read(const char *url);

  private:
    int timeoutInMS;
```

```
    int titleIndex;
    int descIndex;
    int dateIndex;

    pt2Function titleCallback;
    pt2Function descCallback;
    pt2Function dateCallback;

    bool parseURL(const char* url);

    char titleBuffer[TITLE_BUFFER_SIZE];
    char descBuffer[DESC_BUFFER_SIZE];
    char dateBuffer[PUBDATE_BUFFER_SIZE];

    char host[30];
    char path[100];


    void runStateMachine(char ch);

};

#endif
```

Appendix G:
User Interface Testing Code:

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <EEPROM.h>

ESP8266WebServer server(80);

const char* ssid = "test";
const char* passphrase = "test";
const char* zipcode = "test";
const char* zipcode2 = "test";
const char* auxscreen = "1";
String st;
String content;
int statusCode;
```

```
void setup() {
  Serial.begin(115200);
  EEPROM.begin(512);
  delay(10);
  Serial.println();
  Serial.println();
  Serial.println("Startup");
  // read eeprom for ssid and pass
  //Serial.println("Reading EEPROM ssid");
  String esid;
  for (int i = 0; i < 32; ++i)
    {
      esid += char(EEPROM.read(i));
    }
  Serial.print("SSID: ");
  Serial.println(esid);
  //Serial.println("Reading EEPROM pass");
  String epass = "";
  for (int i = 32; i < 96; ++i)
    {
      epass += char(EEPROM.read(i));
    }
  Serial.print("PASS: ");
  Serial.println(epass);
  String ezip = "";
  for (int i = 96; i < 101; ++i)
    {
      ezip += char(EEPROM.read(i));
    }
  Serial.print("ZipCOde: ");
  Serial.println(ezip);
   String eaux = "";
  for (int i = 107; i < 110; ++i)
    {
      eaux += char(EEPROM.read(i));
    }
  Serial.print("AuxScreen: ");
  Serial.println(eaux);
  int AuxScreenSelect = eaux.toInt();
   String ezip2 = "";
  for (int i = 110; i < 115; ++i)
    {
      ezip2 += char(EEPROM.read(i));
```

```
    }
  Serial.print("ZipCOde2: ");
  Serial.println(ezip2);
  setupAP();
}

/*bool testWifi(void) {
  Serial.println("Connect timed out, opening AP");
  return false;
} */

void launchWeb(int webtype) {
  Serial.println("");
  Serial.print("SoftAP IP: ");
  Serial.println(WiFi.softAPIP());
  createWebServer(webtype);
  // Start the server
  server.begin();
  Serial.println("Server started");
}

void setupAP(void) {
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);
  int n = WiFi.scanNetworks();
  Serial.println("scan done");
  if (n == 0)
    Serial.println("no networks found");
  else
  {
    Serial.print(n);
    Serial.println(" networks found");
    for (int i = 0; i < n; ++i)
     {
      // Print SSID and RSSI for each network found
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.print(")");
      Serial.println((WiFi.encryptionType(i) == ENC_TYPE_NONE)?" ":"*");
```

```
      delay(10);
     }
  }
  Serial.println("");
  st = "<ol>";
  for (int i = 0; i < n; ++i)
    {
      // Print SSID and RSSI for each network found
      st += "<li>";
      st += WiFi.SSID(i);
      st += " (";
      st += WiFi.RSSI(i);
      st += ")";
      st += (WiFi.encryptionType(i) == ENC_TYPE_NONE)?" ":"*";
      st += "</li>";
    }
  st += "</ol>";
  delay(100);
  WiFi.softAP(ssid, passphrase, 6);
  Serial.println("softap");
  launchWeb(1);
  Serial.println("over");
}

void createWebServer(int webtype)
{
  if ( webtype == 1 ) {
    server.on("/", []() {
        IPAddress ip = WiFi.softAPIP();
        String ipStr = String(ip[0]) + '.' + String(ip[1]) + '.' + String(ip[2]) + '.' + String(ip[3]);
        content = "<!DOCTYPE HTML>\r\n<html>Hello from ESP8266 at ";
        content += ipStr;
        content += "<p>";
        content += st;
        content += "</p><form method='get' action='setting'><label>SSID: </label><input
name='ssid' length=32><label>Password: </label><input name='pass'
length=64><br><label>ZipCode: </label><input name='ZipCode'
length=10><label>AuxScreen:</label><select name='AuxScreen'><option value='1'>AP
Science</option><option value='2'>Reuters</option><option value='3'>NPR</option><option
value='4'>Yahoo Sports</option><option value='7'>Yahoo Sports MLB</option><option
value='5'>Different City Weather</option><option value='6'>Inspirational
Quotes</option></select><br><label>2nd ZipCode: </label><input name='ZipCode2'
length=10><br><input type='submit'></form>";
```

```
    content += "</html>";
    server.send(200, "text/html", content);
  });
  server.on("/setting", []() {
    String qsid = server.arg("ssid");
    String qpass = server.arg("pass");
    String qZip = server.arg("ZipCode");
    String qAux = server.arg("AuxScreen");
    String qZip2 = server.arg("ZipCode2");
    if (qsid.length() > 0 && qpass.length() > 0 && qZip.length() > 0  && qAux.length() > 0  &&
qZip2.length() > 0) {
      Serial.println("clearing eeprom");
      for (int i = 0; i < 128; ++i) { EEPROM.write(i, 0); }
      Serial.println(qsid);
      Serial.println("");
      Serial.println(qpass);
      Serial.println("");
      Serial.println(qZip);
      Serial.println("");
      Serial.println(qAux);
      Serial.println("");
      Serial.println(qZip2);
      Serial.println("");

      Serial.println("writing eeprom ssid:");
      for (int i = 0; i < qsid.length(); ++i)
        {
          EEPROM.write(i, qsid[i]);
          Serial.print("Wrote: ");
          Serial.println(qsid[i]);
        }
      Serial.println("writing eeprom pass:");
      for (int i = 0; i < qpass.length(); ++i)
        {
          EEPROM.write(32+i, qpass[i]);
          Serial.print("Wrote: ");
          Serial.println(qpass[i]);
        }
        Serial.println("writing eeprom ZipCode:");
      for (int i = 0; i < qZip.length(); ++i)
        {
          EEPROM.write(96+i, qZip[i]);
          Serial.print("Wrote: ");
```

```
            Serial.println(qZip[i]);
          }
          Serial.println("writing eeprom AuxScreen:");
          for (int i = 0; i < qAux.length(); ++i)
          {
            EEPROM.write(107+i, qAux[i]);
            Serial.print("Wrote: ");
            Serial.println(qAux[i]);
          }
          Serial.println("writing eeprom ZipCode2:");
          for (int i = 0; i < qZip2.length(); ++i)
          {
            EEPROM.write(110+i, qZip2[i]);
            Serial.print("Wrote: ");
            Serial.println(qZip2[i]);
          }
         EEPROM.commit();
         content = "{\"Success\":\"saved to eeprom... reset to boot into new wifi\"}";
         statusCode = 200;
        } else {
         content = "{\"Error\":\"404 not found\"}";
         statusCode = 404;
         Serial.println("Sending 404");
        }
        server.send(statusCode, "application/json", content);
    });
  } else if (webtype == 0) {
    server.on("/", []() {
      IPAddress ip = WiFi.localIP();
      String ipStr = String(ip[0]) + '.' + String(ip[1]) + '.' + String(ip[2]) + '.' + String(ip[3]);
      server.send(200, "application/json", "{\"IP\":\"" + ipStr + "\"}");
    });
    server.on("/cleareeprom", []() {
      content = "<!DOCTYPE HTML>\r\n<html>";
      content += "<p>Clearing the EEPROM</p></html>";
      server.send(200, "text/html", content);
      Serial.println("clearing eeprom");
      for (int i = 0; i < 128; ++i) { EEPROM.write(i, 0); }
      EEPROM.commit();
    });
  }
}
```

```
void loop() {
  server.handleClient();
}
```

Appendix H:

LCD Testing Code and Header Files

1. Adafruit_GFX.h

```
#ifndef _ADAFRUIT_GFX_H
#define _ADAFRUIT_GFX_H

#if ARDUINO >= 100
#include "Arduino.h"
#include "Print.h"
#else
#include "WProgram.h"
#endif
#include "gfxfont.h"

class Adafruit_GFX : public Print {

        public:

        Adafruit_GFX(int16_t w, int16_t h); // Constructor

        // This MUST be defined by the subclass:
        virtual void drawPixel(int16_t x, int16_t y, uint16_t color) = 0;

        // TRANSACTION API / CORE DRAW API
        // These MAY be overridden by the subclass to provide device-specific
        // optimized code.  Otherwise 'generic' versions are used.
        virtual void startWrite(void);
        virtual void writePixel(int16_t x, int16_t y, uint16_t color);
        virtual void writeFillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);
        virtual void writeFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
        virtual void writeFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
        virtual void writeLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint16_t color);
        virtual void endWrite(void);

        // CONTROL API
        // These MAY be overridden by the subclass to provide device-specific
        // optimized code.  Otherwise 'generic' versions are used.
