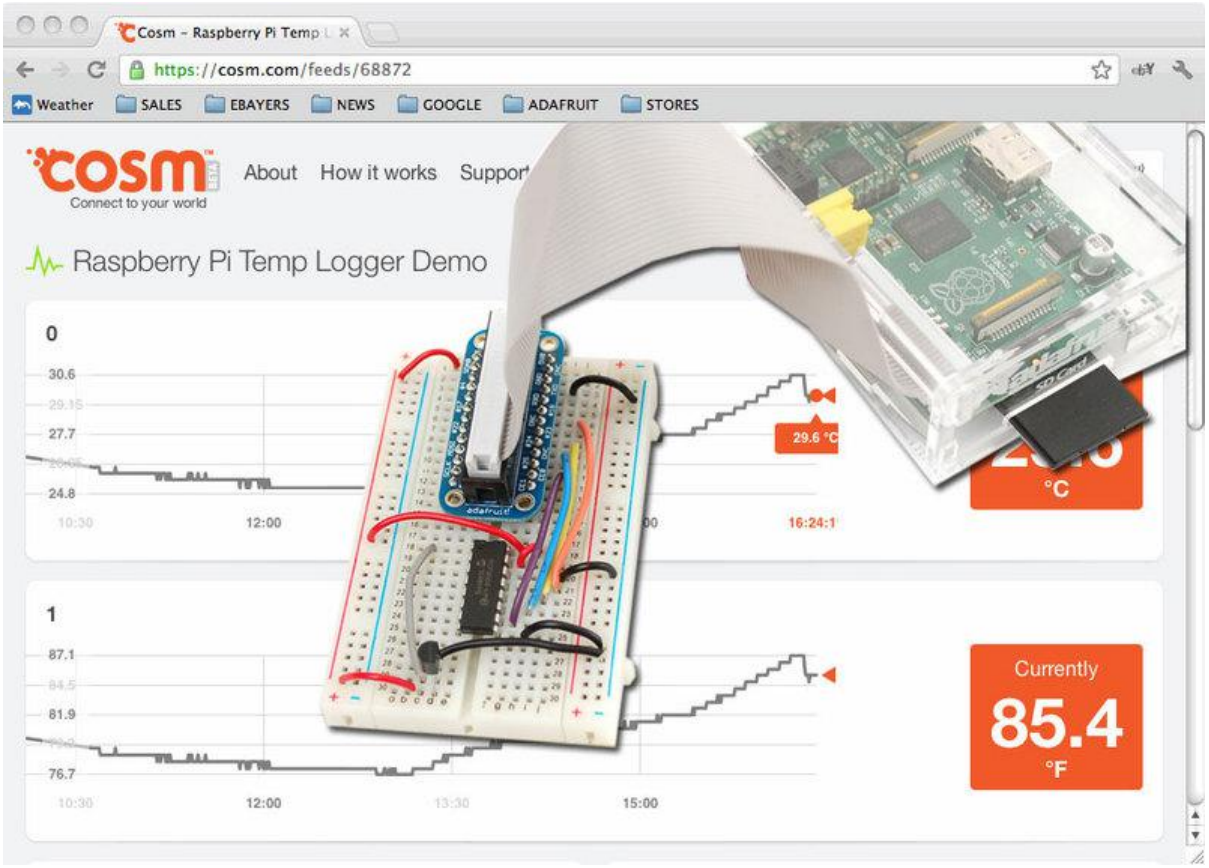
Send Raspberry Pi Data to COSM

Created by Michael Sklar



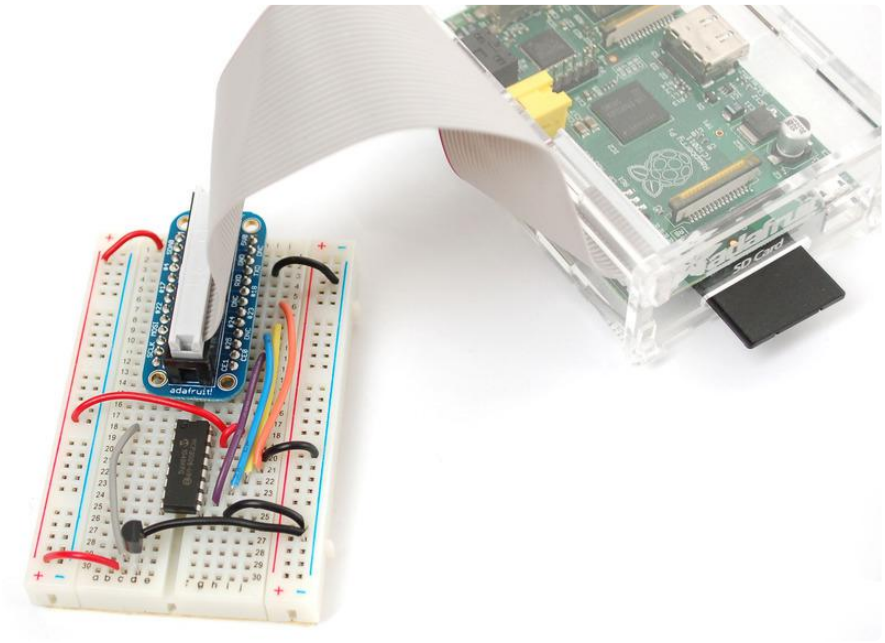
https://learn.adafruit.com/send-raspberry-pi-data-to-cosm

Last updated on 2023-08-29 02:10:15 PM EDT

Table of Contents

Overview

Please Note: Xively no longer has free developer access to their system, so this tutorial is only for historical research. Please check out our other IoT tutorials for alternative services!



The combination of connecting a Raspberry Pi to COSM makes creating a internet of things much easier than it has been in the past. The Pi with it's easy access to ethernet / WiFi and COSM's drop dead simple usability will graph all sensor data you send to it.

This tutorial explains how to connect a analog temperature sensor to the Pi and use a small python script to upload that data for storage and graphing on COSM.

To follow this tutorial you will need

- MCP3008 DIP-package ADC converter chip (http://adafru.it/856)
- Analog Temperature Sensor TMP-36 (http://adafru.it/165)
- Adafruit Pi Cobbler (http://adafru.it/914) - follow the tutorial to assemble it
- Half (http://adafru.it/64) or Full-size breadboard (http://adafru.it/239)
- Breadboarding wires ()
- Raspberry Pi with a internet connection

Hey, that photo up there has the GPIO cable in backwards - so when you wire it up don't follow that pic!

