Temperature Sensor Test Plan

(http://kdobrien.github.io/Projects/EnvironmentalMonitor/)

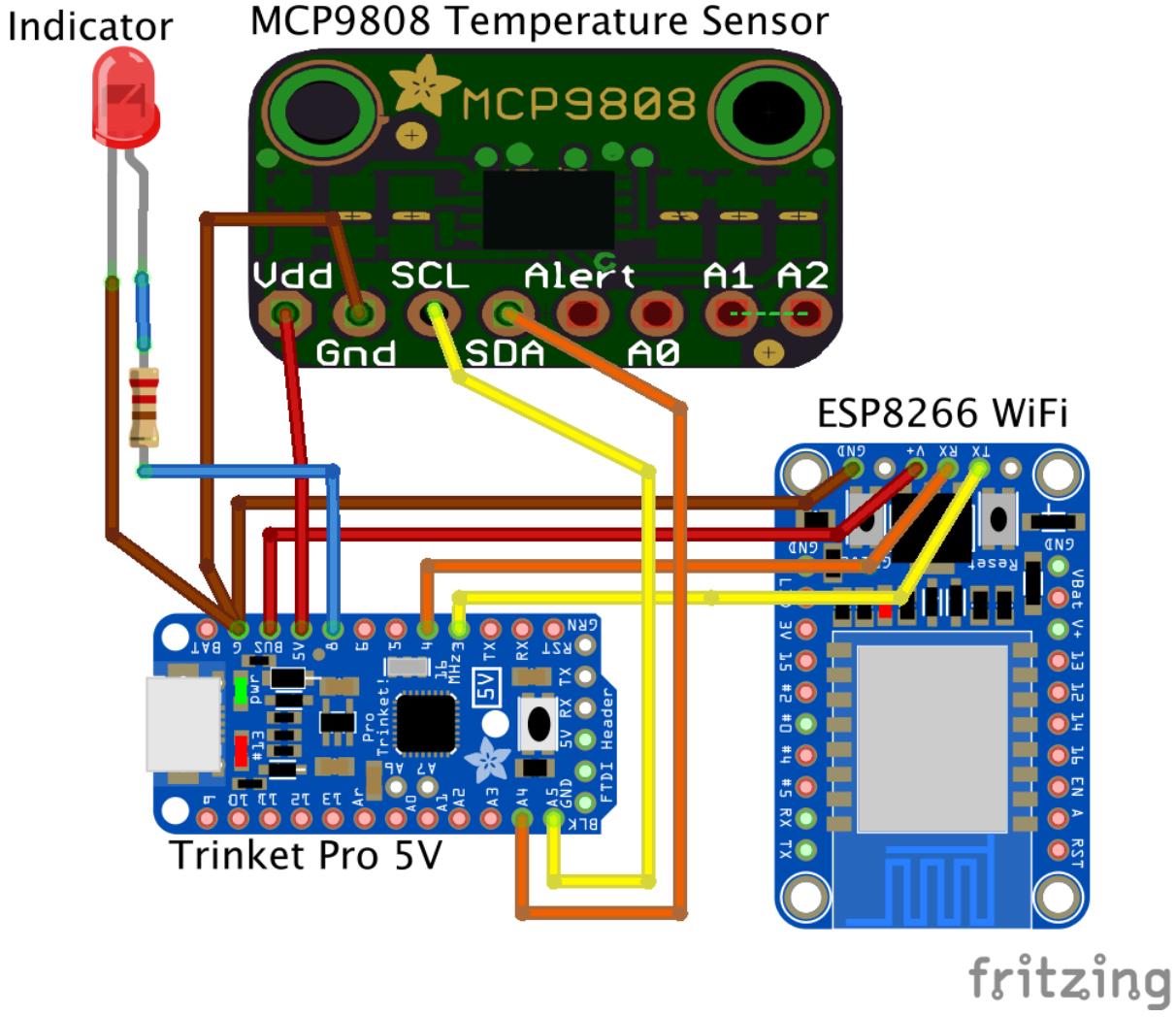
Service environment temperature sensor

The temperature sensor is made up of four components as shown below.

