# Python Packages for IoT

P.S.NANDHINI / CSE
Kongu Engineering College

# 1. JSON

- JSON – JavaScript Object Notation – easy to read and write data-interchange format
- Alternative to XML – easy to be parsed by machine and generation
- Built on two-structures
  - ✔ a collection of name – value pairs (dictionary)
  - ✔ Ordered list of values (lists)
- JSON format – used for serializing and transmitting structured data over the network connection – transmitting data between server and web application
- Exchange of information encoded as JSON involves encoding and decoding provided by python packages

# Python JSON Syntax

JSON is written as key and value pair.

```
{
    “key” : “value”,
    “key” : “value”
}
```

**Methods**

- ✔ dumps() – encoding to JSON objects
- ✔ dump() – encoded string writing on file
- ✔ loads() – Decode the JSON string
- ✔ load() – Decode while JSON file read

# 2. XML

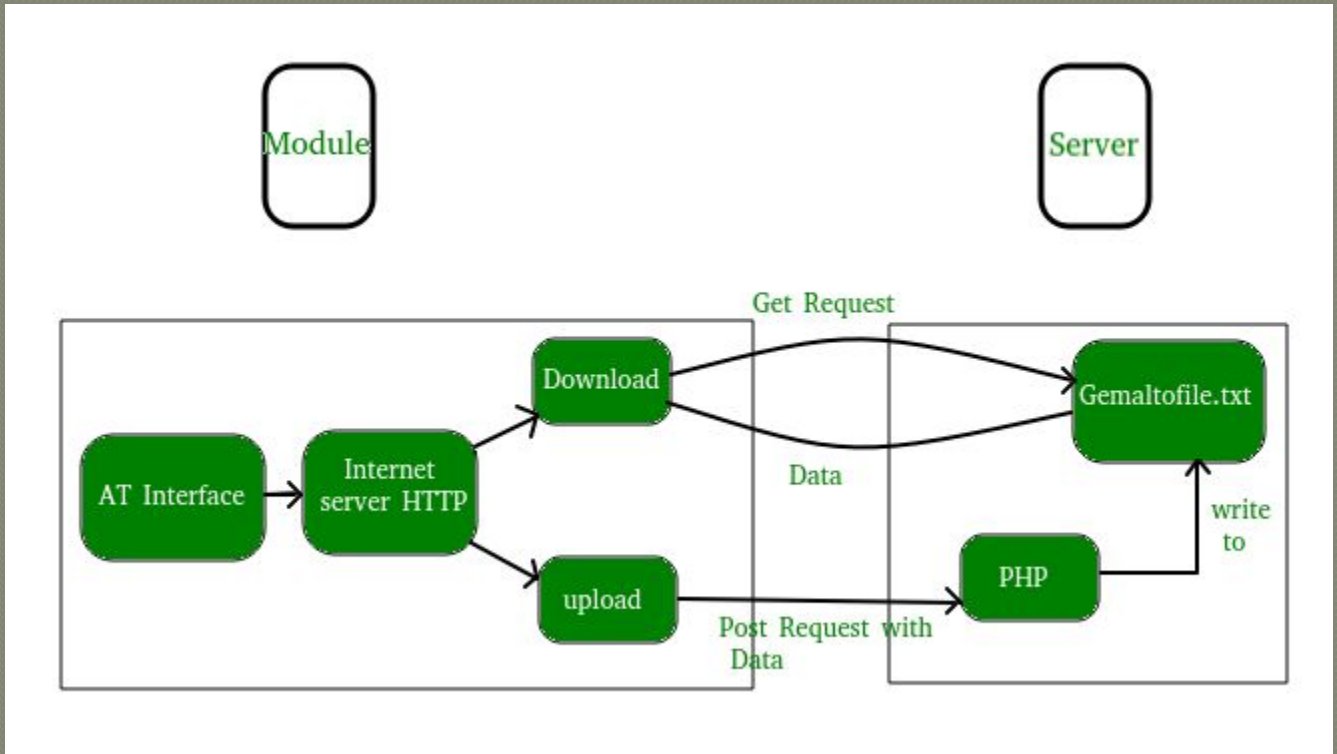
- XML – Extensible Markup Language – data format for structured document interchange
- Python package – **minidom**

# 3. HTTPLib & URLLib

- HTTPLib2 & URLLib2 – python libraries - internet programming
- HTTPLib2 – HTTP Client Library
- URLLib2 – library for fetching URLs
- HTTP is a set of protocols designed to enable communication between clients and servers. It works as a request-response protocol between a client and server.
- A web browser may be the client, and an application on a computer that hosts a web site may be the server.
- So, to request a response from the server, there are mainly two methods:

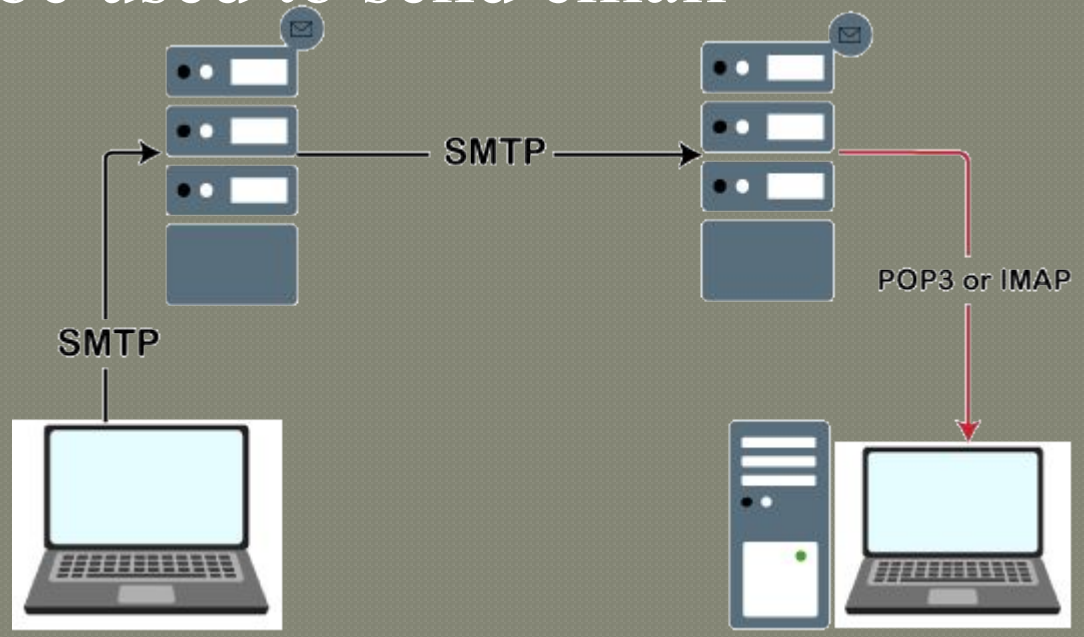
1. GET : to request data from the server.
2. POST : to submit data to be processed to the server.

# HTTP


Module
Server
Get Request
Download
Gemaltofile.txt
AT Interface
Internet server HTTP
Data
write to
PHP
upload
Post Request with Data

# 4. SMTPLib

- SMTP: Sending e-mail and routing email between mail servers
- Package : smtplib module provides an SMTP client session object that can be used to send email



# INTRODUCTION TO RASPBERRY PI

P.S.NANDHINI
AP /CSE
Kongu Engineering College

# OUTLINE

- Introduction to Raspberry PI
- Interfaces (Serial,SPI,I2C) Programming
- Python programming with Raspberry PI with focus of interfacing external gadgets
- Controlling output
- Reading input from pins
- Connecting IoT to Cloud
- ThingSpeak

# Outline



- Basic building blocks of an IoT Device
- Exemplary Device: Raspberry Pi
- Raspberry Pi interfaces
- Programming Raspberry Pi with Python
- Other IoT devices

# Basic building blocks of an IoT Device



- A "Thing" in IoT can be any object that has a unique identifier and which can send/receive data (including user data) over a network
- E.g., smart phone, smart TV, computer, refrigerator, car, etc.
- IoT devices are connected to the Internet and send information about themselves or about their surroundings (e.g. information sensed by the connected sensors) over a network (to other devices or servers/storage) or allow actuation upon the physical entities/environment around them remotely

# Basic building blocks of an IoT Device



- IoT Device Examples
  - A home automation device that allows remotely monitoring the status of appliances and controlling the appliances
  - An industrial machine which sends information about its operation and health monitoring data to a server
  - A car which sends information about its location to a cloud-based service
  - A wireless-enabled wearable device that measures data about a person such as the number of steps walked and sends the data to a cloud-based service

# Basic building blocks of an IoT Device



Sensing

- Sensors can be either on-board IoT device or attached to the device

Actuation

- IoT devices can have various types of actuators attached that allow taking **actions upon the physical entities** in the vicinity of the device

Communication

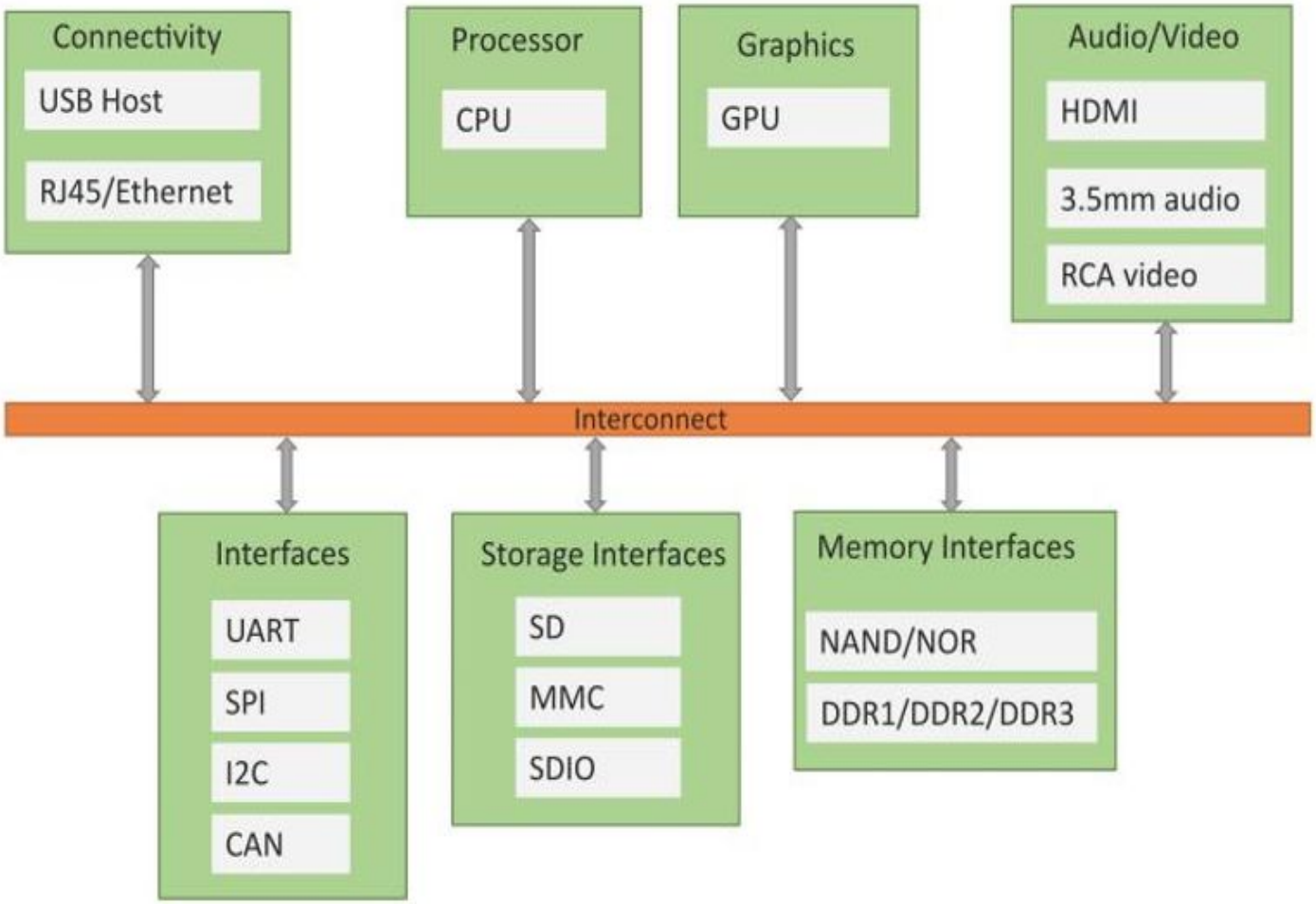
- Communication modules are responsible for sending collected data to other devices or cloud-based servers/storage and receiving data from other devices and commands from remote applications

