

## Send Raspberry Pi Data to COSM

Created by Michael Sklar



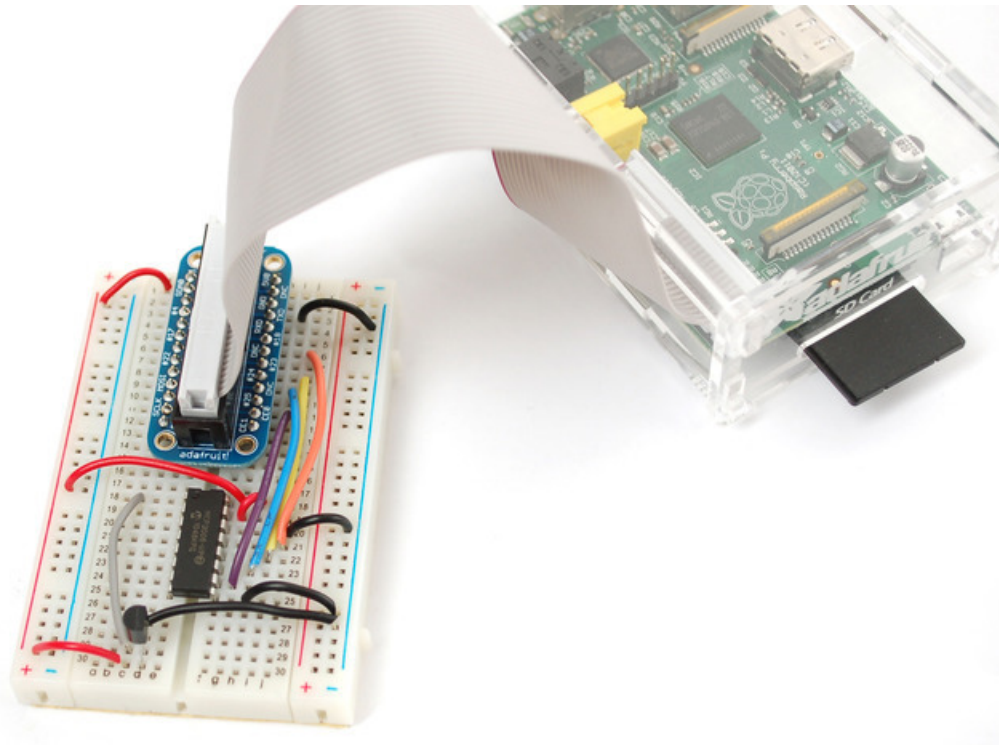
Last updated on 2018-08-22 03:31:16 PM UTC

## Guide Contents

Guide Contents	2
Overview	3
To follow this tutorial you will need	3
Connecting the Cobbler to the MCP3008 and TMP36	4
Why we need an ADC	4
Wiring Diagram	4
TMP36	6
Necessary Packages	8
COSM Account and Feed	12
Setup a Account	12
Add a Feed	12
Python Script	17
The Code	17
Feeds and Keys	19
Run it!	20
COSM Graph View	21

## 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 \(http://adafru.it/856\)](http://adafru.it/856)
- [Analog Temperature Sensor TMP-36 \(http://adafru.it/165\)](http://adafru.it/165)
- [Adafruit Pi Cobbler \(http://adafru.it/914\)](http://adafru.it/914) - follow the tutorial to assemble it
- [Half \(http://adafru.it/64\)](http://adafru.it/64) or [Full-size breadboard \(http://adafru.it/239\)](http://adafru.it/239)
- [Breadboarding wires \(https://adafru.it/aHz\)](https://adafru.it/aHz)
- 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

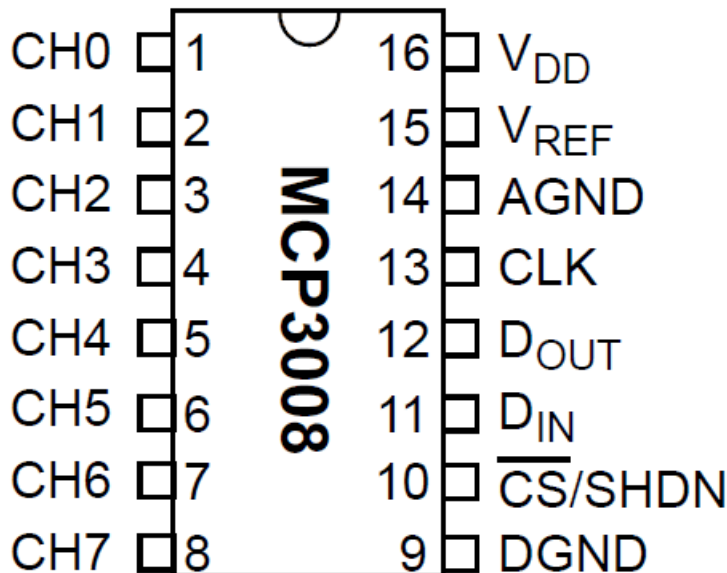
Please Note: Xively no longer has free developer access to their system, so this tutorial is only for historical research. Please check out our other IoT tutorials for alternative services!