Connecting the Cobbler to the MCP3008 and TMP36

Please Note: Xively no longer has free developer access to their system, so this tutorial is only for historical research. Please check out our other IoT tutorials for alternative services!

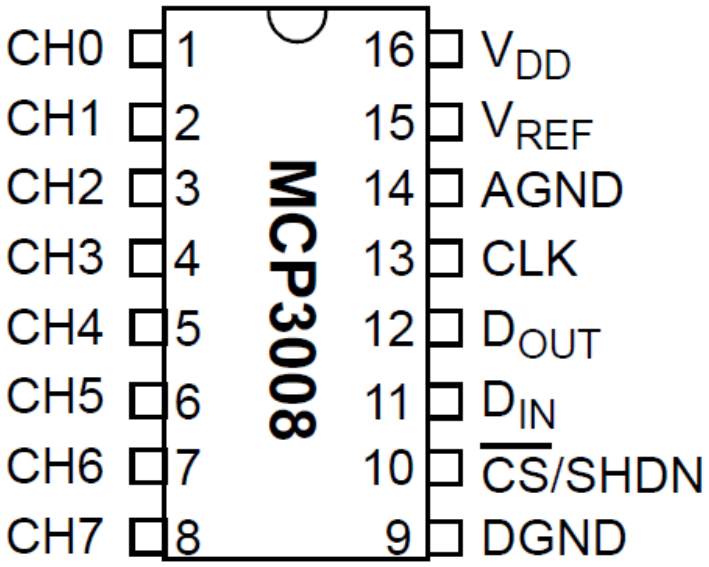
Why we need an ADC

The Raspberry Pi computer does not have a way to read analog inputs. It's a digital-only computer. Compare this to the Arduino, AVR or PIC microcontrollers that often have 6 or more analog inputs! Analog inputs are handy because many sensors are analog outputs, so we need a way to make the Pi analog-friendly.

We'll do that by wiring up an MCP3008 chip (http://adafru.it/856) to it. The MCP3008 (http://adafru.it/856) acts like a 'bridge' between digital and analog. It has 8 analog inputs and the Pi can query it using 4 digital pins. That makes it a perfect addition to the Pi for integrating simple sensors like photocells (), FSRs () or potentiometers, ther

mistors (), etc.!

Lets check the datasheet of the MCP3008 chip. () On the first page in the lower right corner there's a pinout diagram showing the names of the pins.



Wiring Diagram

In order to read analog data we need to use the following pins: VDD (power), DGND (digital ground) to power the MCP3008 chip. We also need four 'SPI' data pins: DOUT (Data Out from MCP3008), CLK (Clock pin), DIN (Data In from Raspberry Pi), and /CS (Chip Select). Finally of course, a source of analog data, we'll be using the TMP36 temperature sensor

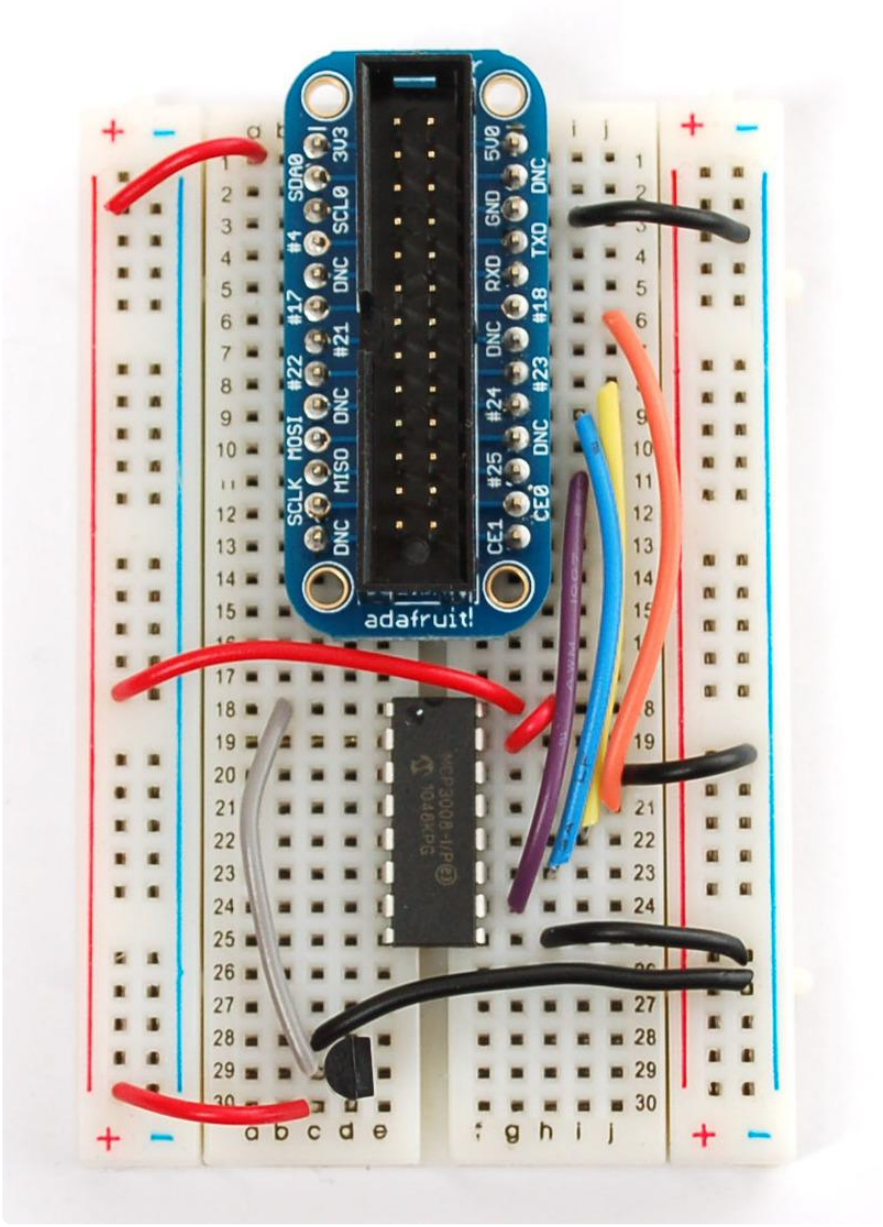
The MCP3008 has a few more pins we need to connect: AGND (analog ground, used sometimes in precision circuitry, which this is not) connects to GND, and VREF (analog voltage reference, used for changing the 'scale' - we want the full scale so tie it to 3.3 V)