Analysis & Processing

- Analysis and processing modules are responsible for making sense of the collected data

# Block diagram of an IoT Device





# Exemplary Device: Raspberry Pi



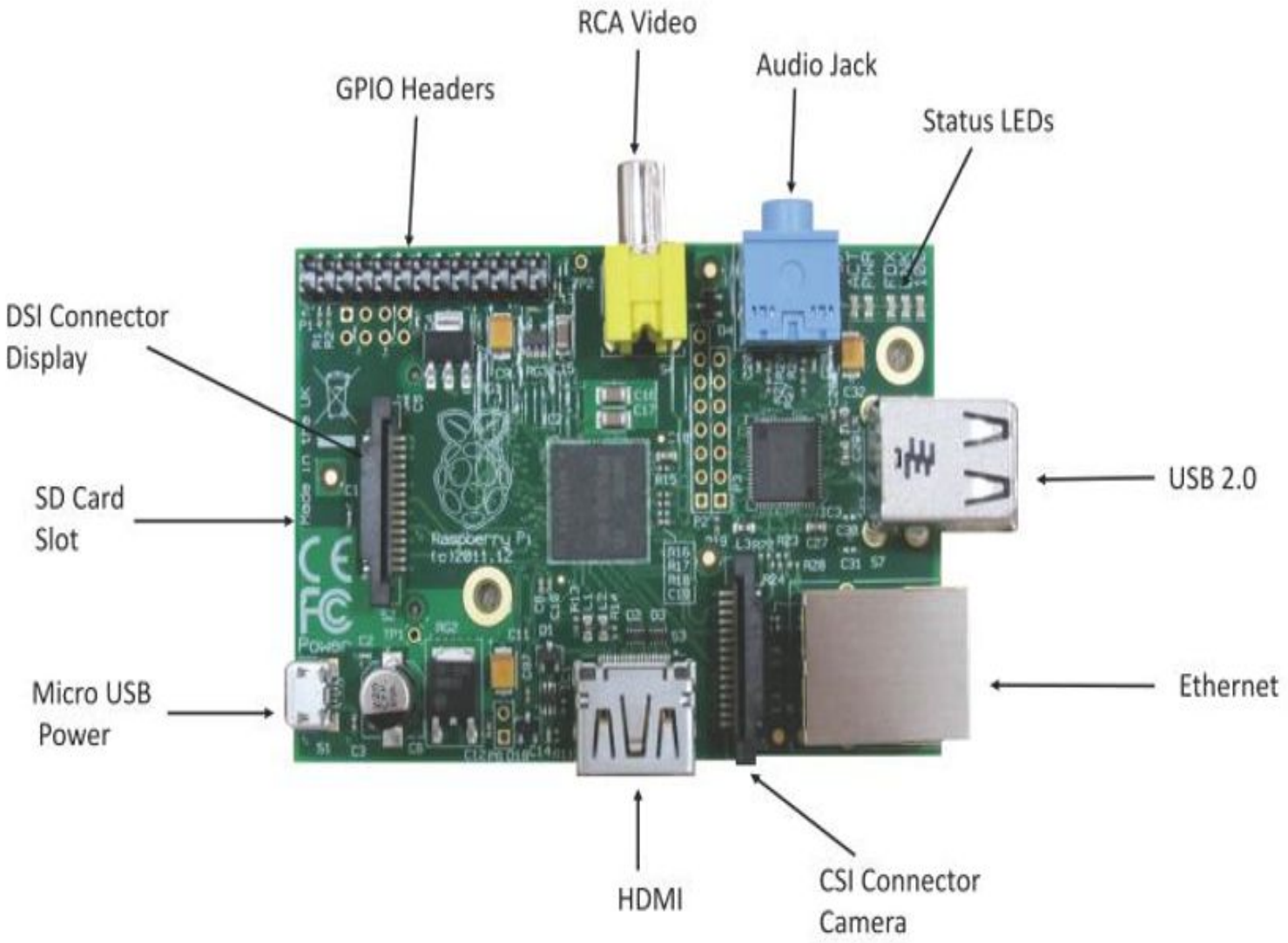
- A low-cost mini-computer with the physical size of a credit card
- Runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do
- Raspberry Pi also allows interfacing sensors and actuators through the general purpose I/O pins (GPIO)
- Raspberry Pi runs Linux operating system, it supports Python "out of the box"