```

```
virtual void setRotation(uint8_t r);
virtual void invertDisplay(boolean i);

// BASIC DRAW API
// These MAY be overridden by the subclass to provide device-specific
// optimized code.  Otherwise 'generic' versions are used.
virtual void
// It's good to implement those, even if using transaction API
drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color),
drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color),
fillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color),
fillScreen(uint16_t color),
// Optional and probably not necessary to change
drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint16_t color),
drawRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);

// These exist only with Adafruit_GFX (no subclass overrides)
void
drawCircle(int16_t x0, int16_t y0, int16_t r, uint16_t color),
drawCircleHelper(int16_t x0, int16_t y0, int16_t r, uint8_t cornername,
        uint16_t color),
fillCircle(int16_t x0, int16_t y0, int16_t r, uint16_t color),
fillCircleHelper(int16_t x0, int16_t y0, int16_t r, uint8_t cornername,
        int16_t delta, uint16_t color),
drawTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
        int16_t x2, int16_t y2, uint16_t color),
fillTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
        int16_t x2, int16_t y2, uint16_t color),
drawRoundRect(int16_t x0, int16_t y0, int16_t w, int16_t h,
        int16_t radius, uint16_t color),
fillRoundRect(int16_t x0, int16_t y0, int16_t w, int16_t h,
        int16_t radius, uint16_t color),
drawBitmap(int16_t x, int16_t y, const uint8_t *bitmap,
int16_t w, int16_t h, uint16_t color),
drawBitmap(int16_t x, int16_t y, const uint8_t *bitmap,
int16_t w, int16_t h, uint16_t color, uint16_t bg),
drawBitmap(int16_t x, int16_t y, uint8_t *bitmap,
int16_t w, int16_t h, uint16_t color),
drawBitmap(int16_t x, int16_t y, uint8_t *bitmap,
int16_t w, int16_t h, uint16_t color, uint16_t bg),
drawXBitmap(int16_t x, int16_t y, const uint8_t *bitmap,
        int16_t w, int16_t h, uint16_t color),
drawChar(int16_t x, int16_t y, unsigned char c, uint16_t color,
```

```
        uint16_t bg, uint8_t size),
        setCursor(int16_t x, int16_t y),
        setTextColor(uint16_t c),
        setTextColor(uint16_t c, uint16_t bg),
        setTextSize(uint8_t s),
        setTextWrap(boolean w),
        cp437(boolean x=true),
        setFont(const GFXfont *f = NULL),
        getTextBounds(char *string, int16_t x, int16_t y,
                int16_t *x1, int16_t *y1, uint16_t *w, uint16_t *h),
        getTextBounds(const __FlashStringHelper *s, int16_t x, int16_t y,
                int16_t *x1, int16_t *y1, uint16_t *w, uint16_t *h);

#if ARDUINO >= 100
        virtual size_t write(uint8_t);
#else
        virtual void   write(uint8_t);
#endif

        int16_t height(void) const;
        int16_t width(void) const;

        uint8_t getRotation(void) const;

        // get current cursor position (get rotation safe maximum values, using: width() for x,
height() for y)
        int16_t getCursorX(void) const;
        int16_t getCursorY(void) const;

        protected:
        const int16_t
        WIDTH, HEIGHT;   // This is the 'raw' display w/h - never changes
        int16_t
        _width, _height, // Display w/h as modified by current rotation
        cursor_x, cursor_y;
        uint16_t
        textcolor, textbgcolor;
        uint8_t
        textsize,
        rotation;
        boolean
        wrap,   // If set, 'wrap' text at right edge of display
        _cp437; // If set, use correct CP437 charset (default is off)
```

```
        GFXfont
        *gfxFont;
};

class Adafruit_GFX_Button {

        public:
        Adafruit_GFX_Button(void);
        // "Classic" initButton() uses center & size
        void initButton(Adafruit_GFX *gfx, int16_t x, int16_t y,
                uint16_t w, uint16_t h, uint16_t outline, uint16_t fill,
                uint16_t textcolor, char *label, uint8_t textsize);
        // New/alt initButton() uses upper-left corner & size
        void initButtonUL(Adafruit_GFX *gfx, int16_t x1, int16_t y1,
                uint16_t w, uint16_t h, uint16_t outline, uint16_t fill,
                uint16_t textcolor, char *label, uint8_t textsize);
        void drawButton(boolean inverted = false);
        boolean contains(int16_t x, int16_t y);

        void press(boolean p);
        boolean isPressed();
        boolean justPressed();
        boolean justReleased();

        private:
        Adafruit_GFX *_gfx;
        int16_t         _x1, _y1; // Coordinates of top-left corner
        uint16_t        _w, _h;
        uint8_t         _textsize;
        uint16_t        _outlinecolor, _fillcolor, _textcolor;
        char            _label[10];

        boolean currstate, laststate;
};

class GFXcanvas1 : public Adafruit_GFX {

        public:
        GFXcanvas1(uint16_t w, uint16_t h);
        ~GFXcanvas1(void);
        void    drawPixel(int16_t x, int16_t y, uint16_t color),
        fillScreen(uint16_t color);
        uint8_t *getBuffer(void);
```

```
        private:
        uint8_t *buffer;
};

class GFXcanvas16 : public Adafruit_GFX {
        GFXcanvas16(uint16_t w, uint16_t h);
        ~GFXcanvas16(void);
        void    drawPixel(int16_t x, int16_t y, uint16_t color),
        fillScreen(uint16_t color);
        uint16_t *getBuffer(void);
        private:
        uint16_t *buffer;
};

#endif // _ADAFRUIT_GFX_H
```

2. Adafruit+GFX.cpp

```
/*
This is the core graphics library for all our displays, providing a common
set of graphics primitives (points, lines, circles, etc.).  It needs to be
paired with a hardware-specific library for each display device we carry
(to handle the lower-level functions).

Adafruit invests time and resources providing this open source code, please
support Adafruit & open-source hardware by purchasing products from Adafruit!

Copyright (c) 2013 Adafruit Industries.  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice,
  this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
```

```
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
 */

#include "Adafruit_GFX.h"
#include "glcdfont.c"
#ifdef __AVR__
  #include <avr/pgmspace.h>
#elif defined(ESP8266) || defined(ESP32)
  #include <pgmspace.h>
#endif

// Many (but maybe not all) non-AVR board installs define macros
// for compatibility with existing PROGMEM-reading AVR code.
// Do our own checks and defines here for good measure...

#ifndef pgm_read_byte
 #define pgm_read_byte(addr) (*(const unsigned char *)(addr))
#endif
#ifndef pgm_read_word
 #define pgm_read_word(addr) (*(const unsigned short *)(addr))
#endif
#ifndef pgm_read_dword
 #define pgm_read_dword(addr) (*(const unsigned long *)(addr))
#endif

// Pointers are a peculiar case...typically 16-bit on AVR boards,
// 32 bits elsewhere.  Try to accommodate both...

#if !defined(__INT_MAX__) || (__INT_MAX__ > 0xFFFF)
 #define pgm_read_pointer(addr) ((void *)pgm_read_dword(addr))
#else
 #define pgm_read_pointer(addr) ((void *)pgm_read_word(addr))
#endif
```

```
#ifndef min
#define min(a,b) (((a) < (b)) ? (a) : (b))
#endif

#ifndef _swap_int16_t
#define _swap_int16_t(a, b) { int16_t t = a; a = b; b = t; }
#endif

Adafruit_GFX::Adafruit_GFX(int16_t w, int16_t h):
WIDTH(w), HEIGHT(h)
{
   _width    = WIDTH;
   _height   = HEIGHT;
   rotation  = 0;
   cursor_y  = cursor_x    = 0;
   textsize  = 1;
   textcolor = textbgcolor = 0xFFFF;
   wrap      = true;
   _cp437    = false;
   gfxFont   = NULL;
}

// Bresenham's algorithm - thx wikpedia
void Adafruit_GFX::writeLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
       uint16_t color) {
   int16_t steep = abs(y1 - y0) > abs(x1 - x0);
   if (steep) {
      _swap_int16_t(x0, y0);
      _swap_int16_t(x1, y1);
   }

   if (x0 > x1) {
      _swap_int16_t(x0, x1);
      _swap_int16_t(y0, y1);
   }

   int16_t dx, dy;
   dx = x1 - x0;
   dy = abs(y1 - y0);

   int16_t err = dx / 2;
   int16_t ystep;
```

```
    if (y0 < y1) {
        ystep = 1;
    } else {
        ystep = -1;
    }

    for (; x0<=x1; x0++) {
        if (steep) {
            writePixel(y0, x0, color);
        } else {
            writePixel(x0, y0, color);
        }
        err -= dy;
        if (err < 0) {
            y0 += ystep;
            err += dx;
        }
    }
}

void Adafruit_GFX::startWrite(){
    // Overwrite in subclasses if desired!
}

void Adafruit_GFX::writePixel(int16_t x, int16_t y, uint16_t color){
    // Overwrite in subclasses if startWrite is defined!
    drawPixel(x, y, color);
}

void Adafruit_GFX::writeFastVLine(int16_t x, int16_t y,
        int16_t h, uint16_t color) {
    // Overwrite in subclasses if startWrite is defined!
    // Can be just writeLine(x, y, x, y+h-1, color);
    // or writeFillRect(x, y, 1, h, color);
    drawFastVLine(x, y, h, color);
}

void Adafruit_GFX::writeFastHLine(int16_t x, int16_t y,
        int16_t w, uint16_t color) {
    // Overwrite in subclasses if startWrite is defined!
    // Example: writeLine(x, y, x+w-1, y, color);
    // or writeFillRect(x, y, w, 1, color);
```

```
    drawFastHLine(x, y, w, color);
}

void Adafruit_GFX::writeFillRect(int16_t x, int16_t y, int16_t w, int16_t h,
        uint16_t color) {
    // Overwrite in subclasses if desired!
    fillRect(x,y,w,h,color);
}

void Adafruit_GFX::endWrite(){
    // Overwrite in subclasses if startWrite is defined!