Below is a wiring diagram. Connect the 3.3V cobbler pin to the left + rail and the GND pin to the right - rail. Connect the following pins for the MCP chip

- MCP3008 VDD -> 3.3V (red)
- MCP3008 VREF -> 3.3V (red)
- MCP3008 AGND -> GND (green)
- MCP3008 CLK -> #18
- MCP3008 DOUT -> #23

- MCP3008 DIN -> #24
- MCP3008 CS -> #25
- MCP3008 DGND -> GND (green)

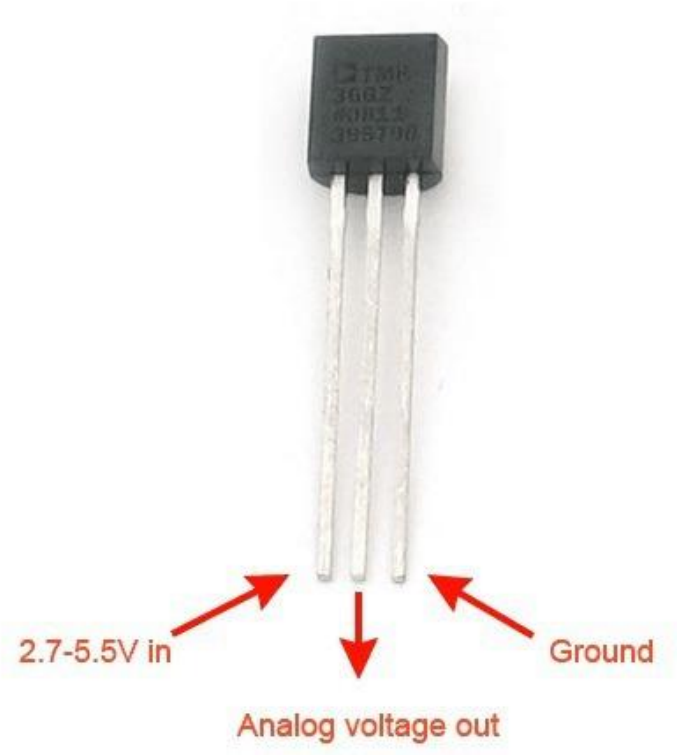
Advanced users may note that the Raspberry Pi does have a hardware SPI interface (the cobbler pins are labeled MISO/MOSI/SCLK/CE0/CE1). The hardware SPI interface is super fast but not included in all distributions. For that reason we are using a bit banged SPI implementation so the SPI pins can be any of the raspberry pi's GPIOs (assuming you update the script). Once you get this project working with the above pinout, feel free to edit the python code to change the pins as you'd like to have them!



TMP36

Finally the TMP36 has three pins that need to be connected. They are numbered from left to right in ascending order when the text of the sensor is facing you.

- pin1: 3.3v
- pin2: analog out --> channel0 on mcp3008 (pin1)
- pin3: gnd



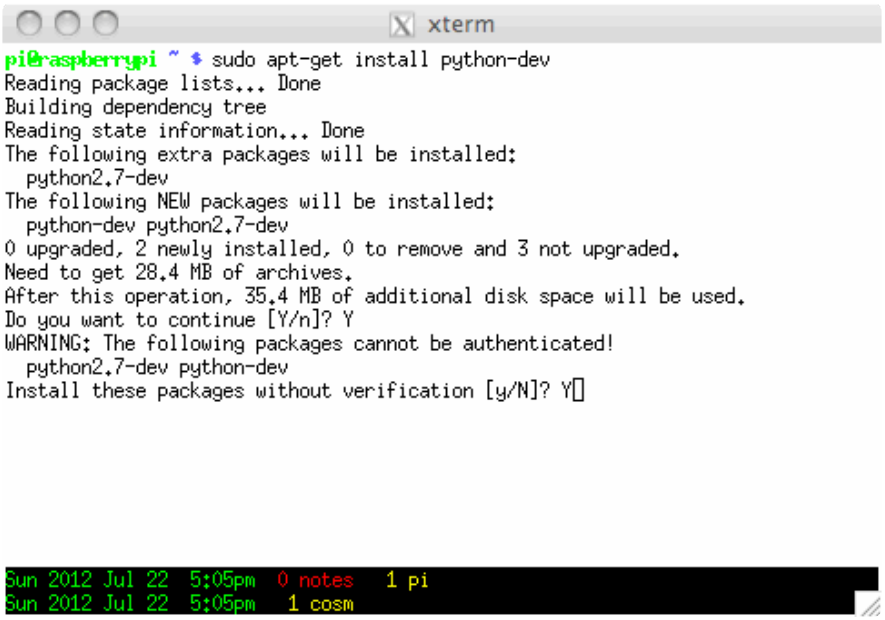
Necessary Packages

Please Note: Xively no longer has free developer access to their system, so this tutorial is only for historical research. Please check out our other IoT tutorials for alternative services!

This guide is based on Debian's "Wheezy" release for Raspberry Pi. It was made available in Mid July 2012. The following items must be installed in order to utilize the Raspberry Pi's GPIO pins and to upload data to COSM.