The MCP9808 can use the Arduino (Trinket Pro 5V) power supply since it draws very little current. The ESP8266 ***must*** use the USB 5V (BUS) supply since it can consume up to 500 mA at times which is beyond the ability of the Trinket Pro 5V power supply output.

The ESP8266 runs at 3.3V. The breakout board has an on-board 3.3V regulator and 5V compatible Tx and Rx pins so it can be used with either a 3.3V or 5V Arduino.

The sensor will be assembled using wiring harnesses between each of the components rather than a single PC board.



Component Test

This section defines the procedure for verifying each of the components. The process involves verifying the Trinket Pro 5V, then testing the MCP9808 with the Trinket Pro 5V, testing the ESP8266 WiFi both standalone and connected to the Trinket Pro 5V and finally testing the indicator led.

Do not connect any of the components to the Trinket Pro 5V until directed to make the connections. This will keep components that are not being tested from interfering with the test for other components.

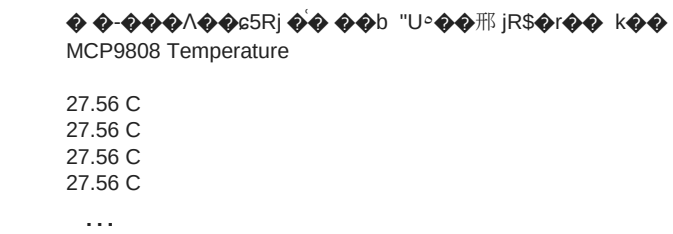
Trinket Pro 5V

- Configure the Arduino IDE to support the Trinket Pro 5V using the instructions at https://learn.adafruit.com/introducing-pro-trinket. Note that the Trinket Pro 5V uses the same chip as the Arduino UNO and is compatible.
- Connect an FTDI cable (USB to serial) between the computer and the Trinket Pro 5V. Note that this will power the board and you do *not* need to connect USB when the FTDI cable is present.
- Upload the Blink sketch from the Arduino IDE examples and verify that the led on the Trinket Pro 5V led blinks as expected.

MCP9808 Temperature Sensor Breakout Board

- Connect +V, Gnd, SCL and SDA from the MCP9808 board to the Trinket Pro 5V as shown in the diagram above.
- Run the I2CScanner (http://playground.arduino.cc/Main/I2cScanner) sketch after opening the Serial Monitor in the Arduino IDE. This will show all addresses that respond on the I2C bus and is a valuable way to check basic connections.
    - Verify that the serial monitor shows address *0x18* is found. This is the base address for the MCP9808.
- Run the *MCP9808 Test Code* sketch (see Appendix A) to verify that the MCP9808 can measure temperature. The serial monitor will show the measured temperature (see below example).

  Note that there may be some unprintable characters shown on the first line since the serial port is being used for both programming and serial output. The program actually starts at the *MCP9808 Temperature* line.