# Raspberry Pi Status leds



| STATUS LED | FUNCTION |
|---|---|
| ACT | SD card access |
| PWR | 3.3V Power is present |
| FDX | Full Duplex LAN connected |
| LNK | Link/Network activity |
| 100 | 100 Mbit LAN connected |

# Raspberry Pi Board

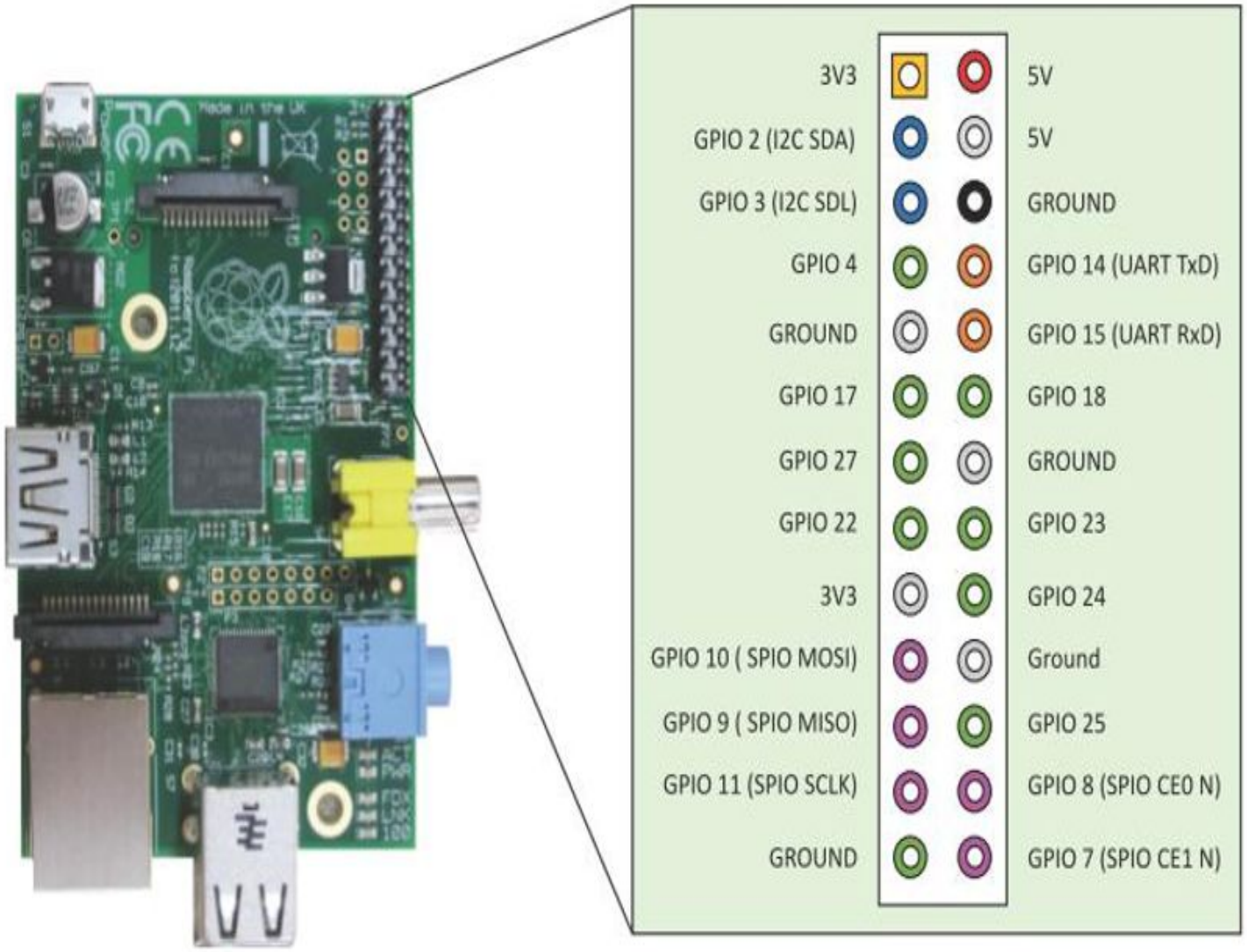


# Linux on Raspberry Pi



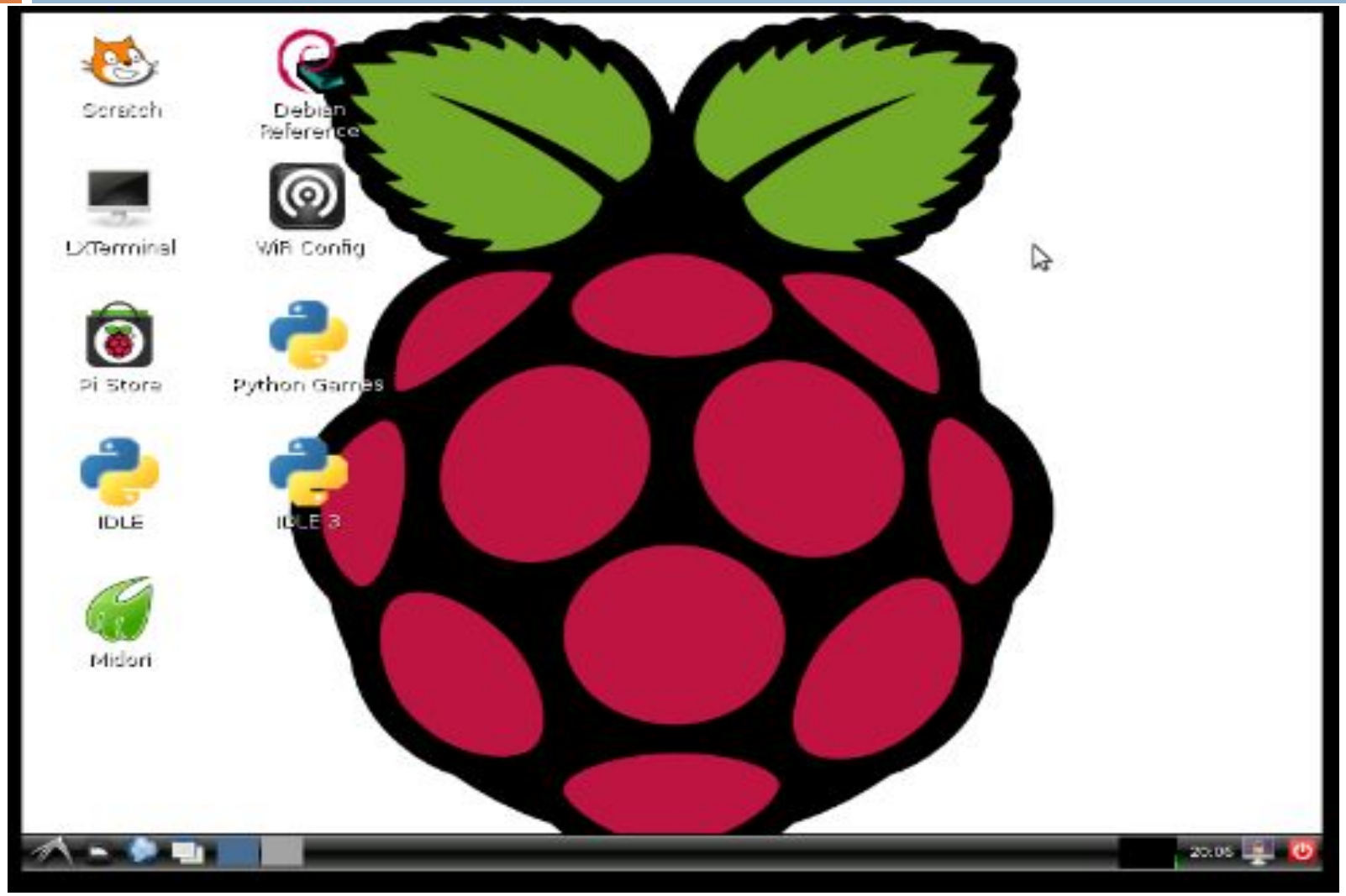
- **Raspbian**
  - Raspbian Linux is a Debian Wheezy port optimized for Raspberry Pi
- **Arch**
  - Arch is an Arch Linux port for AMD devices
- **Pidora**
  - Pidora Linux is a Fedora Linux optimized for Raspberry Pi
- **RaspBMC**
  - RaspBMC is an XBMC media-center distribution for Raspberry Pi
- **OpenELEC**
  - OpenELEC is a fast and user-friendly XBMC media-center distribution
- **RISC OS**
  - RISC OS is a very fast and compact operating system

# Raspberry Pi GPIO headers



# Rasbian Linux Desktop





# Raspberry Pi frequently used commands



| Command | Function | Example |
|---|---|---|
| cd | change directory | cd/home/pi |
| cat | show file contents | cat file.txt |
| ls | list files and folders | ls/home/pi |
| locate | search for a file | locate file.txt |
| lsusb | list usb devices | lsusb |
| pwd | print name for present working directory | pwd |
| mkdir | make directory | mkdir/home/pi/new |
| mv | move(rename) file | mv sourcefile.txt destfile.txt |
| rm | remove file | rm file.txt |
| reboot | reboot device | sudo reboot |
| shutdown | shutdown device | sudo shutdown –h now |

# Raspberry Pi frequently used commands



| Command | Function | Example |
|---|---|---|
| grep | Print lines matching a pattern | grep –r “pi”/home/ |
| df | Report file system disk space usage | df -Th |
| ipconfig | Configure a network interface | ipconfig |
| netstat | Print network connections, routing tables, interface statistics | Netstat -lntp |
| tar | Extract /create archive | Tar –xzf foo.tar.gz |
| wget | Non-interactive network downloader | Wget http://example.com/filr.tar.gz |

# Raspberry Pi Interfaces



- **Serial**
  - The serial interface on Raspberry Pi has receive (Rx) and transmit (Tx) pins for communication with serial peripherals
- **SPI**
  - Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices
- **I2C**
  - The I2C interface pins on Raspberry Pi allows to connect hardware modules
  - I2C interface allows synchronous data transfer with just two pins:

    - SDA (data line) and
    - SCL (clock line)