Add the latest dev packages for Python (2.x)

```
sudo apt-get install python-dev
```

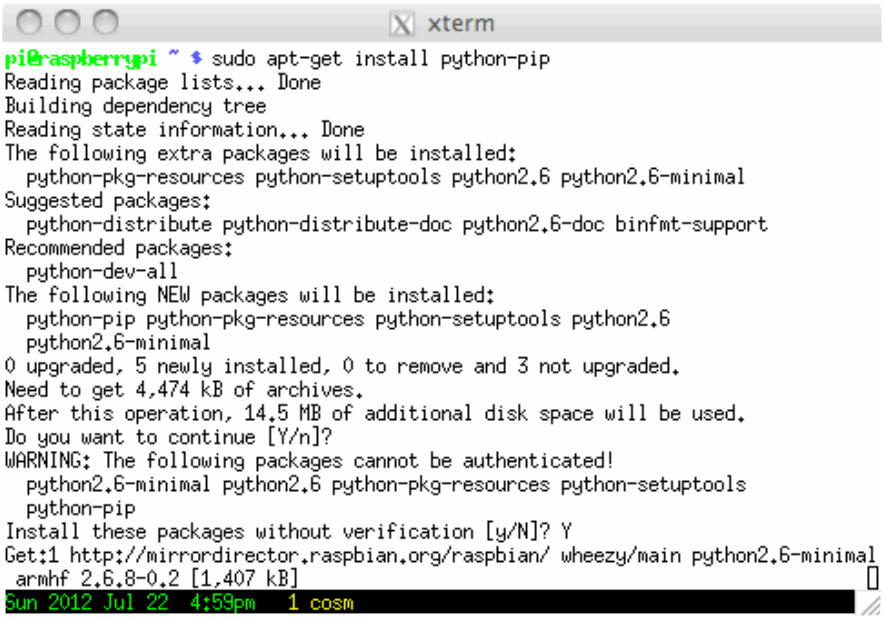


Upgrade distribute (required for RPi.GPIO 0.3.1a) - [No image for this one]

```
sudo easy_install -U distribute
```

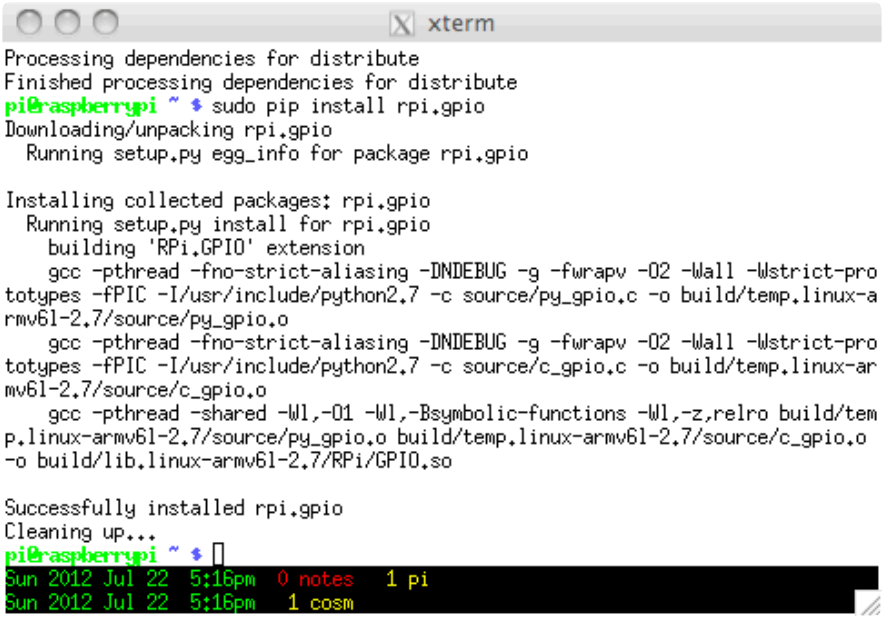
Install python-pip (Pip Installs Packages, python packages)

```
sudo apt-get install python-pip
```



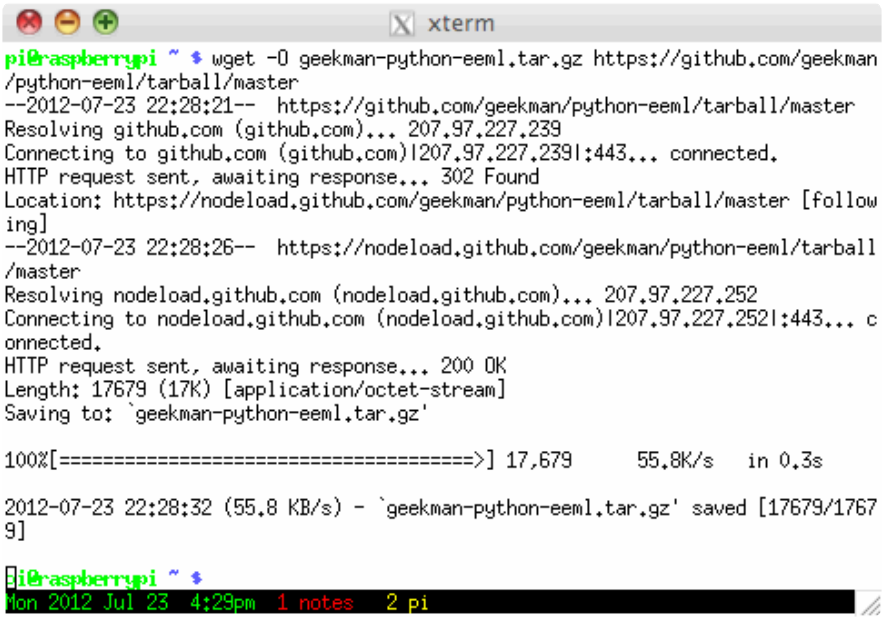
Install rpi.gpio (0.3.1a) or later

```
sudo pip install rpi.gpio
```



Download EEML - markup language COSM accepts

```
wget -O geekman-python-eeml.tar.gz https://github.com/geekman/python-eeml/tarball/master
```