}

void Adafruit_GFX::drawFastVLine(int16_t x, int16_t y,
        int16_t h, uint16_t color) {
    // Update in subclasses if desired!
    startWrite();
    writeLine(x, y, x, y+h-1, color);
    endWrite();
}

void Adafruit_GFX::drawFastHLine(int16_t x, int16_t y,
        int16_t w, uint16_t color) {
    // Update in subclasses if desired!
    startWrite();
    writeLine(x, y, x+w-1, y, color);
    endWrite();
}

void Adafruit_GFX::fillRect(int16_t x, int16_t y, int16_t w, int16_t h,
        uint16_t color) {
    // Update in subclasses if desired!
    startWrite();
    for (int16_t i=x; i<x+w; i++) {
        writeFastVLine(i, y, h, color);
    }
    endWrite();
}

void Adafruit_GFX::fillScreen(uint16_t color) {
    // Update in subclasses if desired!
    fillRect(0, 0, _width, _height, color);
}
```

```
#define distDiff(a,b) ((max(a,b) - min(a,b))+1)

void Adafruit_GFX::drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
        uint16_t color) {
    // Update in subclasses if desired!
    if(x0 == x1){
        drawFastVLine(x0, y0, distDiff(y0,y1), color);
    } else if(y0 == y1){
        drawFastHLine(x0, y0, distDiff(x0,x1), color);
    } else {
        startWrite();
        writeLine(x0, y0, x1, y1, color);
        endWrite();
    }
}


// Draw a circle outline
void Adafruit_GFX::drawCircle(int16_t x0, int16_t y0, int16_t r,
        uint16_t color) {
    int16_t f = 1 - r;
    int16_t ddF_x = 1;
    int16_t ddF_y = -2 * r;
    int16_t x = 0;
    int16_t y = r;

    startWrite();
    writePixel(x0  , y0+r, color);
    writePixel(x0  , y0-r, color);
    writePixel(x0+r, y0  , color);
    writePixel(x0-r, y0  , color);

    while (x<y) {
        if (f >= 0) {
            y--;
            ddF_y += 2;
            f += ddF_y;
        }
        x++;
        ddF_x += 2;
        f += ddF_x;
```

```
        writePixel(x0 + x, y0 + y, color);
        writePixel(x0 - x, y0 + y, color);
        writePixel(x0 + x, y0 - y, color);
        writePixel(x0 - x, y0 - y, color);
        writePixel(x0 + y, y0 + x, color);
        writePixel(x0 - y, y0 + x, color);
        writePixel(x0 + y, y0 - x, color);
        writePixel(x0 - y, y0 - x, color);
    }
    endWrite();
}

void Adafruit_GFX::drawCircleHelper( int16_t x0, int16_t y0,
        int16_t r, uint8_t cornername, uint16_t color) {
    int16_t f     = 1 - r;
    int16_t ddF_x = 1;
    int16_t ddF_y = -2 * r;
    int16_t x     = 0;
    int16_t y     = r;

    while (x<y) {
        if (f >= 0) {
            y--;
            ddF_y += 2;
            f     += ddF_y;
        }
        x++;
        ddF_x += 2;
        f     += ddF_x;
        if (cornername & 0x4) {
            writePixel(x0 + x, y0 + y, color);
            writePixel(x0 + y, y0 + x, color);
        }
        if (cornername & 0x2) {
            writePixel(x0 + x, y0 - y, color);
            writePixel(x0 + y, y0 - x, color);
        }
        if (cornername & 0x8) {
            writePixel(x0 - y, y0 + x, color);
            writePixel(x0 - x, y0 + y, color);
        }
        if (cornername & 0x1) {
            writePixel(x0 - y, y0 - x, color);
```

```
            writePixel(x0 - x, y0 - y, color);
        }
    }
}

void Adafruit_GFX::fillCircle(int16_t x0, int16_t y0, int16_t r,
        uint16_t color) {
    startWrite();
    writeFastVLine(x0, y0-r, 2*r+1, color);
    fillCircleHelper(x0, y0, r, 3, 0, color);
    endWrite();
}

// Used to do circles and roundrects
void Adafruit_GFX::fillCircleHelper(int16_t x0, int16_t y0, int16_t r,
        uint8_t cornername, int16_t delta, uint16_t color) {

    int16_t f     = 1 - r;
    int16_t ddF_x = 1;
    int16_t ddF_y = -2 * r;
    int16_t x     = 0;
    int16_t y     = r;

    while (x<y) {
        if (f >= 0) {
            y--;
            ddF_y += 2;
            f     += ddF_y;
        }
        x++;
        ddF_x += 2;
        f     += ddF_x;

        if (cornername & 0x1) {
            writeFastVLine(x0+x, y0-y, 2*y+1+delta, color);
            writeFastVLine(x0+y, y0-x, 2*x+1+delta, color);
        }
        if (cornername & 0x2) {
            writeFastVLine(x0-x, y0-y, 2*y+1+delta, color);
            writeFastVLine(x0-y, y0-x, 2*x+1+delta, color);
        }
    }
}
```

```
// Draw a rectangle
void Adafruit_GFX::drawRect(int16_t x, int16_t y, int16_t w, int16_t h,
        uint16_t color) {
    startWrite();
    writeFastHLine(x, y, w, color);
    writeFastHLine(x, y+h-1, w, color);
    writeFastVLine(x, y, h, color);
    writeFastVLine(x+w-1, y, h, color);
    endWrite();
}

// Draw a rounded rectangle
void Adafruit_GFX::drawRoundRect(int16_t x, int16_t y, int16_t w,
        int16_t h, int16_t r, uint16_t color) {
    // smarter version
    startWrite();
    writeFastHLine(x+r  , y    , w-2*r, color); // Top
    writeFastHLine(x+r  , y+h-1, w-2*r, color); // Bottom
    writeFastVLine(x    , y+r  , h-2*r, color); // Left
    writeFastVLine(x+w-1, y+r  , h-2*r, color); // Right
    // draw four corners
    drawCircleHelper(x+r    , y+r    , r, 1, color);
    drawCircleHelper(x+w-r-1, y+r    , r, 2, color);
    drawCircleHelper(x+w-r-1, y+h-r-1, r, 4, color);
    drawCircleHelper(x+r    , y+h-r-1, r, 8, color);
    endWrite();
}

// Fill a rounded rectangle
void Adafruit_GFX::fillRoundRect(int16_t x, int16_t y, int16_t w,
        int16_t h, int16_t r, uint16_t color) {
    // smarter version
    startWrite();
    writeFillRect(x+r, y, w-2*r, h, color);

    // draw four corners
    fillCircleHelper(x+w-r-1, y+r, r, 1, h-2*r-1, color);
    fillCircleHelper(x+r    , y+r, r, 2, h-2*r-1, color);
    endWrite();
}

// Draw a triangle
```

```
void Adafruit_GFX::drawTriangle(int16_t x0, int16_t y0,
        int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint16_t color) {
    drawLine(x0, y0, x1, y1, color);
    drawLine(x1, y1, x2, y2, color);
    drawLine(x2, y2, x0, y0, color);
}

// Fill a triangle
void Adafruit_GFX::fillTriangle(int16_t x0, int16_t y0,
        int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint16_t color) {

    int16_t a, b, y, last;

    // Sort coordinates by Y order (y2 >= y1 >= y0)
    if (y0 > y1) {
        _swap_int16_t(y0, y1); _swap_int16_t(x0, x1);
    }
    if (y1 > y2) {
        _swap_int16_t(y2, y1); _swap_int16_t(x2, x1);
    }
    if (y0 > y1) {
        _swap_int16_t(y0, y1); _swap_int16_t(x0, x1);
    }

    startWrite();
    if(y0 == y2) { // Handle awkward all-on-same-line case as its own thing
        a = b = x0;
        if(x1 < a)      a = x1;
        else if(x1 > b) b = x1;
        if(x2 < a)      a = x2;
        else if(x2 > b) b = x2;
        writeFastHLine(a, y0, b-a+1, color);
        endWrite();
        return;
    }

    int16_t
    dx01 = x1 - x0,
    dy01 = y1 - y0,
    dx02 = x2 - x0,
    dy02 = y2 - y0,
    dx12 = x2 - x1,
    dy12 = y2 - y1;
```

```
int32_t
sa   = 0,
sb   = 0;

// For upper part of triangle, find scanline crossings for segments
// 0-1 and 0-2.  If y1=y2 (flat-bottomed triangle), the scanline y1
// is included here (and second loop will be skipped, avoiding a /0
// error there), otherwise scanline y1 is skipped here and handled
// in the second loop...which also avoids a /0 error here if y0=y1
// (flat-topped triangle).
if(y1 == y2) last = y1;   // Include y1 scanline
else         last = y1-1; // Skip it

for(y=y0; y<=last; y++) {
   a   = x0 + sa / dy01;
   b   = x0 + sb / dy02;
   sa += dx01;
   sb += dx02;
   /* longhand:
a = x0 + (x1 - x0) * (y - y0) / (y1 - y0);
b = x0 + (x2 - x0) * (y - y0) / (y2 - y0);
    */
   if(a > b) _swap_int16_t(a,b);
   writeFastHLine(a, y, b-a+1, color);
}

// For lower part of triangle, find scanline crossings for segments
// 0-2 and 1-2.  This loop is skipped if y1=y2.
sa = dx12 * (y - y1);
sb = dx02 * (y - y0);
for(; y<=y2; y++) {
   a   = x1 + sa / dy12;
   b   = x0 + sb / dy02;
   sa += dx12;
   sb += dx02;
   /* longhand:
a = x1 + (x2 - x1) * (y - y1) / (y2 - y1);
b = x0 + (x2 - x0) * (y - y0) / (y2 - y0);
    */
   if(a > b) _swap_int16_t(a,b);
   writeFastHLine(a, y, b-a+1, color);
}
endWrite();
```

```
}

// Draw a 1-bit image (bitmap) at the specified (x,y) position from the
// provided bitmap buffer (must be PROGMEM memory) using the specified
// foreground color (unset bits are transparent).
void Adafruit_GFX::drawBitmap(int16_t x, int16_t y,
        const uint8_t *bitmap, int16_t w, int16_t h, uint16_t color) {

    int16_t i, j, byteWidth = (w + 7) / 8;
    uint8_t byte = 0;

    startWrite();
    for(j=0; j<h; j++) {
        for(i=0; i<w; i++) {
            if(i & 7) byte <<= 1;
            else      byte   = pgm_read_byte(bitmap + j * byteWidth + i / 8);
            if(byte & 0x80) writePixel(x+i, y+j, color);
        }
    }
    endWrite();
}

// Draw a 1-bit image (bitmap) at the specified (x,y) position from the
// provided bitmap buffer (must be PROGMEM memory) using the specified
// foreground (for set bits) and background (for clear bits) colors.
void Adafruit_GFX::drawBitmap(int16_t x, int16_t y,
        const uint8_t *bitmap, int16_t w, int16_t h, uint16_t color, uint16_t bg) {

    int16_t i, j, byteWidth = (w + 7) / 8;
    uint8_t byte = 0;

    startWrite();
    for(j=0; j<h; j++) {
        for(i=0; i<w; i++ ) {
            if(i & 7) byte <<= 1;
            else      byte   = pgm_read_byte(bitmap + j * byteWidth + i / 8);
            if(byte & 0x80) writePixel(x+i, y+j, color);
            else            writePixel(x+i, y+j, bg);
        }
    }
    endWrite();
}
```

```
// drawBitmap() variant for RAM-resident (not PROGMEM) bitmaps.
void Adafruit_GFX::drawBitmap(int16_t x, int16_t y,
        uint8_t *bitmap, int16_t w, int16_t h, uint16_t color) {

    int16_t i, j, byteWidth = (w + 7) / 8;
    uint8_t byte = 0;

    startWrite();
    for(j=0; j<h; j++) {
        for(i=0; i<w; i++ ) {
            if(i & 7) byte <<= 1;
            else      byte   = bitmap[j * byteWidth + i / 8];
            if(byte & 0x80) writePixel(x+i, y+j, color);
        }
    }
    endWrite();
}

// drawBitmap() variant w/background for RAM-resident (not PROGMEM) bitmaps.
void Adafruit_GFX::drawBitmap(int16_t x, int16_t y,
        uint8_t *bitmap, int16_t w, int16_t h, uint16_t color, uint16_t bg) {

    int16_t i, j, byteWidth = (w + 7) / 8;
    uint8_t byte = 0;

    startWrite();
    for(j=0; j<h; j++) {
        for(i=0; i<w; i++ ) {
            if(i & 7) byte <<= 1;
            else      byte   = bitmap[j * byteWidth + i / 8];
            if(byte & 0x80) writePixel(x+i, y+j, color);
            else            writePixel(x+i, y+j, bg);
        }
    }
    endWrite();
}

//Draw XBitMap Files (*.xbm), exported from GIMP,
//Usage: Export from GIMP to *.xbm, rename *.xbm to *.c and open in editor.