### 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 \(http://adafru.it/856\)](http://adafru.it/856) to it. The [MCP3008 \(http://adafru.it/856\)](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 \(https://adafru.it/aHA\)](https://adafru.it/aHA), [FSRs \(https://adafru.it/aHC\)](https://adafru.it/aHC) or potentiometers, [thermistors \(https://adafru.it/aHD\)](https://adafru.it/aHD), etc.!

[Lets check the datasheet of the MCP3008 chip. \(https://adafru.it/aHE\)](https://adafru.it/aHE) 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

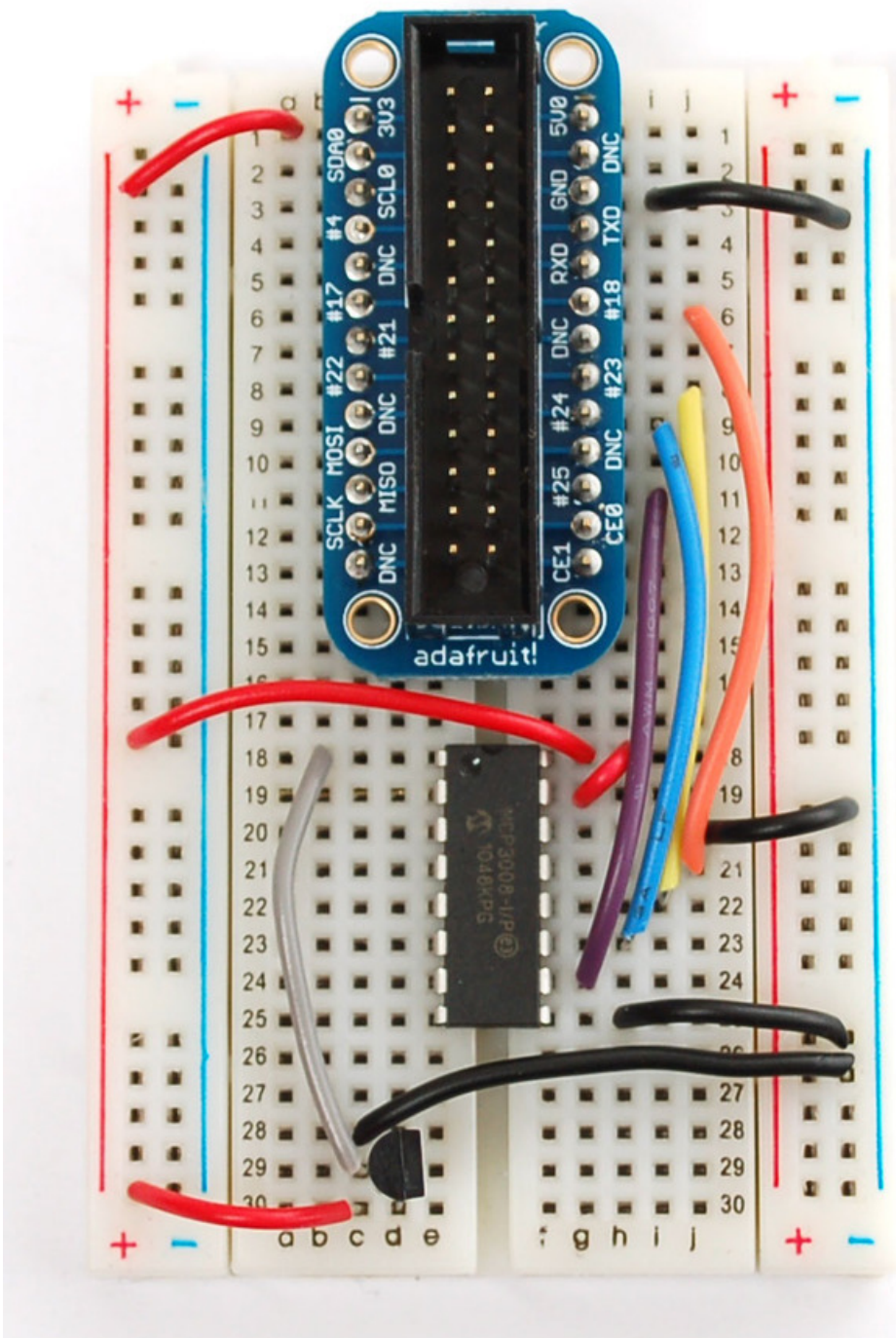
The MCP3008 has a few more pins we need to connect: **AGND** (analog ground, used sometimes in precision circuitry, which this is not) connects to **GND**, and **VREF** (analog voltage reference, used for changing the 'scale' - we want the

full scale so tie it to **3.3V**)

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)