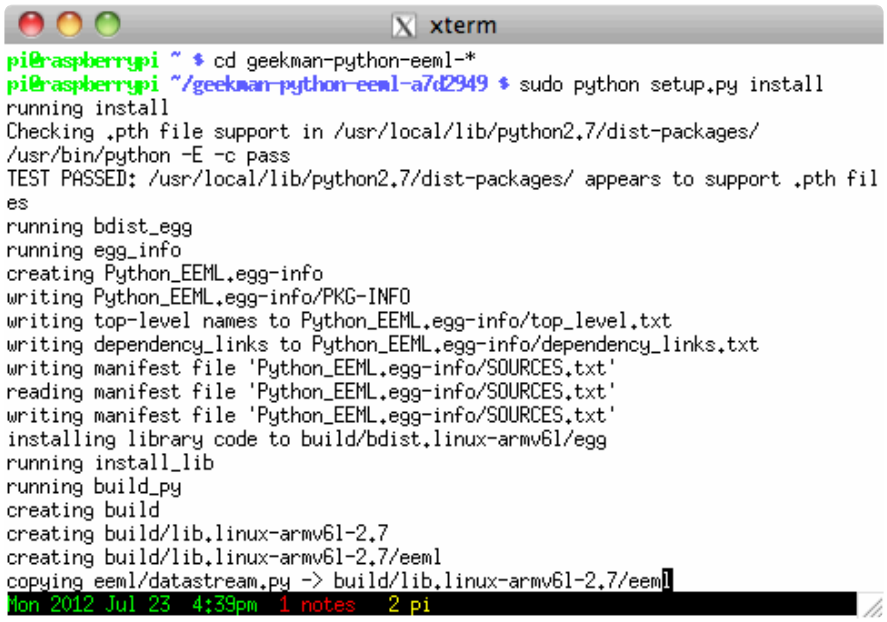


Extract the EEML tarball

```
tar zxvf geekman-python-eeml.tar.gz
```

Change into the directory and install the EEML python package

```
cd geekman-python-eeml*
sudo python setup.py install
```



COSM Account and Feed

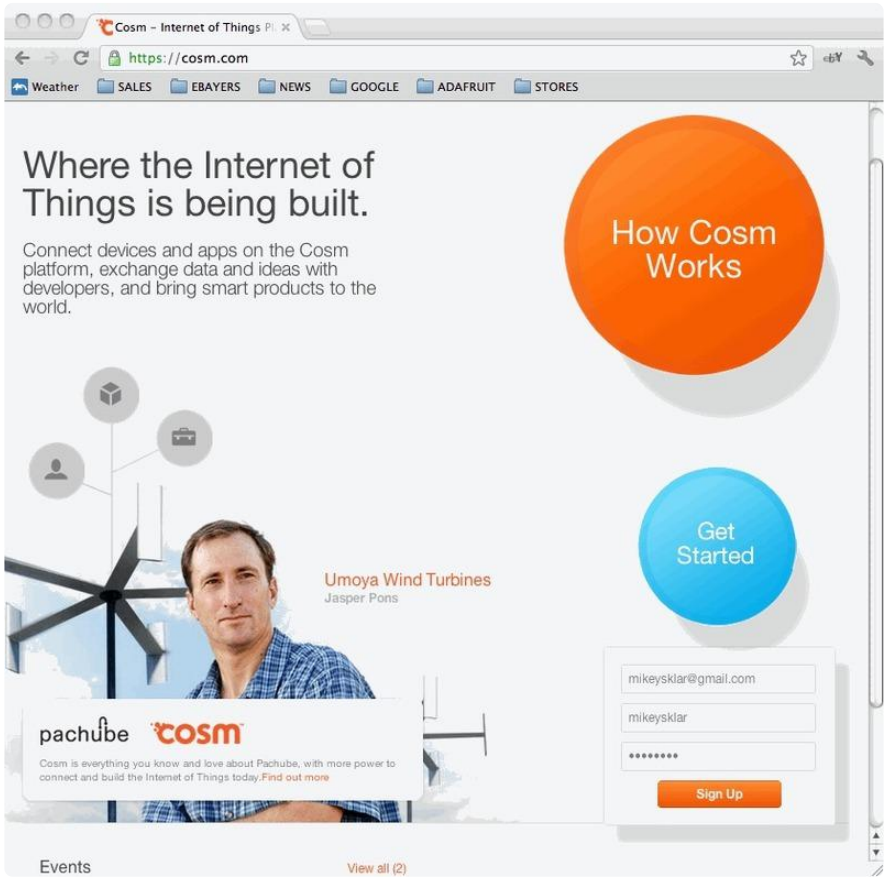
Please Note: Xively no longer has free developer access to their system, so this tutorial is only for historical research. Please check out our other IoT tutorials for alternative services!

COSM (used to be Pachube) helps connect little devices like the raspberry pi to the internet. You will need to do the following to use COSM.

- Setup a Account
- Create a Feed
- Save the API_KEY
- Save the FEED ID

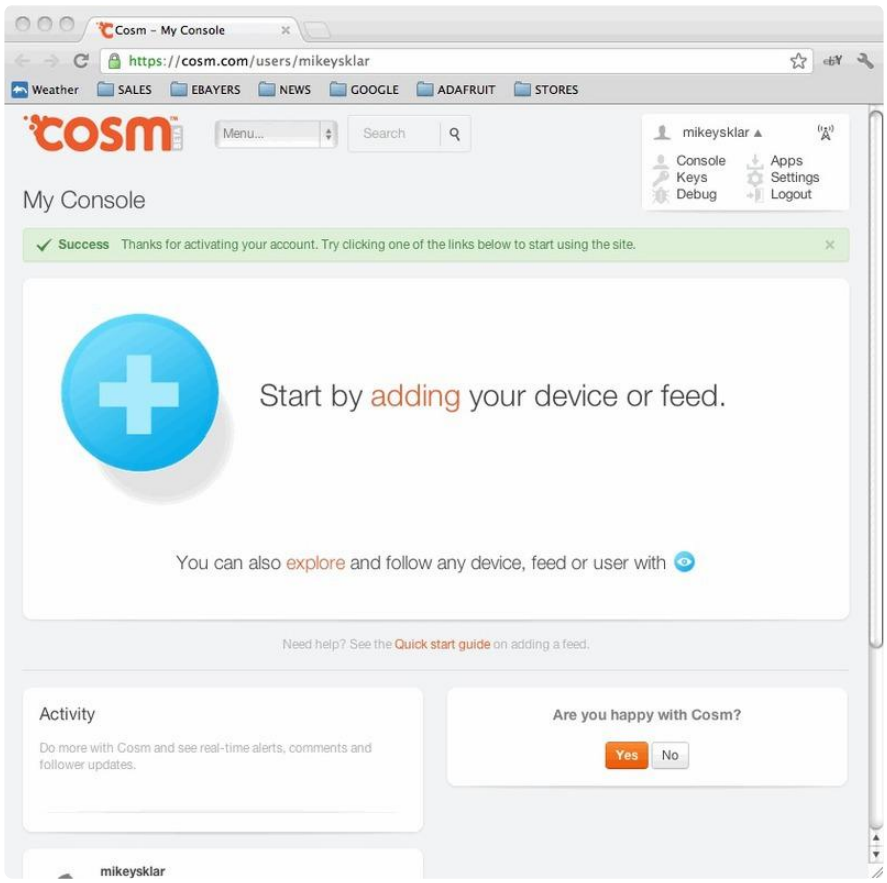
Setup a Account

You will need to create a COSM account. Click on the blue "Get Started" circle to create a new account. It's your typical e-mail/password followed by password verification. You will need to check your e-mail and click the verification link.