//C Array can be directly used with this function
void Adafruit_GFX::drawXBitmap(int16_t x, int16_t y,
        const uint8_t *bitmap, int16_t w, int16_t h, uint16_t color) {
```

```
    int16_t i, j, byteWidth = (w + 7) / 8;
    uint8_t byte = 0;

    startWrite();
    for(j=0; j<h; j++) {
      for(i=0; i<w; i++ ) {
        if(i & 7) byte >>= 1;
        else      byte   = pgm_read_byte(bitmap + j * byteWidth + i / 8);
        if(byte & 0x01) writePixel(x+i, y+j, color);
      }
    }
    endWrite();
}

// Draw a character
void Adafruit_GFX::drawChar(int16_t x, int16_t y, unsigned char c,
       uint16_t color, uint16_t bg, uint8_t size) {

    if(!gfxFont) { // 'Classic' built-in font

       if((x >= _width)            || // Clip right
             (y >= _height)           || // Clip bottom
             ((x + 6 * size - 1) < 0) || // Clip left
             ((y + 8 * size - 1) < 0))   // Clip top
          return;

       if(!_cp437 && (c >= 176)) c++; // Handle 'classic' charset behavior

       startWrite();
       for(int8_t i=0; i<6; i++ ) {
          uint8_t line;
          if(i < 5) line = pgm_read_byte(font+(c*5)+i);
          else      line = 0x0;
          for(int8_t j=0; j<8; j++, line >>= 1) {
             if(line & 0x1) {
                if(size == 1) writePixel(x+i, y+j, color);
                else          writeFillRect(x+(i*size), y+(j*size), size, size, color);
             } else if(bg != color) {
                if(size == 1) writePixel(x+i, y+j, bg);
                else          writeFillRect(x+i*size, y+j*size, size, size, bg);
             }
          }
       }
```

```
    endWrite();

  } else { // Custom font

    // Character is assumed previously filtered by write() to eliminate
    // newlines, returns, non-printable characters, etc.  Calling drawChar()
    // directly with 'bad' characters of font may cause mayhem!

    c -= pgm_read_byte(&gfxFont->first);
    GFXglyph *glyph  = &(((GFXglyph *)pgm_read_pointer(&gfxFont->glyph))[c]);
    uint8_t  *bitmap = (uint8_t *)pgm_read_pointer(&gfxFont->bitmap);

    uint16_t bo = pgm_read_word(&glyph->bitmapOffset);
    uint8_t  w  = pgm_read_byte(&glyph->width),
             h  = pgm_read_byte(&glyph->height);
    int8_t   xo = pgm_read_byte(&glyph->xOffset),
             yo = pgm_read_byte(&glyph->yOffset);
    uint8_t  xx, yy, bits = 0, bit = 0;
    int16_t  xo16 = 0, yo16 = 0;

    if(size > 1) {
      xo16 = xo;
      yo16 = yo;
    }

    // Todo: Add character clipping here

    // NOTE: THERE IS NO 'BACKGROUND' COLOR OPTION ON CUSTOM FONTS.
    // THIS IS ON PURPOSE AND BY DESIGN.  The background color feature
    // has typically been used with the 'classic' font to overwrite old
    // screen contents with new data.  This ONLY works because the
    // characters are a uniform size; it's not a sensible thing to do with
    // proportionally-spaced fonts with glyphs of varying sizes (and that
    // may overlap).  To replace previously-drawn text when using a custom
    // font, use the getTextBounds() function to determine the smallest
    // rectangle encompassing a string, erase the area with fillRect(),
    // then draw new text.  This WILL infortunately 'blink' the text, but
    // is unavoidable.  Drawing 'background' pixels will NOT fix this,
    // only creates a new set of problems.  Have an idea to work around
    // this (a canvas object type for MCUs that can afford the RAM and
    // displays supporting setAddrWindow() and pushColors()), but haven't
    // implemented this yet.
```

```
      startWrite();
      for(yy=0; yy<h; yy++) {
        for(xx=0; xx<w; xx++) {
          if(!(bit++ & 7)) {
            bits = pgm_read_byte(&bitmap[bo++]);
          }
          if(bits & 0x80) {
            if(size == 1) {
              writePixel(x+xo+xx, y+yo+yy, color);
            } else {
              writeFillRect(x+(xo16+xx)*size, y+(yo16+yy)*size, size, size, color);
            }
          }
          bits <<= 1;
        }
      }
      endWrite();

    } // End classic vs custom font
}

#if ARDUINO >= 100
size_t Adafruit_GFX::write(uint8_t c) {
#else
void Adafruit_GFX::write(uint8_t c) {
#endif
    if(!gfxFont) { // 'Classic' built-in font

      if(c == '\n') {
        cursor_y += textsize*8;
        cursor_x  = 0;
      } else if(c == '\r') {
        // skip em
      } else {
        if(wrap && ((cursor_x + textsize * 6) >= _width)) { // Heading off edge?
          cursor_x  = 0;            // Reset x to zero
          cursor_y += textsize * 8; // Advance y one line
        }
        drawChar(cursor_x, cursor_y, c, textcolor, textbgcolor, textsize);
        cursor_x += textsize * 6;
      }

    } else { // Custom font
```

```
      if(c == '\n') {
        cursor_x  = 0;
        cursor_y += (int16_t)textsize *
                    (uint8_t)pgm_read_byte(&gfxFont->yAdvance);
      } else if(c != '\r') {
        uint8_t first = pgm_read_byte(&gfxFont->first);
        if((c >= first) && (c <= (uint8_t)pgm_read_byte(&gfxFont->last))) {
          uint8_t   c2    = c - pgm_read_byte(&gfxFont->first);
          GFXglyph *glyph = &(((GFXglyph *)pgm_read_pointer(&gfxFont->glyph))[c2]);
          uint8_t   w     = pgm_read_byte(&glyph->width),
                    h     = pgm_read_byte(&glyph->height);
          if((w > 0) && (h > 0)) { // Is there an associated bitmap?
            int16_t xo = (int8_t)pgm_read_byte(&glyph->xOffset); // sic
            if(wrap && ((cursor_x + textsize * (xo + w)) >= _width)) {
              // Drawing character would go off right edge; wrap to new line
              cursor_x  = 0;
              cursor_y += (int16_t)textsize *
                          (uint8_t)pgm_read_byte(&gfxFont->yAdvance);
            }
            drawChar(cursor_x, cursor_y, c, textcolor, textbgcolor, textsize);
          }
          cursor_x += pgm_read_byte(&glyph->xAdvance) * (int16_t)textsize;
        }
      }

    }
#if ARDUINO >= 100
    return 1;
#endif
}

void Adafruit_GFX::setCursor(int16_t x, int16_t y) {
    cursor_x = x;
    cursor_y = y;
}

int16_t Adafruit_GFX::getCursorX(void) const {
    return cursor_x;
}

int16_t Adafruit_GFX::getCursorY(void) const {
    return cursor_y;
```

```
}

void Adafruit_GFX::setTextSize(uint8_t s) {
   textsize = (s > 0) ? s : 1;
}

void Adafruit_GFX::setTextColor(uint16_t c) {
   // For 'transparent' background, we'll set the bg
   // to the same as fg instead of using a flag
   textcolor = textbgcolor = c;
}

void Adafruit_GFX::setTextColor(uint16_t c, uint16_t b) {
   textcolor   = c;
   textbgcolor = b;
}

void Adafruit_GFX::setTextWrap(boolean w) {
   wrap = w;
}

uint8_t Adafruit_GFX::getRotation(void) const {
   return rotation;
}

void Adafruit_GFX::setRotation(uint8_t x) {
   rotation = (x & 3);
   switch(rotation) {
      case 0:
      case 2:
         _width  = WIDTH;
         _height = HEIGHT;
         break;
      case 1:
      case 3:
         _width  = HEIGHT;
         _height = WIDTH;
         break;
   }
}

// Enable (or disable) Code Page 437-compatible charset.
// There was an error in glcdfont.c for the longest time -- one character
```

```
// (#176, the 'light shade' block) was missing -- this threw off the index
// of every character that followed it.  But a TON of code has been written
// with the erroneous character indices.  By default, the library uses the
// original 'wrong' behavior and old sketches will still work.  Pass 'true'
// to this function to use correct CP437 character values in your code.
void Adafruit_GFX::cp437(boolean x) {
    _cp437 = x;
}

void Adafruit_GFX::setFont(const GFXfont *f) {
    if(f) {          // Font struct pointer passed in?
        if(!gfxFont) { // And no current font struct?
            // Switching from classic to new font behavior.
            // Move cursor pos down 6 pixels so it's on baseline.
            cursor_y += 6;
        }
    } else if(gfxFont) { // NULL passed.  Current font struct defined?
        // Switching from new to classic font behavior.