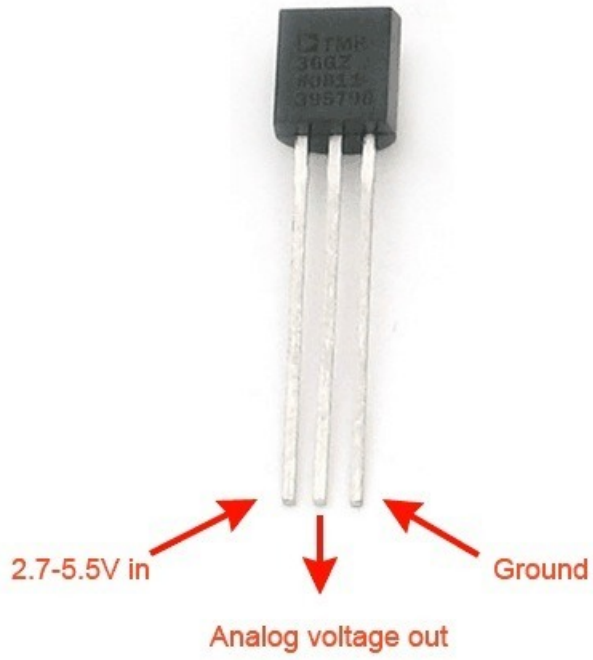
Advanced users may note that the Raspberry Pi does have a hardware SPI interface (the cobbler pins are labeled MISO/MOSI/SCLK/CE0/CE1). The hardware SPI interface is super fast but not included in all distributions. For that reason we are using a bit banded SPI implementation so the SPI pins can be any 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



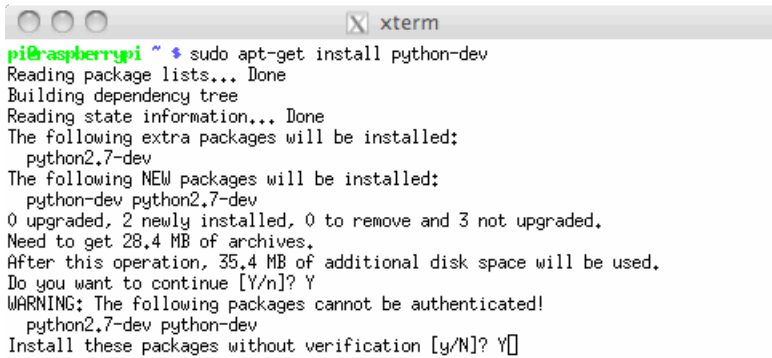
## Necessary Packages

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
```



```
pi@raspberrypi ~$ sudo apt-get install python-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  python2.7-dev
The following NEW packages will be installed:
  python-dev python2.7-dev
0 upgraded, 2 newly installed, 0 to remove and 3 not upgraded.
Need to get 28.4 MB of archives.
After this operation, 35.4 MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
WARNING: The following packages cannot be authenticated!
  python2.7-dev python-dev
Install these packages without verification [y/N]? Y
```



```
Sun 2012 Jul 22 5:05pm 0 notes 1 pi
Sun 2012 Jul 22 5:05pm 1 cosm
```

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
```



```
xterm
pi@raspberrypi ~ $ sudo apt-get install python-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  python-pkg-resources python-setuptools python2.6 python2.6-minimal
Suggested packages:
  python-distribute python-distribute-doc python2.6-doc binfmt-support
Recommended packages:
  python-dev-all
The following NEW packages will be installed:
  python-pip python-pkg-resources python-setuptools python2.6
  python2.6-minimal
0 upgraded, 5 newly installed, 0 to remove and 3 not upgraded.
Need to get 4,474 kB of archives.
After this operation, 14.5 MB of additional disk space will be used.
Do you want to continue [Y/n]?
WARNING: The following packages cannot be authenticated!
  python2.6-minimal python2.6 python-pkg-resources python-setuptools
  python-pip
Install these packages without verification [y/N]? Y
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main python2.6-minimal
  armhf 2.6.8-0.2 [1,407 kB]
Sun 2012 Jul 22 4:53pm 1 cosm
```

Install rpi.gpio (0.3.1a) or later

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

```
xterm
Processing dependencies for distribute
Finished processing dependencies for distribute
pi@raspberrypi ~ $ sudo pip install rpi.gpio
Downloading/unpacking rpi.gpio
Running setup.py egg_info for package rpi.gpio

Installing collected packages: rpi.gpio
Running setup.py install for rpi.gpio
building 'Rpi,GPIO' extension
gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-pro
totypes -fPIC -I/usr/include/python2.7 -c source/py_gpio.c -o build/temp.linu
x-armv6l-2.7/source/py_gpio.o
gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-pro
totypes -fPIC -I/usr/include/python2.7 -c source/c_gpio.c -o build/temp.linu
x-armv6l-2.7/source/c_gpio.o
gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-z,relro build/tem
p.linu
x-armv6l-2.7/source/py_gpio.o build/temp.linu
x-armv6l-2.7/source/c_gpio.o -o build/lib.linu
x-armv6l-2.7/Rpi/GPIO.so

Successfully installed rpi.gpio
Cleaning up...
pi@raspberrypi ~ $
Sun 2012 Jul 22 5:16pm 0 notes 1 pi
Sun 2012 Jul 22 5:16pm 1 cosm
```