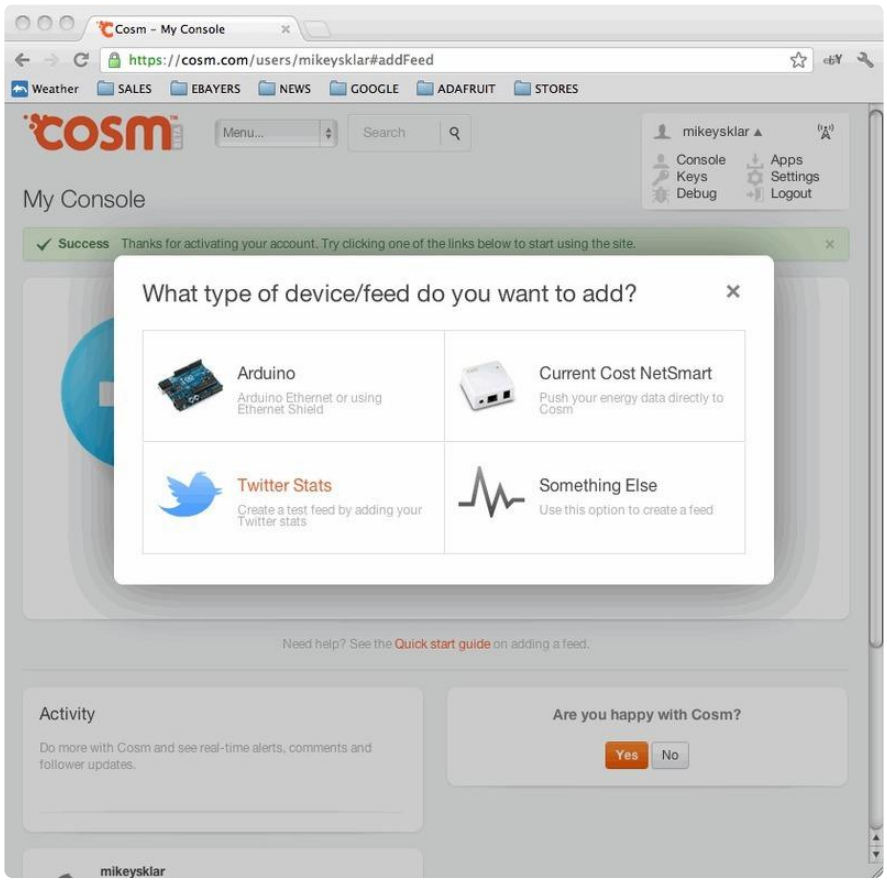


Add a Feed

Click the blue plus to add a feed.



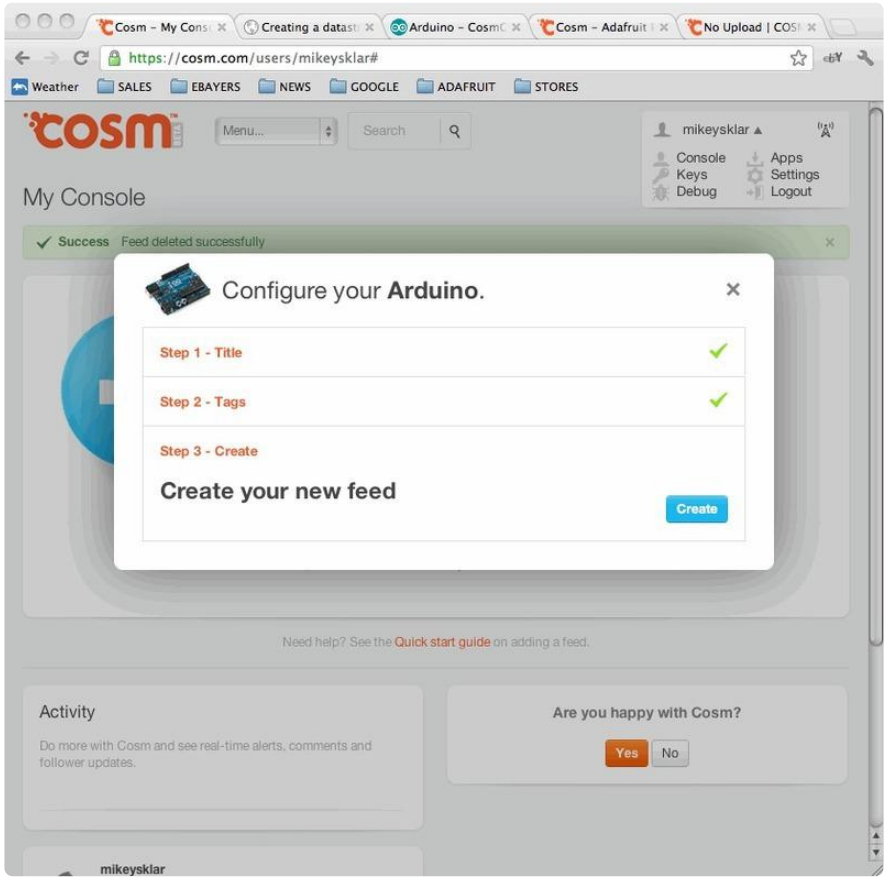
Select Arduino



Give your new feed a title and tags.