# Raspberry Pi Example: Interfacing LED and switch with Raspberry Pi

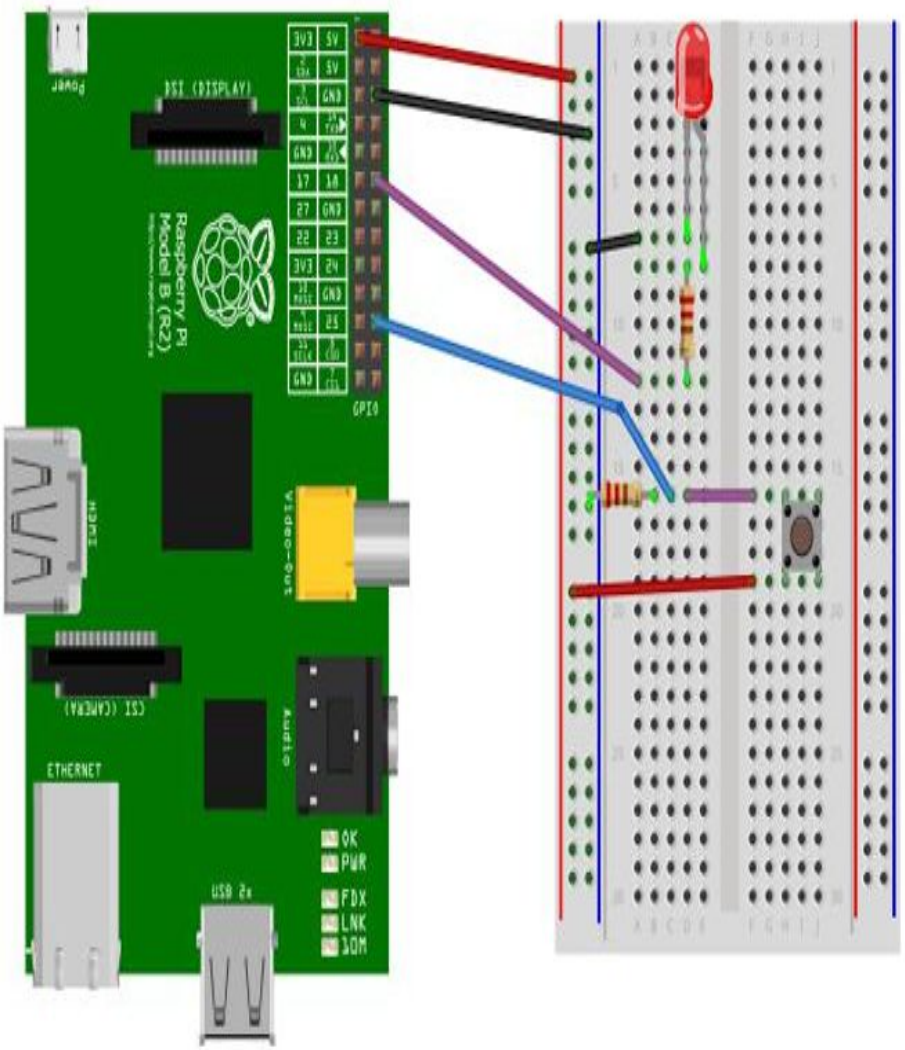


```
from time import sleep
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)

#Switch Pin
GPIO.setup(25, GPIO.IN)
#LED Pin
GPIO.setup(18, GPIO.OUT)
state=false

def toggleLED(pin):
        state = not state
        GPIO.output(pin, state)

while True:
        try:
                if (GPIO.input(25) == True):
                        toggleLED(pin)
                sleep(.01)
                except KeyboardInterrupt:
                        exit()
```



# Other IoT Devices



pcDuino

# Other IoT Devices



BeagleBone Black

# Other IoT Devices



## Cubieboard

# Cloud Offering - ThingSpeak

P.S.Nandhini / CSE

Kongu Engineering College

# ThingSpeak

- ThingSpeak is an IoT analytics platform service that allows user to aggregate, visualize, and analyze live data streams in the cloud.
- The user can send data to ThingSpeak from their devices, create instant visualization of live data, and send alerts.

# ThingSpeak Features

- ✔ Collect data in private Channels
- ✔ Share data with public Channels
- ✔ RESTful and MQTT APIs
- ✔ Analytics and Visualization
- ✔ Event Scheduling
- ✔ Alerts
- ✔ App Integration

# 1. Write data to Channel

- REST API
- MQTT API

# REST API

- Update channel data with HTTP GET or POST
- Write many entries to channel in JSON format with single HTTP POST

# Update channel data with HTTP GET or POST

- URL - https://api.thingspeak.com/update.json
- Response – Success – HTTP Status code – 200 OK

| Name | Description | Value Type |
|---|---|---|
| api_key | (Required) Write API Key for this specific channel. You can also send the Write API Key by using a THINGSPEAKAPIKEY HTTP header. The Write API Key is found on the **API Keys** tab of the channel view. | string |
| field<X> | (Optional) Field X data, where X is the field ID | any |
| lat | (Optional) Latitude in degrees, specified as a value between -90 and 90. | decimal |
| long | (Optional) Longitude in degrees, specified as a value between -180 and 180. | decimal |
| elevation | (Optional) Elevation in meters | integer |
| status | (Optional) Status update message. | string |
| twitter | (Optional) Twitter® username linked to ThingTweet | string |
| tweet | (Optional) Twitter status update | string |
| created_at | (Optional) Date when feed entry was created, in ISO 8601 format, for example: 2014-12-31 23:59:59. The date you specify must be unique within the channel. Time zones can be specified using the timezone parameter. | datetime |

## Text Example

```
POST https://api.thingspeak.com/update
     api_key=XXXXXXXXXXXXXXXX
     field1=73
```

### JSON Example

```
POST https://api.thingspeak.com/update.json
     api_key=XXXXXXXXXXXXXXXX
     field1=73
```

The response is a JSON object of the new entry, for example:

```
{
  "channel_id": 3,
  "field1": '73',
  "field2": null,
  "field3": null,
  "field4": null,
  "field5": null,
  "field6": null,
  "field7": null,
  "field8": null,
  "created_at": '2014-02-25T14:13:01-05:00',
  "entry_id": 320,
  "status": null,
  "latitude": null,
  "longitude": null,
  "elevation": null
}
```