        // Move cursor pos up 6 pixels so it's at top-left of char.
        cursor_y -= 6;
    }
    gfxFont = (GFXfont *)f;
}

// Pass string and a cursor position, returns UL corner and W,H.
void Adafruit_GFX::getTextBounds(char *str, int16_t x, int16_t y,
        int16_t *x1, int16_t *y1, uint16_t *w, uint16_t *h) {
    uint8_t c; // Current character

    *x1 = x;
    *y1 = y;
    *w  = *h = 0;

    if(gfxFont) {

        GFXglyph *glyph;
        uint8_t   first = pgm_read_byte(&gfxFont->first),
                  last  = pgm_read_byte(&gfxFont->last),
                  gw, gh, xa;
        int8_t    xo, yo;
        int16_t   minx = _width, miny = _height, maxx = -1, maxy = -1,
                  gx1, gy1, gx2, gy2, ts = (int16_t)textsize,
                  ya = ts * (uint8_t)pgm_read_byte(&gfxFont->yAdvance);
```

```
    while((c = *str++)) {
      if(c != '\n') { // Not a newline
        if(c != '\r') { // Not a carriage return, is normal char
          if((c >= first) && (c <= last)) { // Char present in current font
            c    -= first;
            glyph = &(((GFXglyph *)pgm_read_pointer(&gfxFont->glyph))[c]);
            gw    = pgm_read_byte(&glyph->width);
            gh    = pgm_read_byte(&glyph->height);
            xa    = pgm_read_byte(&glyph->xAdvance);
            xo    = pgm_read_byte(&glyph->xOffset);
            yo    = pgm_read_byte(&glyph->yOffset);
            if(wrap && ((x + (((int16_t)xo + gw) * ts)) >= _width)) {
              // Line wrap
              x  = 0;  // Reset x to 0
              y += ya; // Advance y by 1 line
            }
            gx1 = x   + xo * ts;
            gy1 = y   + yo * ts;
            gx2 = gx1 + gw * ts - 1;
            gy2 = gy1 + gh * ts - 1;
            if(gx1 < minx) minx = gx1;
            if(gy1 < miny) miny = gy1;
            if(gx2 > maxx) maxx = gx2;
            if(gy2 > maxy) maxy = gy2;
            x += xa * ts;
          }
        } // Carriage return = do nothing
      } else { // Newline
        x  = 0;  // Reset x
        y += ya; // Advance y by 1 line
      }
    }
    // End of string
    *x1 = minx;
    *y1 = miny;
    if(maxx >= minx) *w  = maxx - minx + 1;
    if(maxy >= miny) *h  = maxy - miny + 1;

  } else { // Default font

    uint16_t lineWidth = 0, maxWidth = 0; // Width of current, all lines
```

```
      while((c = *str++)) {
        if(c != '\n') { // Not a newline
          if(c != '\r') { // Not a carriage return, is normal char
            if(wrap && ((x + textsize * 6) >= _width)) {
              x  = 0;            // Reset x to 0
              y += textsize * 8; // Advance y by 1 line
              if(lineWidth > maxWidth) maxWidth = lineWidth; // Save widest line
              lineWidth  = textsize * 6; // First char on new line
            } else { // No line wrap, just keep incrementing X
              lineWidth += textsize * 6; // Includes interchar x gap
            }
          } // Carriage return = do nothing
        } else { // Newline
          x  = 0;            // Reset x to 0
          y += textsize * 8; // Advance y by 1 line
          if(lineWidth > maxWidth) maxWidth = lineWidth; // Save widest line
          lineWidth = 0;     // Reset lineWidth for new line
        }
      }
      // End of string
      if(lineWidth) y += textsize * 8; // Add height of last (or only) line
      if(lineWidth > maxWidth) maxWidth = lineWidth; // Is the last or only line the widest?
      *w = maxWidth - 1;               // Don't include last interchar x gap
      *h = y - *y1;

    } // End classic vs custom font
}

// Same as above, but for PROGMEM strings
void Adafruit_GFX::getTextBounds(const __FlashStringHelper *str,
      int16_t x, int16_t y, int16_t *x1, int16_t *y1, uint16_t *w, uint16_t *h) {
    uint8_t *s = (uint8_t *)str, c;

    *x1 = x;
    *y1 = y;
    *w  = *h = 0;

    if(gfxFont) {

      GFXglyph *glyph;
      uint8_t   first = pgm_read_byte(&gfxFont->first),
            last  = pgm_read_byte(&gfxFont->last),
            gw, gh, xa;
```

```
int8_t    xo, yo;
int16_t   minx = _width, miny = _height, maxx = -1, maxy = -1,
        gx1, gy1, gx2, gy2, ts = (int16_t)textsize,
        ya = ts * (uint8_t)pgm_read_byte(&gfxFont->yAdvance);

while((c = pgm_read_byte(s++))) {
   if(c != '\n') { // Not a newline
      if(c != '\r') { // Not a carriage return, is normal char
         if((c >= first) && (c <= last)) { // Char present in current font
            c    -= first;
            glyph = &(((GFXglyph *)pgm_read_pointer(&gfxFont->glyph))[c]);
            gw    = pgm_read_byte(&glyph->width);
            gh    = pgm_read_byte(&glyph->height);
            xa    = pgm_read_byte(&glyph->xAdvance);
            xo    = pgm_read_byte(&glyph->xOffset);
            yo    = pgm_read_byte(&glyph->yOffset);
            if(wrap && ((x + (((int16_t)xo + gw) * ts)) >= _width)) {
               // Line wrap
               x  = 0;  // Reset x to 0
               y += ya; // Advance y by 1 line
            }
            gx1 = x   + xo * ts;
            gy1 = y   + yo * ts;
            gx2 = gx1 + gw * ts - 1;
            gy2 = gy1 + gh * ts - 1;
            if(gx1 < minx) minx = gx1;
            if(gy1 < miny) miny = gy1;
            if(gx2 > maxx) maxx = gx2;
            if(gy2 > maxy) maxy = gy2;
            x += xa * ts;
         }
      } // Carriage return = do nothing
   } else { // Newline
      x  = 0;  // Reset x
      y += ya; // Advance y by 1 line
   }
}
// End of string
*x1 = minx;
*y1 = miny;
if(maxx >= minx) *w  = maxx - minx + 1;
if(maxy >= miny) *h  = maxy - miny + 1;
```

```
    } else { // Default font

        uint16_t lineWidth = 0, maxWidth = 0; // Width of current, all lines

        while((c = pgm_read_byte(s++))) {
            if(c != '\n') { // Not a newline
                if(c != '\r') { // Not a carriage return, is normal char
                    if(wrap && ((x + textsize * 6) >= _width)) {
                        x  = 0;            // Reset x to 0
                        y += textsize * 8; // Advance y by 1 line
                        if(lineWidth > maxWidth) maxWidth = lineWidth; // Save widest line
                        lineWidth  = textsize * 6; // First char on new line
                    } else { // No line wrap, just keep incrementing X
                        lineWidth += textsize * 6; // Includes interchar x gap
                    }
                } // Carriage return = do nothing
            } else { // Newline
                x  = 0;            // Reset x to 0
                y += textsize * 8; // Advance y by 1 line
                if(lineWidth > maxWidth) maxWidth = lineWidth; // Save widest line
                lineWidth = 0;     // Reset lineWidth for new line
            }
        }
        // End of string
        if(lineWidth) y += textsize * 8; // Add height of last (or only) line
        if(lineWidth > maxWidth) maxWidth = lineWidth; // Is the last or only line the widest?
        *w = maxWidth - 1;               // Don't include last interchar x gap
        *h = y - *y1;

    } // End classic vs custom font
}

// Return the size of the display (per current rotation)
int16_t Adafruit_GFX::width(void) const {
    return _width;
}

int16_t Adafruit_GFX::height(void) const {
    return _height;
}

void Adafruit_GFX::invertDisplay(boolean i) {
    // Do nothing, must be subclassed if supported by hardware
```

```
}

/***************************************************************************/
// code for the GFX button UI element

Adafruit_GFX_Button::Adafruit_GFX_Button(void) {
  _gfx = 0;
}

// Classic initButton() function: pass center & size
void Adafruit_GFX_Button::initButton(
 Adafruit_GFX *gfx, int16_t x, int16_t y, uint16_t w, uint16_t h,
 uint16_t outline, uint16_t fill, uint16_t textcolor,
 char *label, uint8_t textsize)
{
  // Tweak arguments and pass to the newer initButtonUL() function...
  initButtonUL(gfx, x - (w / 2), y - (h / 2), w, h, outline, fill,
    textcolor, label, textsize);
}

// Newer function instead accepts upper-left corner & size
void Adafruit_GFX_Button::initButtonUL(
 Adafruit_GFX *gfx, int16_t x1, int16_t y1, uint16_t w, uint16_t h,
 uint16_t outline, uint16_t fill, uint16_t textcolor,
 char *label, uint8_t textsize)
{
  _x1           = x1;
  _y1           = y1;
  _w            = w;
  _h            = h;
  _outlinecolor = outline;
  _fillcolor    = fill;
  _textcolor    = textcolor;
  _textsize     = textsize;
  _gfx          = gfx;
  strncpy(_label, label, 9);
}

void Adafruit_GFX_Button::drawButton(boolean inverted) {
  uint16_t fill, outline, text;

  if(!inverted) {
    fill    = _fillcolor;
```

```
    outline = _outlinecolor;
    text    = _textcolor;
  } else {
    fill    = _textcolor;
    outline = _outlinecolor;
    text    = _fillcolor;
  }

  uint8_t r = min(_w, _h) / 4; // Corner radius
  _gfx->fillRoundRect(_x1, _y1, _w, _h, r, fill);
  _gfx->drawRoundRect(_x1, _y1, _w, _h, r, outline);

  _gfx->setCursor(_x1 + (_w/2) - (strlen(_label) * 3 * _textsize),
    _y1 + (_h/2) - (4 * _textsize));
  _gfx->setTextColor(text);
  _gfx->setTextSize(_textsize);
  _gfx->print(_label);
}

boolean Adafruit_GFX_Button::contains(int16_t x, int16_t y) {
  return ((x >= _x1) && (x < (_x1 + _w)) &&
          (y >= _y1) && (y < (_y1 + _h)));
}

void Adafruit_GFX_Button::press(boolean p) {
  laststate = currstate;
  currstate = p;
}

boolean Adafruit_GFX_Button::isPressed() { return currstate; }
boolean Adafruit_GFX_Button::justPressed() { return (currstate && !laststate); }
boolean Adafruit_GFX_Button::justReleased() { return (!currstate && laststate); }

// ----------------------------------------------------------------------

// GFXcanvas1 and GFXcanvas16 (currently a WIP, don't get too comfy with the
// implementation) provide 1- and 16-bit offscreen canvases, the address of
// which can be passed to drawBitmap() or pushColors() (the latter appears
// to only be in Adafruit_TFTLCD at this time).  This is here mostly to
// help with the recently-added proportionally-spaced fonts; adds a way to
// refresh a section of the screen without a massive flickering clear-and-
// redraw...but maybe you'll find other uses too.  VERY RAM-intensive, since
// the buffer is in MCU memory and not the display driver...GXFcanvas1 might
```

```
// be minimally useful on an Uno-class board, but this and GFXcanvas16 are
// much more likely to require at least a Mega or various recent ARM-type
// boards (recommended, as the text+bitmap draw can be pokey).  GFXcanvas1
// requires 1 bit per pixel (rounded up to nearest byte per scanline),
// GFXcanvas16 requires 2 bytes per pixel (no scanline pad).