Title: "Raspberry Pi Temperature" (or whatever you like)
Tags: raspberry pi, temperature, adc (or make up your own)

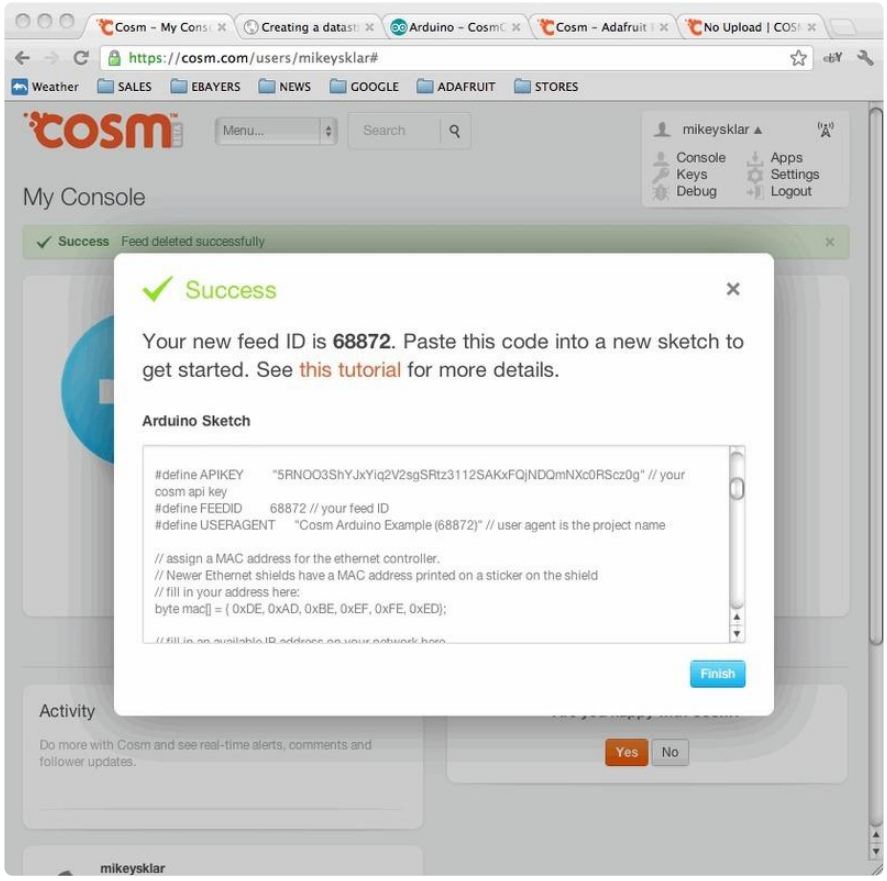
Select the "Create" button.



You need to extract the API_KEY and FEEDID from the code sample that COSM provides. These will go into the python script that we setup on the next page. The API_KEY lets COSM knows who is connecting and to which feed they want to send data.

In this example the API_KEY
is: 5RNOO3ShYJxYiq2V2sgSRtz3112SAKxFQjNDQmNXc0RScz0g
The FEEDID is: 68872

Do not use those numbers, use your own!



Python Script

Please Note: Xively no longer has free developer access to their system, so this tutorial is only for historical research. Please check out our other IoT tutorials for alternative services!

The Code

This 100+ line python script can be pasted into a editor and saved on your raspberry pi.

The script is fairly simple. Half of the code (the readadc function) is a function that will 'talk' to the MCP3008 chip using four digital pins to 'bit bang' the SPI interface (this is because not all Raspberry Pi's have the hardware SPI function).

The MCP3008 is a 10-bit ADC. That means it will read a value from 0 to 1023 (2^^10 = 1024 values) where 0 is the same as 'ground' and '1023' is the same as '3.3 volts'. We don't convert the number to voltage although its easy to do that by multiplying the number by (3.3 / 1023).

Every 30 seconds we:

- read the adc value on channel 0 (temperature sensor)
- convert the adc value to millivolts: millivolts = read_adc0 * (3300.0 / 1023.0)
- convert the millivolts value to a celsius temperature: temp_C = ((millivolts - 100.0) / 10.0) - 40.0
- convert the celsius temperature to a fahrenheit temperature: temp_F = (temp_C * 9.0 / 5.0) + 32)
- then send the data up to pachube to be saved and graphed

```
#!/usr/bin/env python
import time
import os
import RPi.GPIO as GPIO
import eeml

GPIO.setmode(GPIO.BCM)
DEBUG = 1
LOGGER = 1

# read SPI data from MCP3008 chip, 8 possible adc's (0 thru 7)
def readadc(adcnum, clockpin, mosipin, misopin, cspin):
        if ((adcnum > 7) or (adcnum < 0)):
                return -1
        GPIO.output(cspin, True)

        GPIO.output(clockpin, False)  # start clock low
        GPIO.output(cspin, False)     # bring CS low

        commandout = adcnum
        commandout |= 0x18  # start bit + single-ended bit
        commandout <<= 3    # we only need to send 5 bits here
        for i in range(5):
                if (commandout & 0x80):
                        GPIO.output(mosipin, True)
                else:
                        GPIO.output(mosipin, False)
                commandout <<= 1
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)

        adcout = 0
        # read in one empty bit, one null bit and 10 ADC bits
        for i in range(12):
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)
                adcout <<= 1
```

```
                if (GPIO.input(misopin)):
                        adcout |= 0x1

        GPIO.output(cspin, True)

        adcout /= 2       # first bit is 'null' so drop it
        return adcout

# change these as desired - they're the pins connected from the
# SPI port on the ADC to the Cobbler
SPICLK = 18
SPIMISO = 23
SPIMOSI = 24
SPICS = 25

# set up the SPI interface pins
GPIO.setup(SPIMOSI, GPIO.OUT)
GPIO.setup(SPIMISO, GPIO.IN)
GPIO.setup(SPICLK, GPIO.OUT)
GPIO.setup(SPICS, GPIO.OUT)

# COSM variables. The API_KEY and FEED are specific to your COSM account and
#API_KEY = '5RNOO3ShYJxYiq2V2sgSRtz3112SAKxFQjNDQmNXc0RScz0g'
#FEED = 68872
API_KEY = 'YOUR_API_KEY'
FEED = YOUR_FEED_ID

API_URL = '/v2/feeds/{feednum}.xml' .format(feednum = FEED)

# temperature sensor connected channel 0 of mcp3008
adcnum = 0

while True:
        # read the analog pin (temperature sensor LM35)
        read_adc0 = readadc(adcnum, SPICLK, SPIMOSI, SPIMISO, SPICS)

        # convert analog reading to millivolts = ADC * ( 3300 / 1024 )
        millivolts = read_adc0 * ( 3300.0 / 1024.0)

        # 10 mv per degree
        temp_C = ((millivolts - 100.0) / 10.0) - 40.0

        # convert celsius to fahrenheit
        temp_F = ( temp_C * 9.0 / 5.0 ) + 32

        # remove decimal point from millivolts
        millivolts = "%d" % millivolts