### XML Example

```
POST https://api.thingspeak.com/update.xml
     api_key=XXXXXXXXXXXXXXXX
     field1=73
```

The response is an XML object of the new entry, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<feed>
  <channel-id type="integer">3</channel-id>
  <field1>73</field1>
  <field2 nil="true"/>
  <field3 nil="true"/>
  <field4 nil="true"/>
  <field5 nil="true"/>
  <field6 nil="true"/>
  <field7 nil="true"/>
  <field8 nil="true"/>
  <created-at type="dateTime">2014-02-25T14:15:42-05:00</created-at>
  <entry-id type="integer">321</entry-id>
  <status nil="true"/>
  <latitude type="decimal" nil="true"/>
  <longitude type="decimal" nil="true"/>
  <elevation nil="true"/>
</feed>
```

# Sample

## Examples

### Write Data with GET

You can use your web browser to complete GET HTTP requests to the RESTful API for ThingSpeak™.

Copy the URL to the address bar of your web browser, changing `<write_api_key>` to your user API Key, which is found in **Account** > **My Profile**.

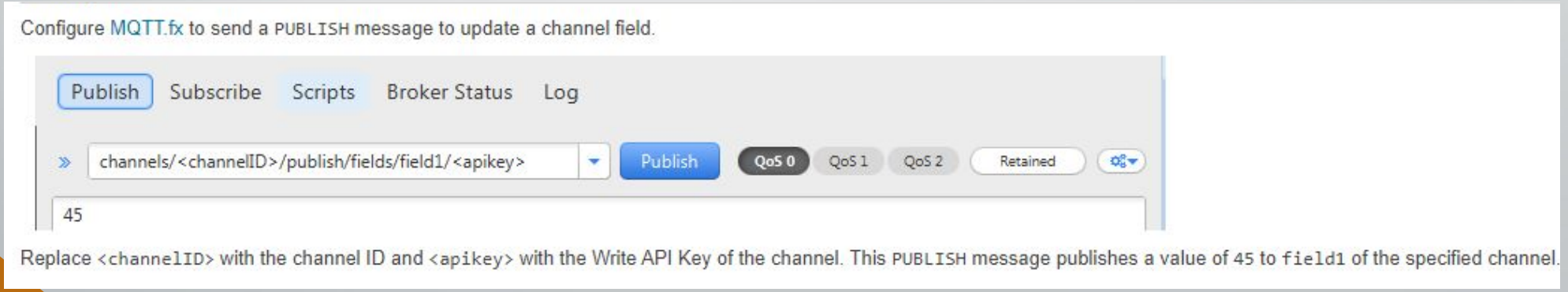
```
https://api.thingspeak.com/update.json?api_key=<write_api_key>&field1=123
```

The response is a JSON object of the new entry, and a 200 OK from the server.