// NOT EXTENSIVELY TESTED YET.  MAY CONTAIN WORST BUGS KNOWN TO
HUMANKIND.

GFXcanvas1::GFXcanvas1(uint16_t w, uint16_t h) : Adafruit_GFX(w, h) {
  uint16_t bytes = ((w + 7) / 8) * h;
  if((buffer = (uint8_t *)malloc(bytes))) {
    memset(buffer, 0, bytes);
  }
}

GFXcanvas1::~GFXcanvas1(void) {
  if(buffer) free(buffer);
}

uint8_t* GFXcanvas1::getBuffer(void) {
  return buffer;
}

void GFXcanvas1::drawPixel(int16_t x, int16_t y, uint16_t color) {
  // Bitmask tables of 0x80>>X and ~(0x80>>X), because X>>Y is slow on AVR
  static const uint8_t PROGMEM
  GFXsetBit[] = { 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01 },
  GFXclrBit[] = { 0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE };

  if(buffer) {
    if((x < 0) || (y < 0) || (x >= _width) || (y >= _height)) return;

    int16_t t;
    switch(rotation) {
      case 1:
        t = x;
        x = WIDTH  - 1 - y;
        y = t;
        break;
      case 2:
        x = WIDTH  - 1 - x;
        y = HEIGHT - 1 - y;
        break;
```

```
        case 3:
          t = x;
          x = y;
          y = HEIGHT - 1 - t;
          break;
      }

      uint8_t *ptr = &buffer[(x / 8) + y * ((WIDTH + 7) / 8)];
      if(color) *ptr |= pgm_read_byte(&GFXsetBit[x & 7]);
      else      *ptr &= pgm_read_byte(&GFXclrBit[x & 7]);
   }
}

void GFXcanvas1::fillScreen(uint16_t color) {
   if(buffer) {
      uint16_t bytes = ((WIDTH + 7) / 8) * HEIGHT;
      memset(buffer, color ? 0xFF : 0x00, bytes);
   }
}

GFXcanvas16::GFXcanvas16(uint16_t w, uint16_t h) : Adafruit_GFX(w, h) {
   uint16_t bytes = w * h * 2;
   if((buffer = (uint16_t *)malloc(bytes))) {
      memset(buffer, 0, bytes);
   }
}

GFXcanvas16::~GFXcanvas16(void) {
   if(buffer) free(buffer);
}

uint16_t* GFXcanvas16::getBuffer(void) {
   return buffer;
}

void GFXcanvas16::drawPixel(int16_t x, int16_t y, uint16_t color) {
   if(buffer) {
      if((x < 0) || (y < 0) || (x >= _width) || (y >= _height)) return;

      int16_t t;
      switch(rotation) {
        case 1:
          t = x;
```

```
            x = WIDTH  - 1 - y;
            y = t;
            break;
        case 2:
            x = WIDTH  - 1 - x;
            y = HEIGHT - 1 - y;
            break;
        case 3:
            t = x;
            x = y;
            y = HEIGHT - 1 - t;
            break;
    }

    buffer[x + y * WIDTH] = color;
  }
}

void GFXcanvas16::fillScreen(uint16_t color) {
  if(buffer) {
    uint8_t hi = color >> 8, lo = color & 0xFF;
    if(hi == lo) {
      memset(buffer, lo, WIDTH * HEIGHT * 2);
    } else {
      uint16_t i, pixels = WIDTH * HEIGHT;
      for(i=0; i<pixels; i++) buffer[i] = color;
    }
  }
}
```

Appendix I:
Final Code

```
#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>
#include <EEPROM.h>
#include <SPI.h>

#include <ArduinoJson.h>
```

```
#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_ST7735.h> // Hardware-specific library
#include "RSSReader.h"

ESP8266WebServer server(80);

float timeNow = 1150;

const char* ssid = "test";
const char* passphrase = "test";
/*const char* zipcode = "test";
const char* zipcode2 = "test";
const char* auxscreen = "1"; */ //Probably don't need
String st;
String content;
int statusCode;
int val;
int inputPin ;
int pirState = LOW; // we start, assuming no motion detected
int count;
int i;
int change = 0;

int Button_Press = 0;


//const char SSID[]     = "ND-guest";
//const char PASSWORD[] = "";
//String WU_API_KEY ="2df07ee2d9774288";
//String WU_LOCATION = "46617"; //read from EEPROM
//String GET = "/api/" + WU_API_KEY + "/conditions/q/" + "WU_LOCATION" + ".json";



// 5 minutes between update checks. The free developer account has a limit
// on the  number of calls so don't go wild.
#define DELAY_NORMAL    (30*1000)
// 20 minute delay between updates after an error
#define DELAY_ERROR     (20*60*1000)

#define WUNDERGROUND "api.wunderground.com"
#define NUMBER_OF_FEEDS (sizeof(rssFeedURLs) / sizeof(char *))
int rssFeedIndex;
```

```
int QuoteIndex;

// Create RSS reader instance with 3 second timeout
RSSReader reader = RSSReader(300);

// Array of feed URLs
const char *rssFeedURLs [] = {
  "https://sports.yahoo.com/mlb/rss.xml", // A Few
  "https://sports.yahoo.com/top/rss.xml",
  "www.npr.org/rss/rss.php?id=1001", // 2 headlines
  "http://feeds.bbci.co.uk/news/rss.xml", //Not working
  "http://hosted.ap.org/lineups/SCIENCEHEADS-rss_2.0.xml?SITE=OHLIM&SECTION=HOME",
// A few
  "http://www.latimes.com/rss2.0.xml", // works
  "http://feeds.reuters.com/reuters/topNews",
  "rssfeeds.usatoday.com/usatoday-NewsTopStories",//doesn't work
};

// Array of Inspirational Quotes
const char *InspirationalQuotes[] = {
  "Try, like we did",
  "Tweet at your parents",
};




//Necessary for RSSReader
void setTitleCallback(pt2Function titleCallback);
void setDescCallback(pt2Function descCallback);
void setPubDateCallback(pt2Function dateCallback);



#define TFT_CS     16
#define TFT_RST    -1  // you can also connect this to the Arduino reset
#define TFT_DC     12
#define white 0xFFFF
#define black 0x0000
#define yellow 0xFFE0
#define cyan 0x07FF

#define TFT_MOSI 14
```

```
#define TFT_SCLK 13
Adafruit_ST7735 tft = Adafruit_ST7735(-1, TFT_DC, TFT_MOSI, TFT_SCLK, TFT_RST);
#define muxA 5
#define muxB 2
#define muxC 15

void clearTFT();
void drawTime(float a, byte b);
void drawSun(byte a);
void drawCloud();
void drawRain(byte a);
void drawStorm(byte a);
void drawSnow();

float color = white;

void setup() {
  Serial.begin(115200);
  delay(500);
  Serial.print("Button_Press:");
  pinMode(0, INPUT);
  digitalWrite(0, HIGH);
  if (digitalRead(0) == 0) {
    Button_Press = 1;
    Serial.println(Button_Press);
  }
  else {
    Button_Press = 0;
    Serial.println(Button_Press);
  }

  /* pinMode(muxB, OUTPUT);
  pinMode(muxC, OUTPUT);
  pinMode(TFT_CS, OUTPUT);
  yield();
  pinMode(muxA, INPUT);
  pinMode(muxB, INPUT);
  pinMode(muxC, INPUT);
  pinMode(TFT_CS, INPUT);
  /*
  digitalWrite(muxA, 0);
  digitalWrite(muxB, 1);
  digitalWrite(muxC, 1);
```

```
  delay(20);

  if (digitalRead(TFT_CS) == 1) {
  //  Button_Press = 0;
  }
  else {
  //   Button_Press = 1;
  }
  */

  // Initialize serial interface and allow it to stablize

  //Serial.begin(115200);
  EEPROM.begin(512);
  delay(10);
  Serial.println();
  Serial.println();
  Serial.println("Startup");
  // read eeprom for ssid and pass
  Serial.println("Reading EEPROM ssid");
  String esid;
  for (int i = 0; i < 32; ++i)
  {
    esid += char(EEPROM.read(i));
  }
  Serial.print("SSID: ");
  Serial.println(esid);
  String epass = "";
  for (int i = 32; i < 96; ++i)
  {
    epass += char(EEPROM.read(i));
  }
  Serial.print("PASS: ");
  Serial.println(epass);

  char SSID[32];
  esid.toCharArray(SSID, 32); //Converts String from EEPROM to Char for WiFi function

  char PASSWORD[64];
  epass.toCharArray(PASSWORD, 64); //This reads the Password out of EEPROM and converts
to a Char

  /* String eChange = "";
```

```
 for (int i = 120; i < 122; ++i)
 {
   eChange += char(EEPROM.read(i));
 }
 Serial.print("Change ");
 Serial.println(eChange);
 //Button_Press = eChange.toInt();*/

delay(1000);
if (Button_Press == 1) {
  setupAP();
} else {
  Serial.print(F("Connecting to "));
  Serial.println(SSID);

  // Attempt WiFi network connection

  WiFi.begin(SSID, PASSWORD);
  //serial.println(GET)

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(F("."));
  }
  Serial.println();
  Serial.println(F("WiFi connected"));
  Serial.println(F("IP address: "));
  Serial.println(WiFi.localIP());



  // Setup callbacks for title and pubDate tags in RSS XML
  reader.setTitleCallback(&titleCallback);
  reader.setPubDateCallback(&pubDateCallback);

  // Initialize feed index to first entry in feed list
  String eaux = "";
  for (int i = 107; i < 110; ++i)
  {
    eaux += char(EEPROM.read(i));
  }
  Serial.print("AuxScreen: ");
  Serial.println(eaux);