Download EEML - markup language COSM accepts

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

```
pi@raspberrypi ~$ wget -O geekman-python-eeml.tar.gz https://github.com/geekman/python-eeml/tarball/master
--2012-07-23 22:28:21-- https://github.com/geekman/python-eeml/tarball/master
Resolving github.com (github.com)... 207.97.227.239
Connecting to github.com (github.com)|207.97.227.239|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nodeload.github.com/geekman/python-eeml/tarball/master [following]
--2012-07-23 22:28:26-- https://nodeload.github.com/geekman/python-eeml/tarball/master
Resolving nodeload.github.com (nodeload.github.com)... 207.97.227.252
Connecting to nodeload.github.com (nodeload.github.com)|207.97.227.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 17679 (17K) [application/octet-stream]
Saving to: `geekman-python-eeml.tar.gz'

100%[=====>] 17,679      55.8K/s   in 0.3s

2012-07-23 22:28:32 (55.8 KB/s) - `geekman-python-eeml.tar.gz' saved [17679/17679]

pi@raspberrypi ~$
Mon 2012 Jul 23 4:29pm 1 notes 2 pi
```

Extract the EEML tarball

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

```
pi@raspberrypi ~$ tar zxvf geekman-python-eeml.tar.gz
geekman-python-eeml-a7d2949/
geekman-python-eeml-a7d2949/.gitignore
geekman-python-eeml-a7d2949/AUTHORS
geekman-python-eeml-a7d2949/LICENSE
geekman-python-eeml-a7d2949/README.rst
geekman-python-eeml-a7d2949/eeml/
geekman-python-eeml-a7d2949/eeml/__init__.py
geekman-python-eeml-a7d2949/eeml/datastream.py
geekman-python-eeml-a7d2949/epydoc.conf
geekman-python-eeml-a7d2949/example/
geekman-python-eeml-a7d2949/example/read_serial.py
geekman-python-eeml-a7d2949/example/simple_example.py
geekman-python-eeml-a7d2949/setup.py
geekman-python-eeml-a7d2949/test/
geekman-python-eeml-a7d2949/test/eemltest.py
geekman-python-eeml-a7d2949/test/pachube.py
pi@raspberrypi ~$

Mon 2012 Jul 23 10:30pm 0 src 1 run 2 dl
Mon 2012 Jul 23 4:30pm 1 notes 2 pi
```

Change into the directory and install the EEML python package

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

```
pi@raspberrypi ~ $ cd geekman-python-eeml-*
pi@raspberrypi ~/geekman-python-eeml-a7d2949 $ sudo python setup.py install
running install
Checking .pth file support in /usr/local/lib/python2.7/dist-packages/
/usr/bin/python -E -c pass
TEST PASSED: /usr/local/lib/python2.7/dist-packages/ appears to support .pth fil
es
running bdist_egg
running egg_info
creating Python_EEML.egg-info
writing Python_EEML.egg-info/PKG-INFO
writing top-level names to Python_EEML.egg-info/top_level.txt
writing dependency_links to Python_EEML.egg-info/dependency_links.txt
writing manifest file 'Python_EEML.egg-info/SOURCES.txt'
reading manifest file 'Python_EEML.egg-info/SOURCES.txt'
writing manifest file 'Python_EEML.egg-info/SOURCES.txt'
installing library code to build/bdist.linux-armv6l/egg
running install_lib
running build_py
creating build
creating build/lib.linux-armv6l-2.7
creating build/lib.linux-armv6l-2.7/eeml
copying eeml/datastream.py -> build/lib.linux-armv6l-2.7/eeml
Mon 2012 Jul 23 4:33pm 1 notes 2 pi
```