```

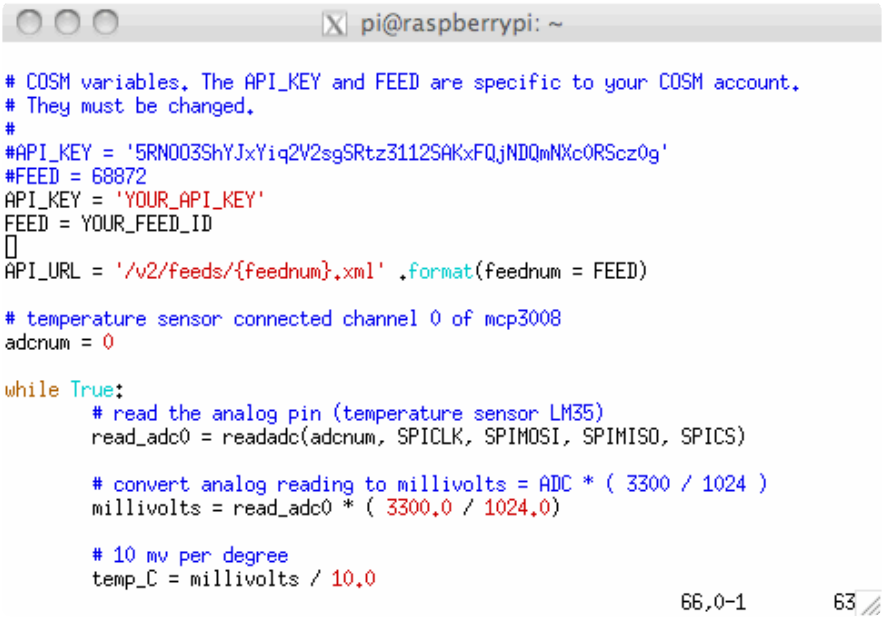
```
        # show only one decimal place for temprature and voltage readings
        temp_C = "%.1f" % temp_C
        temp_F = "%.1f" % temp_F

        if DEBUG:
                print("read_adc0:\t", read_adc0)
                print("millivolts:\t", millivolts)
                print("temp_C:\t\t", temp_C)
                print("temp_F:\t\t", temp_F)
                print("\n")

        if LOGGER:
                # open up your cosm feed
                pac = eeml.Pachube(API_URL, API_KEY)

                #send celsius data
                pac.update([eeml.Data(0, temp_C, unit=eeml.Celsius())])

                #send fahrenheit data
                pac.update([eeml.Data(1, temp_F, unit=eeml.Fahrenheit())])

                # send data to cosm
                pac.put()

        # hang out and do nothing for 10 seconds, avoid flooding cosm
        time.sleep(30)
```

adafruit-cosm-temp.py hosted with ❤ by **GitHub** **view raw**

Feeds and Keys

Update the API_KEY and FEED values to the ones that COSM provided you.

Copying over the API key incorrectly is a common (and easy to make) mistake. So have another person check your typing if you have problems!



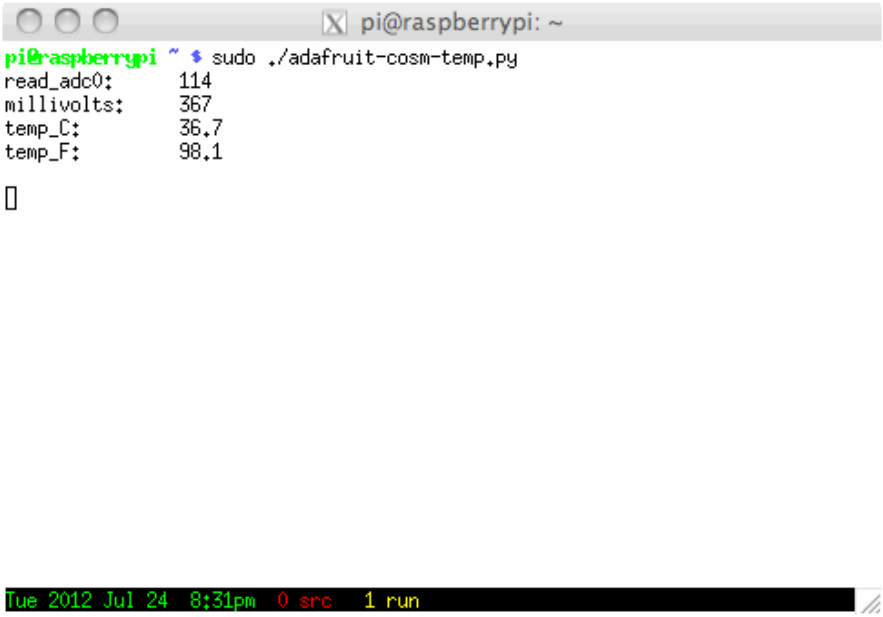
Run it!

Now that you have the code modified with your keys, go ahead and make the file executable.

```
$ chmod +x adafruit-cosm-temp.py
```

Run the script. With DEBUG = 1 (default) you will see of the adc0 value, millivolts, celsius and fahrenheit on sent to your terminals STDOUT. These same values are also being sent up to COSM.

```
$ sudo ./adafruit-cosm-temp.py
```



If you're having python crash due to an unstable internet connection, check out this handy thread over at CoSM http://community.cosm.com/node/114

COSM Graph View

Please Note: Xively no longer has free developer access to their system, so this tutorial is only for historical research. Please check out our other IoT tutorials for alternative services!

This is how COSM displays the temperature we are sending it (). We can see both celsius and fahrenheit temperature graphs. The graphs have independent sliders so it can easily be adjusted from minutes to weeks to months. There are a lot of fun settings for viewing the graph data.

A really cool feature is that you can have triggers go off based on the data values. COSM will alert you via HTTP POST or Twitter so that you can setup alarms if things go bad. If we connected up more sensors the MCP3008 we could easily have more graphs appear.