```

```
    rssFeedIndex = eaux.toInt();
    String equote = "";
    for (int i = 110; i < 112; ++i)
    {
      equote += char(EEPROM.read(i));
    }
    Serial.print("Inspirational Quote: ");
    Serial.println(equote);
    QuoteIndex = equote.toInt();
    //rssFeedIndex = 0; //Read from EEPROM
    /////////
    //QuoteIndex= 0; //Read from EEPROM
    //Initialize Zip Code
    // HTTP request
  }
  pinMode(inputPin, INPUT);
  pinMode(muxA, OUTPUT);
  pinMode(muxB, OUTPUT);
  pinMode(muxC, OUTPUT);
  pinMode(TFT_CS, OUTPUT);
  digitalWrite(muxA, 0);
  digitalWrite(muxB, 1);
  digitalWrite(muxC, 0);
  yield();
  digitalWrite(muxA, 1);
  digitalWrite(muxB, 0);
  digitalWrite(muxC, 0);
  digitalWrite(TFT_CS, LOW);
  tft.initR(INITR_BLACKTAB);   // initialize a ST7735S chip, black tab
  tft.setRotation(135);
  tft.setTextWrap(false);
  tft.setTextSize(5);
  clearTFT();
  drawTime(timeNow, 1);
  delay(10);
  ESP.wdtFeed();
  yield();
  delay(500);
  ESP.wdtFeed();
  digitalWrite(TFT_CS, HIGH);
  yield();
  delay(50);
}
```

```
static char respBuf[4096];
float currentTime = 0;
void loop() {

  digitalWrite(muxA, HIGH);
  digitalWrite(muxB, HIGH);
  digitalWrite(muxC, HIGH);
  pinMode(TFT_CS, INPUT);
  val = digitalRead(TFT_CS);
  ESP.wdtFeed();

  if (Button_Press == 1) {
    server.handleClient();
  } else {
    ESP.wdtFeed();
    //const int httpPort = 80;
    // Short delay to keep networking happy

    int j = 0;
    for (j = 0; j < 1 ; j++) {
      delay(10);

      // Get URL of RSS feed to display
      const char *url = rssFeedURLs[rssFeedIndex];
      Serial.println(url);
      // Read and parse the RSS feed
      bool result = reader.read(url);
      Serial.println("j= ");
      Serial.println(j);
      delay(3000);
    }

    String WU_API_KEY = "2df07ee2d9774288";
    String ezip = "";
    for (int i = 96; i < 101; ++i)
    {
      ezip += char(EEPROM.read(i));
    }
    Serial.print("ZipCode: ");
    Serial.println(ezip);
    ESP.wdtFeed();
    String WU_LOCATION = ezip; //read from EEPROM
```

```
  String aa = "api.wunderground.com";
  String a = "GET /api/" + WU_API_KEY + "/conditions/q/" + WU_LOCATION + ".json
HTTP/1.1\r\n";
  String b = "User-Agent: ESP8266/0.1\r\n";
  String c = "Accept: */*\r\n";
  String d = "Host: " + aa + "\r\n";
  String e = "Connection: close\r\n";
  String f = "\r\n";
  String g = a + b + c + d + e + f;


  WiFiClient httpclient;
  const int httpPort = 80;
  if (!httpclient.connect(WUNDERGROUND, httpPort)) {
    //Serial.println(F("connection failed"));
    delay(DELAY_ERROR);
    return;
  }

  // This will send the http request to the server
  httpclient.print(g);
  httpclient.flush();
  Serial.print(F("Connecting to "));
  Serial.println(WUNDERGROUND);
  ESP.wdtFeed();
  //Inspirational Quotes part
  const char *quote = InspirationalQuotes[QuoteIndex];
  Serial.print("\n Inspirational Quote of the Day:  ");
  Serial.println(quote);
  yield();
  ESP.wdtFeed();
  pinMode(TFT_CS, OUTPUT);
  digitalWrite(muxA, 1);
  digitalWrite(muxB, 0);
  digitalWrite(muxC, 0);
  delay(20);
  ESP.wdtFeed();
  digitalWrite(TFT_CS, LOW);
  clearTFT();
  tft.setTextSize(7);
  tft.setTextColor(white, black);
  tft.setCursor(0, 45);
  String buff = "   Try, like we did.    ";
```

```
int i = 0;
for (i = 0; i < 39; i++) {
  tft.setCursor(0, 45);
  tft.print(buff.substring(i, i + 4));
  delay(3);
}
delay(500);
ESP.wdtFeed();
yield();
clearTFT();
tft.setTextSize(5);
drawTime(timeNow, 1);

delay(500);
clearTFT();
ESP.wdtFeed();
yield();
drawSun(1);
delay(500);
ESP.wdtFeed();
yield();
clearTFT();
drawCloud();
delay(500);
ESP.wdtFeed();
yield();
clearTFT();
drawRain(1);
delay(500);
ESP.wdtFeed();
yield();
clearTFT();
drawSnow();
delay(500);
ESP.wdtFeed();
yield();
clearTFT();
drawStorm(1);
digitalWrite(TFT_CS, HIGH);
ESP.wdtFeed();
yield();
Serial.print("\n \n");
ESP.wdtFeed();
```

```
//Collect http response headers and content from WEather Underground
// HTTP headers are discarded
// The content is formatted in JSON and is left in respBuf
int respLen = 0;
bool skip_headers = true;
while (httpclient.connected() || httpclient.available()) {
  if (skip_headers) {
    String aLine = httpclient.readStringUntil('\n');
    //Serial.println(aLine);
    //Serial.println("^aLine");
    // Blank line denotes end of headers
    if (aLine.length() <= 1) {
      skip_headers = false;
    }
  }
  else {
    int bytesIn;
    bytesIn = httpclient.read((uint8_t *)&respBuf[respLen], sizeof(respBuf) - respLen);
    //Serial.print(F("bytesIn ")); Serial.println(bytesIn);
    if (bytesIn > 0) {
      respLen += bytesIn;
      if (respLen > sizeof(respBuf)) respLen = sizeof(respBuf);
    }
    else if (bytesIn < 0) {
      //Serial.print(F("read error "));
      // Serial.println(bytesIn);
    }
  }
  delay(1);

}
httpclient.stop();

if (respLen >= sizeof(respBuf)) {
  Serial.print(F("respBuf overflow "));
  Serial.println(respLen);
  delay(DELAY_ERROR);
  return;
}
// Terminate the C string
respBuf[respLen++] = '\0';
//Serial.print(F("respLen "));
//Serial.println(respLen);
```

```
    //Serial.println(respBuf);
    if (showWeather(respBuf)) {
      delay(DELAY_NORMAL);
    }
    else {
      delay(DELAY_ERROR);
    }

    String qChange = "1";
    Serial.print("Change:");
    for (int i = 0; i < qChange.length(); ++i)
    {
      EEPROM.write(120 + i, qChange[i]);
      Serial.print("Wrote: ");
      Serial.println(qChange[i]);
    }
    EEPROM.commit();
  }



}

void titleCallback(char *titleStr) {
  yield();
  char buffer[TITLE_BUFFER_SIZE];
  ESP.wdtFeed();
  Serial.print("\nTitle: ");
  Serial.println(titleStr);
  // Make buffer empty
  buffer[0] = '\0';
  strcpy(buffer, " ");
  strcat(buffer, titleStr);
  strcat(buffer, "    ");
}


bool showWeather(char *json)
{
  StaticJsonBuffer<3 * 1024> jsonBuffer;

  // Skip characters until first '{' found
```

```
  // Ignore chunked length, if present
  char *jsonstart = strchr(json, '{');
  //Serial.print(F("jsonstart ")); Serial.println(jsonstart);
  if (jsonstart == NULL) {
    //Serial.println(F("JSON data missing"));
    return false;
  }
  json = jsonstart;
  ESP.wdtFeed();
  // Parse JSON
  JsonObject& root = jsonBuffer.parseObject(json);
  if (!root.success()) {
    //Serial.println(F("jsonBuffer.parseObject() failed"));
    return false;
  }
  yield();
  // Extract weather info from parsed JSON
  JsonObject& current = root["current_observation"];
  const char *temp_string = current["temperature_string"];
  Serial.println(temp_string);
  const char *weather = current["weather"];
  Serial.println(weather);
  const char *observation_time = current["local_time_rfc822"];
  Serial.println(observation_time);
  const char *place = current["display_location"]["full"];
  Serial.println(place);
  ESP.wdtFeed();
  yield();


  return true;
}

// Callback called every time a pubDate tag is found in the RSS XML
void pubDateCallback(char *dateStr) {
  Serial.println(dateStr);
}

void launchWeb(int webtype) {
  Serial.println("");
  //Serial.println("WiFi connected");
  //Serial.print("Local IP: ");
  //Serial.println(WiFi.localIP());
```

```
  Serial.print("SoftAP IP: ");
  Serial.println(WiFi.softAPIP());
  createWebServer(webtype);
  // Start the server
  server.begin();
  Serial.println("Server started");
}

void setupAP(void) {
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);
  int n = WiFi.scanNetworks();
  Serial.println("scan done");
  if (n == 0)
    Serial.println("no networks found");
  else
  {
    Serial.print(n);
    Serial.println(" networks found");
    for (int i = 0; i < n; ++i)
    {
      // Print SSID and RSSI for each network found
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.print(")");
      Serial.println((WiFi.encryptionType(i) == ENC_TYPE_NONE) ? " " : "*");
      delay(10);
    }
  }
  Serial.println("");
  st = "<ol>";
  for (int i = 0; i < n; ++i)