## COSM Account and Feed

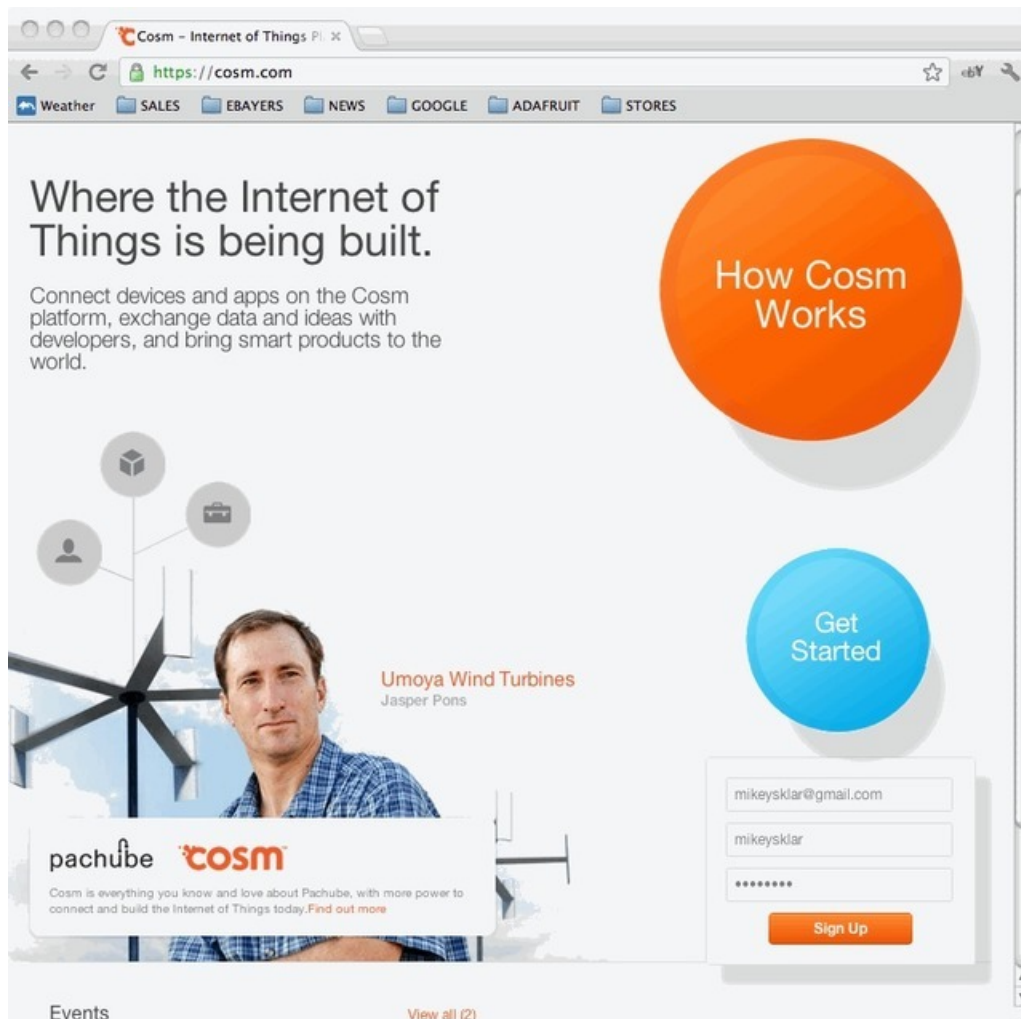
Please Note: Xively no longer has free developer access to their system, so this tutorial is only for historical research. Please check out our other IoT tutorials for alternative services!