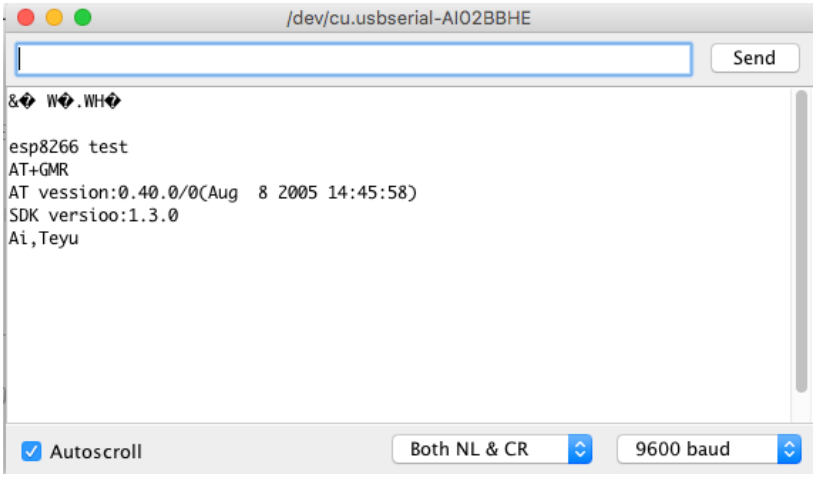
ESP8266 WiFi Breakout Board

Note: For this project, the ESP8266 must be loaded with the original ESP8266 firmware. The original firmware provides an API that is based on AT-style commands that makes the ESP8266 look like a serial modem. Instructions for flashing the firmware can be found at http://www.electrodragon.com/w/ESP8266_firmware_flasher. The firmware used for this project is version 1.3.0.2 which is found at http://www.electrodragon.com/w/ESP8266_AT_Commands.

- **ESP8266 Optional stand-alone check:**

  Use a terminal emulator like Cool Terminal (http://freeware.the-meiers.org/) to connect to the ESP8266 breakout board using the FTDI cable. You should be able to send *AT+GMR* to get the revision of the firmware sent to the terminal. Note that the baud rate is 115200 for the default firmware.

- **ESP8266 Connection to Arduino**
  - Connect the ESP8266 module to the Trinket Pro 5V as shown in the diagram above.
  - Run the *ESP8266 Arduino Connection Test* sketch found in Appendix A to test the connection to the ESP8266.
    - Set the serial monitor for a baud rate of 9600 and enable *Both NL & CR* (new line plus carriage return) for the terminal.
    - Send the characters *AT+GMR* to check the version of firmware. You should get a response similar to the figure below. Don't worry if there are some unprintable characters in the lines after *AT+GMR*.



- **Reset Baud Rate**
  - Using the same setup as the previous step, send the command *AT+CIOBAUD=9600*. This will change the ESP8266 baud rate from 115200 to 9600 which is more reliable for the SoftwareSerial port.
  - Run the *ESP8266 Arduino Connection Test* sketch found in Appendix A to test the changed baud rate. You should be able to send the *AT+GMR* string and see the firmware version with no unprintable characters.

Note: If you make a mistake and can no longer communicate with the ESP8266 due to an incorrect baud rate, you will need to re-flash the firmware. The flash process uses the baud rate 115200 automatically and the original baud rate will be restored on the ESP8266.

- **ESP8266 Internet Connection Test**
  - Use this step to join a WiFi network. Use the setup from the previous step.
    - Send ***AT+CWLAP*** to list all available WiFi access points. One or more access points should be listed in the Arduino IDE serial monitor.

    - Send ***AT+CWJAP="yourSSID","yourPassword"*** to join a network – use the proper SSID and password for your network and make sure the double quote (“) characters are included.

      You should get the following two lines for a successful connect.

      WIFI CONNECTED
      WIFI GOT IP

    - Send ***AT+CIFSR*** to show your IP address after a successful join. You should get the following two lines with values assigned by your network in the quotes.

      +CIFSR:STAIP,"192.168.1.233"
      +CIFSR:STAMAC,"18:fe:34:db:52:eb"

Indicator Test

Connect the indicator (led) as shown in the diagram above. There should be a 330 Ohm resistor between the Trinket Pro 5V and the LED as shown in the diagram.

Run the following sketch and verify the indicator blinks at a rate of once per second.

```
byte pin = 8;


void setup() {
 pinMode(pin,OUTPUT);

}

void loop() {
  digitalWrite(pin, HIGH);
  delay(500);
  digitalWrite(pin, LOW);
  delay(500);
}
```

Integration Test

Once all components have been tested, the application software (see web site) must be installed.

The following process will test the sensor along with the interactions to external servers.

Sensor Startup

Case 1: Network not available/cannot be opened

- Turn off the WiFi network router or configure the installed software to use the wrong (non-existent) network.
    - When plugged in (start without power), the sensor should blink the led slowly at first and then quickly to show that a connection was not made.