  {
    // Print SSID and RSSI for each network found
    st += "<li>";
    st += WiFi.SSID(i);
    st += " (";
    st += WiFi.RSSI(i);
    st += ")";
```

```
    st += (WiFi.encryptionType(i) == ENC_TYPE_NONE) ? " " : "*";
    st += "</li>";
  }
  st += "</ol>";
  delay(100);
  WiFi.softAP(ssid, passphrase, 6);
  Serial.println("softap");
  launchWeb(1);
  Serial.println("over");
}

void createWebServer(int webtype)
{
  if ( webtype == 1 ) {
    server.on("/", []() {
      IPAddress ip = WiFi.softAPIP();
      String ipStr = String(ip[0]) + '.' + String(ip[1]) + '.' + String(ip[2]) + '.' + String(ip[3]);
      content = "<!DOCTYPE HTML>\r\n<html>Hello from ESP8266 at ";
      content += ipStr;
      content += "<p>";
      content += st;
      content += "</p><form method='get' action='setting'><label>SSID: </label><input
name='ssid' length=32><label>Password: </label><input name='pass'
length=64><br><label>ZipCode: </label><input name='ZipCode'
length=10><label>AuxScreen:</label><select name='AuxScreen'><option value='4'>AP
Science</option><option value='6'>Reuters</option><option value='2'>NPR</option><option
value='5'>LA Times</option><option value='0'>Yahoo Sports</option><option value='1'>Yahoo
Sports MLB</option><option value='3'>BBC</option><option value='7'>USA
Today</option></select><br><label>Inspirational Quote:</label><select
name='ZipCode2'><option value='0'>0</option></select><br><input type='submit'></form>";
      content += "</html>";
      server.send(200, "text/html", content);
    });
    server.on("/setting", []() {
      String qsid = server.arg("ssid");
      String qpass = server.arg("pass");
      String qZip = server.arg("ZipCode");
      String qAux = server.arg("AuxScreen");
      String qZip2 = server.arg("ZipCode2");
      if (qsid.length() > 0 && qpass.length() > 0 && qZip.length() > 0  && qAux.length() > 0  &&
qZip2.length() > 0) {
        Serial.println("clearing eeprom");
        for (int i = 0; i < 128; ++i) {
```

```
  EEPROM.write(i, 0);
}
Serial.println(qsid);
Serial.println("");
Serial.println(qpass);
Serial.println("");
Serial.println(qZip);
Serial.println("");
Serial.println(qAux);
Serial.println("");
Serial.println(qZip2);
Serial.println("");

Serial.println("writing eeprom ssid:");
for (int i = 0; i < qsid.length(); ++i)
{
  EEPROM.write(i, qsid[i]);
  Serial.print("Wrote: ");
  Serial.println(qsid[i]);
}
Serial.println("writing eeprom pass:");
for (int i = 0; i < qpass.length(); ++i)
{
  EEPROM.write(32 + i, qpass[i]);
  Serial.print("Wrote: ");
  Serial.println(qpass[i]);
}
Serial.println("writing eeprom ZipCode:");
for (int i = 0; i < qZip.length(); ++i)
{
  EEPROM.write(96 + i, qZip[i]);
  Serial.print("Wrote: ");
  Serial.println(qZip[i]);
}
Serial.println("writing eeprom AuxScreen:");
for (int i = 0; i < qAux.length(); ++i)
{
  EEPROM.write(107 + i, qAux[i]);
  Serial.print("Wrote: ");
  Serial.println(qAux[i]);
}
Serial.println("writing eeprom ZipCode2:");
for (int i = 0; i < qZip2.length(); ++i)
```

```
      {
        EEPROM.write(110 + i, qZip2[i]);
        Serial.print("Wrote: ");
        Serial.println(qZip2[i]);
      }
      String qChange = "0";
      Serial.print("Change:");
      for (int i = 0; i < qChange.length(); ++i)
      {
        EEPROM.write(120 + i, qChange[i]);
        Serial.print("Wrote: ");
        Serial.println(qChange[i]);
      }
      EEPROM.commit();
      content = "{\"Success\":\"saved to eeprom... reset to boot into new wifi\"}";
      statusCode = 200;
    } else {
      content = "{\"Error\":\"404 not found\"}";
      statusCode = 404;
      Serial.println("Sending 404");
    }
    server.send(statusCode, "application/json", content);
  });
 } else if (webtype == 0) {
  server.on("/", []() {
    IPAddress ip = WiFi.localIP();
    String ipStr = String(ip[0]) + '.' + String(ip[1]) + '.' + String(ip[2]) + '.' + String(ip[3]);
    server.send(200, "application/json", "{\"IP\":\"" + ipStr + "\"}");
  });
  server.on("/cleareeprom", []() {
    content = "<!DOCTYPE HTML>\r\n<html>";
    content += "<p>Clearing the EEPROM</p></html>";
    server.send(200, "text/html", content);
    Serial.println("clearing eeprom");
    for (int i = 0; i < 128; ++i) {
      EEPROM.write(i, 0);
    }
    EEPROM.commit();
  });
 }
}

void drawTime(float a, byte b) {
```

```
  char buff[6];
  if (b == 1) {
    tft.setTextSize(5);
    tft.setCursor(5, 45);
    if (a < 1000) {
      dtostrf(a, 3, 0, buff);
      sprintf(buff, "0%c:%c%c", buff[0], buff[1], buff[2]);
    }
    else {
      dtostrf(a, 4, 0, buff);
      sprintf(buff, "%c%c:%c%c", buff[0], buff[1], buff[2], buff[3]);
    }
  }
  else {
    tft.setTextSize(7);
    tft.setCursor(0, 45);
    dtostrf(a, 3, 0, buff);
    sprintf(buff, "%c:%c%c", buff[0], buff[1], buff[2]);
  }
  tft.print(buff);
}

void drawSnow() {
  int i;
  tft.fillRoundRect(10, 40, 140, 50, 25, color);
  tft.fillCircle(100, 40, 35, color);
  tft.fillCircle(55, 35, 20, color);
  for (i = 0; i < 6; i++) {
    tft.drawLine(20 + i * 25, 100, 24 + i * 25, 104, color);
    tft.drawLine(24 + i * 25, 100, 20 + i * 25, 104, color);
    tft.drawLine(22 + i * 25, 100, 22 + i * 25, 104, color);
    tft.drawLine(20 + i * 25, 102, 24 + i * 25, 102, color);
    tft.drawLine(15 + i * 25, 120, 19 + i * 25, 124, color);
    tft.drawLine(19 + i * 25, 120, 15 + i * 25, 124, color);
    tft.drawLine(17 + i * 25, 120, 17 + i * 25, 124, color);
    tft.drawLine(15 + i * 25, 122, 19 + i * 25, 122, color);
    tft.drawLine(5 + i * 25, 110, 9 + i * 25, 114, color);
    tft.drawLine(9 + i * 25, 110, 5 + i * 25, 114, color);
    tft.drawLine(7 + i * 25, 110, 7 + i * 25, 114, color);
    tft.drawLine(5 + i * 25, 112, 9 + i * 25, 112, color);
  }
}
```

```
void drawStorm(byte a) {
  int i;
  if (a == 1) {
    drawRain(1);
  }
  else {
    drawRain(0);
  }
  tft.drawLine(15 + 2 * 25, 100, 10 + 2 * 25, 105, black);
  tft.drawLine(15 + 3 * 25, 100, 10 + 3 * 25, 105, black);
  tft.drawLine(16 + 3 * 25, 100, 11 + 3 * 25, 105, black);
  tft.drawLine(10 + 2 * 25, 120, 5 + 2 * 25, 125, black);
  tft.drawLine(11 + 2 * 25, 120, 6 + 2 * 25, 125, black);
  if (a == 1) {
    color = yellow;
  }
  for (i = 0; i < 15; i++) {
    tft.drawLine(90 - i, 90, 75 - i, 105, color);
  }
  for (i = 0; i < 13; i++) {
    if (i < 6) {
      tft.drawLine(74 + i, 104, 60, 125, color);
    }
    else {
      tft.drawLine(74 + i, 104, 61, 125, color);
    }
  }
  color = white;
}

void drawRain(byte a) {
  int i;
  tft.fillRoundRect(10, 40, 140, 50, 25, color);
  tft.fillCircle(100, 40, 35, color);
  tft.fillCircle(55, 35, 20, color);
  if (a == 1) {
    color = cyan;
  }
  for (i = 0; i < 6; i++) {
    tft.drawLine(15 + i * 25, 100, 10 + i * 25, 105, color);
    tft.drawLine(16 + i * 25, 100, 11 + i * 25, 105, color);
    tft.drawLine(10 + i * 25, 120, 5 + i * 25, 125, color);
    tft.drawLine(11 + i * 25, 120, 6 + i * 25, 125, color);
```

```
    if (i < 5) {
      tft.drawLine(28 + i * 25, 110, 23 + i * 25, 115, color);
      tft.drawLine(27 + i * 25, 110, 22 + i * 25, 115, color);
    }
  }
  color = white;
}

void drawCloud() {
  tft.fillRoundRect(10, 55, 140, 50, 25, color);
  tft.fillCircle(100, 55, 35, color);
  tft.fillCircle(55, 50, 20, color);
}

void drawSun(byte a) {
  if (a == 1) {
    color = yellow;
  }
  tft.setCursor(0, 0);
  int k;
  tft.fillCircle(80, 64, 40, color);
  for (k = 40; k < 60; k = k + 4) {
    tft.fillCircle(80, 64 + k, 5, color);
    tft.fillCircle(80, 64 - k, 5, color);
    tft.fillCircle(80 + k, 64, 5, color);
    tft.fillCircle(80 - k, 64, 5, color);
    tft.fillCircle(.8 * 80 + k, .8 * 64 + k, 5, color);
    tft.fillCircle(1.2 * 80 - k, .8 * 64 + k, 5, color);
    tft.fillCircle(.8 * 80 + k, 1.2 * 64 - k, 5, color);
    tft.fillCircle(1.2 * 80 - k, 1.2 * 64 - k, 5, color);
  }
  color = white;
}

void clearTFT() {
  tft.fillScreen(ST7735_BLACK);
}
```