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

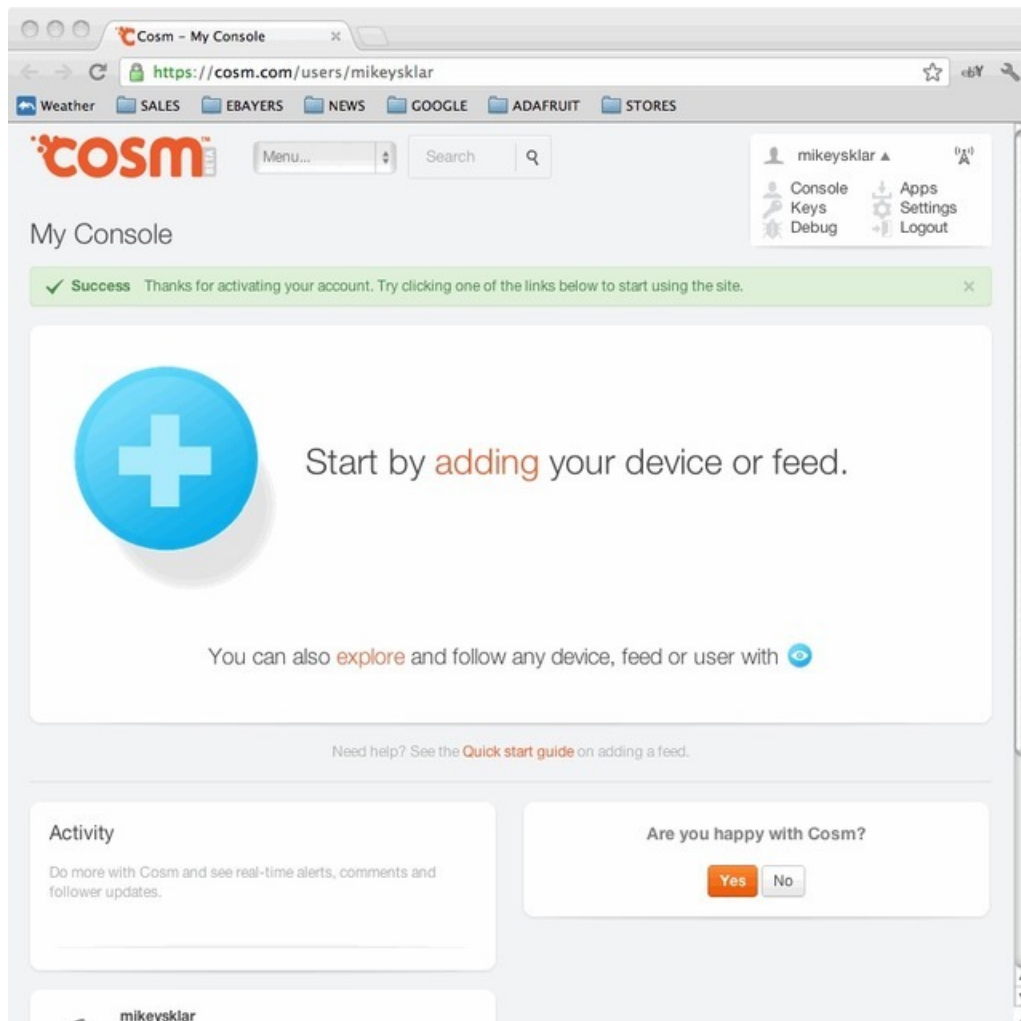
### Setup a Account

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.

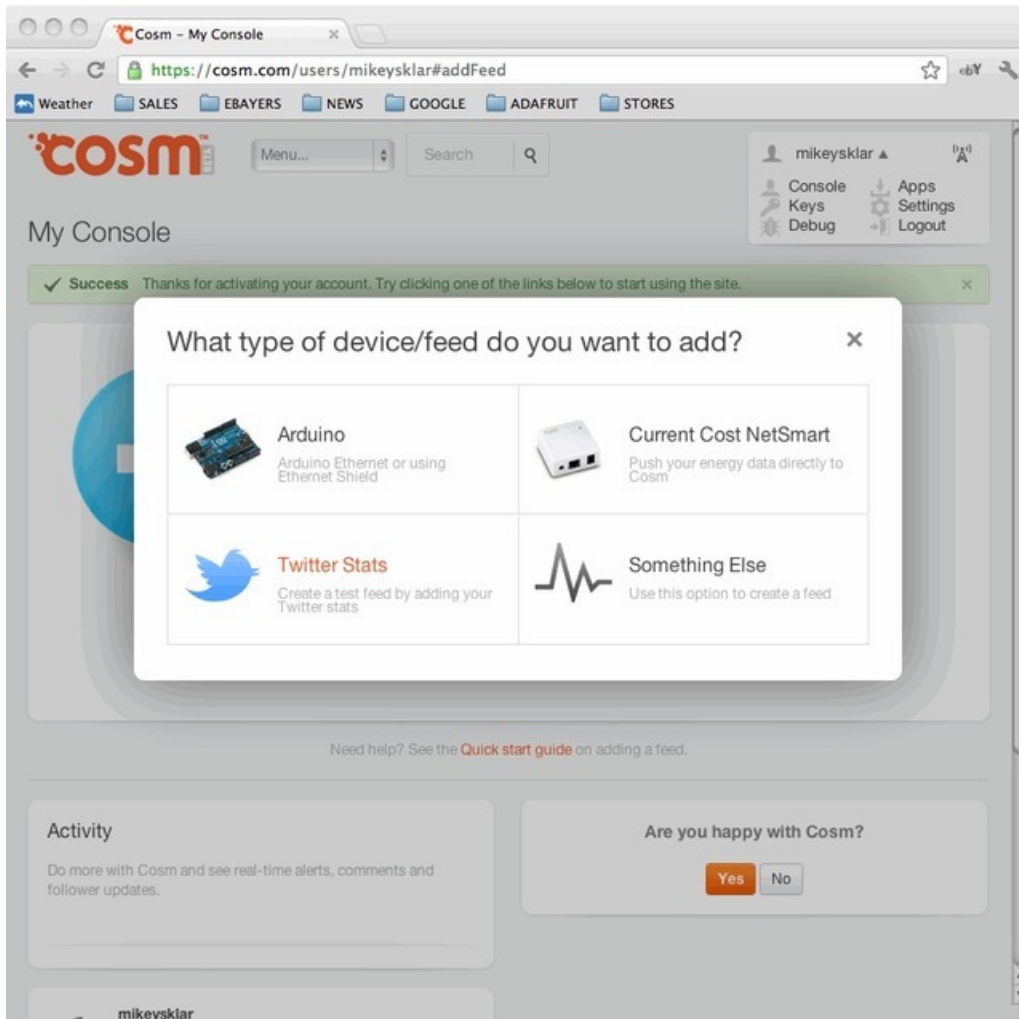


### Add a Feed

Click the blue plus to add a feed.