Case 2: Network available/autoconfig server

- **No autoconfig server available**
    - Disable the autoconfig server or configure the installed software to use a different or non-existent host
    - When plugged in (start without power), the led should blink slowly and then blink quickly to indicate configuration failed.
- **Autoconfig server available, no configuration match**
    - Remove or disable the autoconfig record for the sensor on the autoconfig server
    - When plugged in (start without power), the led should blink slowly and then blink quickly to indicate configuration failed.
    - The autoconfig server log should show a configuration attempt from the sensor with no match.
- **Autoconfig server available, configuration match**
    - Add or enable the autoconfig record for the sensor on the autoconfig server.
    - When plugged in (start without power), the led should blink slowly and then turn off.
    - The autoconfig server log should show a configuration attempt from the sensor with a successful match.

Case 3: Measure Temperature

- **Environmental Data Collector Functioning**.
    - Verify that the configured environmental data collector receives a measurement from the sensor within 3 minutes of powering on the sensor.
- **Unavailable Environmental Data Collector**
    - Disable the Environmental Data Collector for a period of 15 minutes
    - After the 10 minutes have expired, enable the Environmental Data Collector
    - At least 3 measurements (the cached data on the sensor) should arrive on the Environmental Data Collector within 3 minutes of being re-enabled.

Appendix A: Code for Component Test

MCP9808 Test Code (get readings)

```
#include <Wire.h>

uint8_t mcpAddr = 0x18;  // MCP9808 I2C address

void setup()
{
  Wire.begin();
  Serial.begin(9600);
  Serial.println("\nMCP9808 Temperature");
  Serial.println("");
}

double readMCP9808(byte address) // Return temperature in degrees C
{
  // Data sheet defines temp as 12 bit number.  Upper byte uses 4 bits and
  // all eight bits of the lower byte are used.
  //    Ta = (upperByte & 0x0F) * 16 + lowerByte / 16        for temp >= 0
  //    Ta = 256 - (upperByte & 0x0F) * 16 + lowerByte / 16  for temp < 0
  //
  //    (upperByte & 0x0F) * 16 = ((upperByte << 8) & 0x0F00) / 16 since << 8 == *256
  //    so we get Ta = ((upperByte << 8) * 0x0F00) / 16 + lowerByte / 16
  //    which is     = ((upperByte << 8 | lowerByte) & 0x0FFF) / 16
  uint16_t temp;
  float reading;

  Wire.beginTransmission(address);
  Wire.write(0x05); // index the temperature register
  Wire.endTransmission();

  Wire.requestFrom(address, 2);
  temp = Wire.read() << 8; // Get bits 15..8 of the register
  temp |= Wire.read();     // Get bits  7..0 of the register

  reading = (temp & 0x0FFF) / 16.0; // Get rid of first 4 bits, then divide by 16
  if ( temp & 0x1000) // Negative temperature?
    reading = 256 - reading;

  return reading;
}

void loop()
{
  Serial.print(readMCP9808(mcpAddr));
  Serial.println(" C");
  delay(2000);  // wait for next meas
}
```

ESP8266 Arduino Connection Test

```
#include <SoftwareSerial.h>

SoftwareSerial esp8266(3,4); // RX, TX for Pro Trinket

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial)
    ; // wait for serial port to connect. Needed for native USB port only
  Serial.println("\n\nesp8266 test");

  esp8266.begin(115200); // First time.
  //esp8266.begin(9600); // STEP 2 after reset of baud rate (AT+CIOBAUD=9600)
}

void loop() { // echo each serial port to the other for pass-through
  if (Serial.available()) {
    esp8266.write(Serial.read());
  }
  if (esp8266.available()) {
    Serial.write(esp8266.read());
  }
}
```

ESP8266 Reset Baud Rate Test

```
#include <SoftwareSerial.h>

SoftwareSerial esp8266(3,4); // RX, TX for Pro Trinket

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial)
    ; // wait for serial port to connect. Needed for native USB port only
  Serial.println("\n\nesp8266 test");

  //esp8266.begin(115200); // First time.
  esp8266.begin(9600); // STEP 2 after reset of baud rate (AT+CIOBAUD=9600)
}

void loop() { // echo each serial port to the other for pass-through
  if (Serial.available()) {
    esp8266.write(Serial.read());
  }
  if (esp8266.available()) {
    Serial.write(esp8266.read());
  }
}
```