Select Arduino

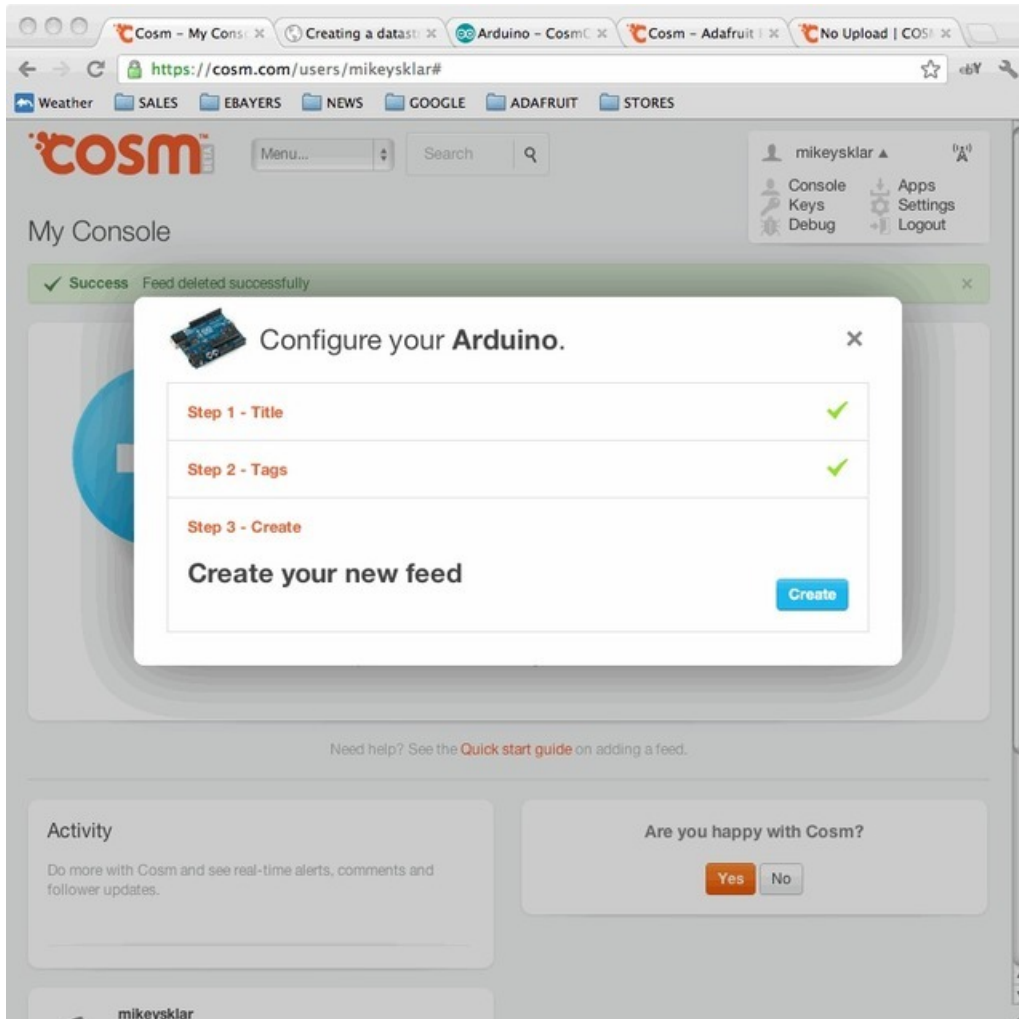


Give your new feed a title and tags.

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 know 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!

The screenshot shows a web browser window with several tabs open: "Cosm - My Console", "Creating a data...", "Arduino - Cosm", "Cosm - Adafruit", and "No Upload | COSM". The address bar shows "https://cosm.com/users/mikeysklar#". The page header includes the Cosm logo, a search bar, and a user profile for "mikeysklar" with options for Console, Keys, Debug, Apps, Settings, and Logout. A green success message at the top reads "Success Feed deleted successfully". A white dialog box with a green checkmark and the word "Success" is centered on the screen. It contains the text: "Your new feed ID is **68872**. Paste this code into a new sketch to get started. See [this tutorial](#) for more details." Below this is a section titled "Arduino Sketch" with a text area containing the following code:

```
#define APIKEY    "5RN003ShYJxYIq2V2sgSRtz3112SAKxFQjNDQmNXc0RScz0g" // your  
cosm api key  
#define FEEDID    68872 // your feed ID  
#define USERAGENT "Cosm Arduino Example (68872)" // user agent is the project name  
  
// assign a MAC address for the ethernet controller.  
// Newer Ethernet shields have a MAC address printed on a sticker on the shield  
// fill in your address here:  
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};  
  
// fill in an available IP address on your network here.
```

At the bottom of the dialog box is a blue "Finish" button. Below the dialog box, there are "Yes" and "No" buttons. The background page shows an "Activity" section with the text "Do more with Cosm and see real-time alerts, comments and follower updates." and a user profile for "mikeysklar".



# 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

```
1  #!/usr/bin/env python
2  import time
3  import os
4  import RPi.GPIO as GPIO
5  import eeml
6
7  GPIO.setmode(GPIO.BCM)
8  DEBUG = 1
9  LOGGER = 1
10
11 # read SPI data from MCP3008 chip, 8 possible adc's (0 thru 7)
12 def readadc(adcnun, clockpin, mosipin, misopin, cspin):
13     if ((adcnun > 7) or (adcnun < 0)):
14         return -1
15     GPIO.output(cspin, True)
16
17     GPIO.output(clockpin, False) # start clock low
18     GPIO.output(cspin, False) # bring CS low
19
20     commandout = adcnun
21     commandout |= 0x18 # start bit + single-ended bit
22     commandout <<= 3 # we only need to send 5 bits here
23     for i in range(5):
24         if (commandout & 0x80):
25             GPIO.output(mosipin, True)
26         else:
27             GPIO.output(mosipin, False)
28         commandout <<= 1
29         GPIO.output(clockpin, True)
```

```

30     GPIO.output(clockpin, False)
31
32     adcout = 0
33     # read in one empty bit, one null bit and 10 ADC bits
34     for i in range(12):
35         GPIO.output(clockpin, True)
36         GPIO.output(clockpin, False)
37         adcout <<= 1
38         if (GPIO.input(misopin)):
39             adcout |= 0x1
40
41     GPIO.output(cspin, True)
42
43     adcout /= 2     # first bit is 'null' so drop it
44     return adcout
45
46 # change these as desired - they're the pins connected from the
47 # SPI port on the ADC to the Cobbler
48 SPICLK = 18
49 SPIMISO = 23
50 SPIMOSI = 24
51 SPICS = 25
52
53 # set up the SPI interface pins
54 GPIO.setup(SPIMOSI, GPIO.OUT)
55 GPIO.setup(SPIMISO, GPIO.IN)
56 GPIO.setup(SPICLK, GPIO.OUT)
57 GPIO.setup(SPICS, GPIO.OUT)
58
59 # COSM variables. The API_KEY and FEED are specific to your COSM account and must be changed
60 #API_KEY = '5RNOO3ShYJxYiq2V2sgSRtz3112SAKxFQjNDQmNXc0RScz0g'
61 #FEED = 68872
62 API_KEY = 'YOUR_API_KEY'
63 FEED = YOUR_FEED_ID
64
65 API_URL = 'v2/feeds/{feednum}.xml'.format(feednum = FEED)
66
67 # temperature sensor connected channel 0 of mcp3008
68 adcnum = 0
69
70 while True:
71     # read the analog pin (temperature sensor LM35)
72     read_adc0 = readadc(adcnum, SPICLK, SPIMOSI, SPIMISO, SPICS)
73
74     # convert analog reading to millivolts = ADC * ( 3300 / 1024 )
75     millivolts = read_adc0 * ( 3300.0 / 1024.0)
76
77     # 10 mv per degree
78     temp_C = ((millivolts - 100.0) / 10.0) - 40.0
79
80     # convert celsius to fahrenheit
81     temp_F = ( temp_C * 9.0 / 5.0 ) + 32
82
83     # remove decimal point from millivolts
84     millivolts = "%d" % millivolts
85
86     # show only one decimal place for temprature and voltage readings

```

```
87 temp_C = "%.1f" % temp_C
88 temp_F = "%.1f" % temp_F
89
90 if DEBUG:
91     print("read_adc0:\t", read_adc0)
92     print("millivolts:\t", millivolts)
93     print("temp_C:\t\t", temp_C)
94     print("temp_F:\t\t", temp_F)
95     print("\n")
96
97 if LOGGER:
98     # open up your cosm feed
99     pac = eeml.Pachube(API_URL, API_KEY)
100
101     #send celsius data
102     pac.update([eeml.Data(0, temp_C, unit=eeml.Celsius())])
103
104     #send fahrenheit data
105     pac.update([eeml.Data(1, temp_F, unit=eeml.Fahrenheit())])
106
107     # send data to cosm
108     pac.put()
109
110     # hang out and do nothing for 10 seconds, avoid flooding cosm
111     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!

```
pi@raspberrypi: ~
# COSM variables. The API_KEY and FEED are specific to your COSM account.
# They must be changed.
#
#API_KEY = '5RN003ShYJxYiq2V2sgSRtz3112SAKxFOjNDQmNXc0RScz0g'
#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 / 10.0
    66,0-1      63
```

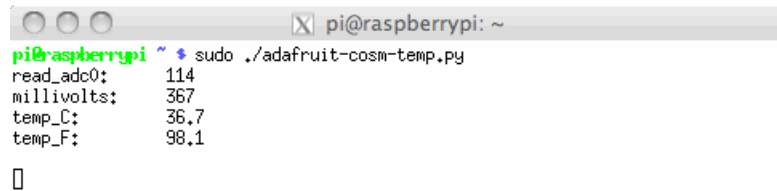
## Run it!

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
```



```
pi@raspberrypi: ~  
pi@raspberrypi ~$ sudo ./adafruit-cosm-temp.py  
read_adc0:    114  
millivolts:   367  
temp_C:       36.7  
temp_F:       98.1  
[]
```

```
Tue 2012 Jul 24 8:31pm 0 src 1 run
```

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 \(https://adafru.it/aNa\)](https://adafru.it/aNa). 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.