```
{
    "channel_id": 266256,
    "created_at": "2018-09-10T17:41:59Z",
    "entry_id": 2,
    "field1": "123",
    "field2": null,
    "field3": null,
    "field4": null,
    "field5": null,
    "field6": null,
    "field7": null,
    "field8": null,
    "latitude": null,
    "longitude": null,
    "elevation": null,
    "status": null
}
```

# MQTT API

- Publish to a channel Feed - Publish message to update multiple channel fields simultaneously
- Publish to a channel Field Feed - Publish message to update single channel field

# 2. Read Data from Channel

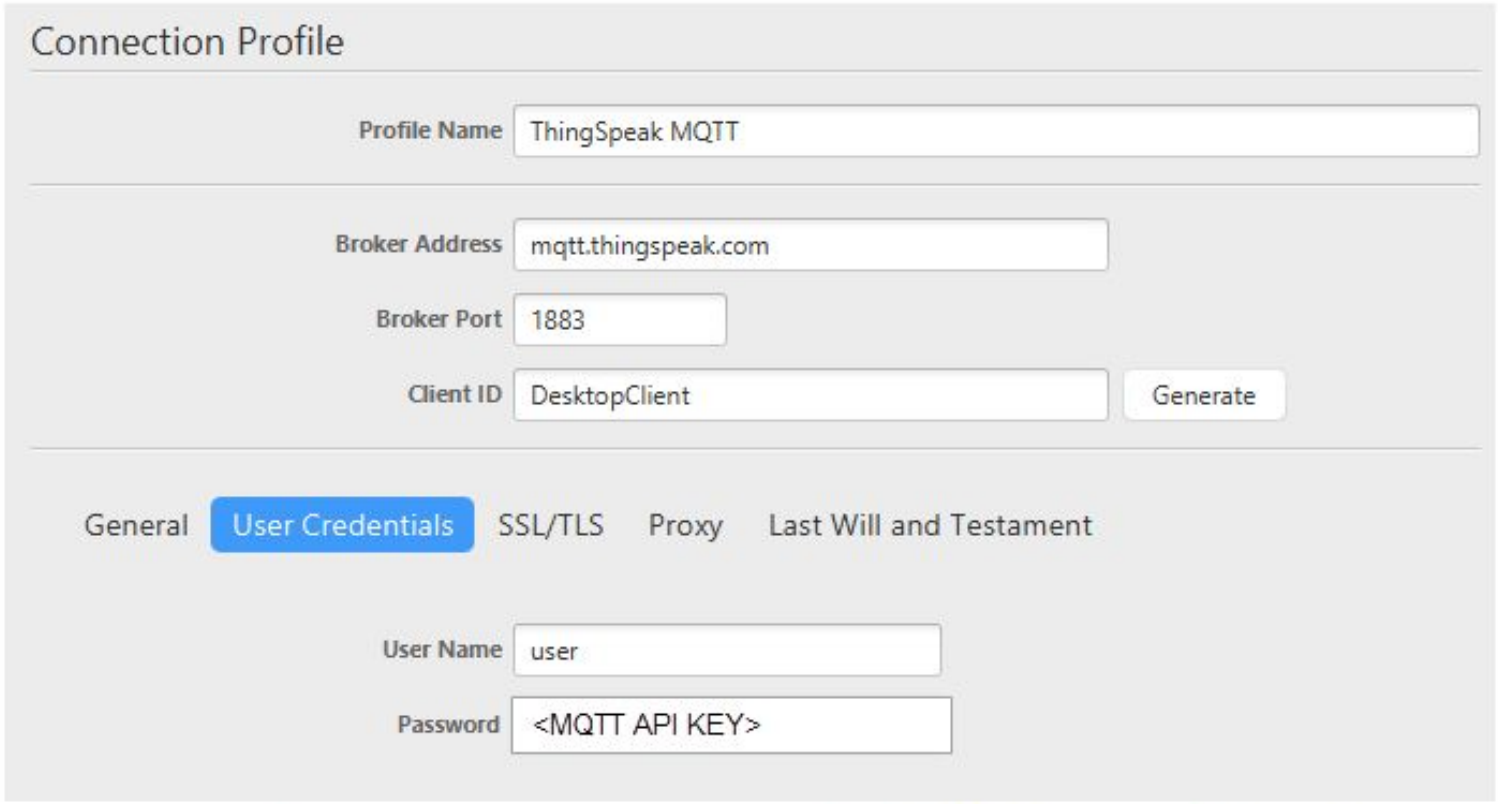
- REST API
- MQTT API

# REST API

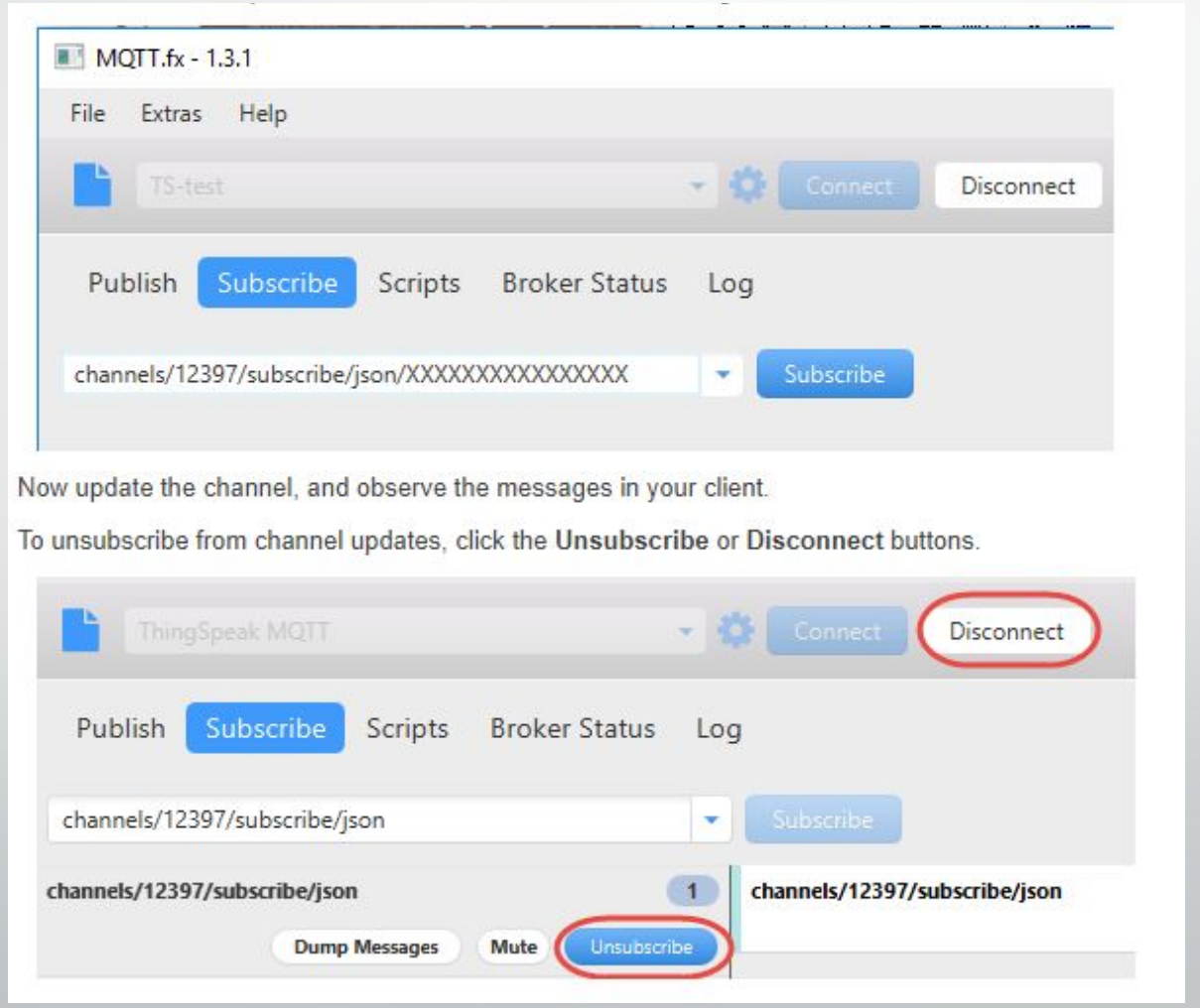
| | |
|---|---|
| `Read Data` | Read data from all fields in channel with HTTP GET |
| `Read Field` | Read data from single field of channel with HTTP GET |
| `Read Status` | Read status field of channel with HTTP GET |
| `Read Last Entry` | Read last entry in channel with HTTP GET |
| `Read Last Field Entry` | Read last entry in channel field with HTTP GET |
| `Read Last Status` | Read last status of channel with HTTP GET |

# MQTT API

Configure MQTT.fx to subscribe to channel updates from the MathWorks® weather station. Use `mqtt.thingspeak.com` and port `1883`. Enter your MQTT API Key as the password.



# MQTT API


MQTT.fx - 1.3.1
File
Extras
Help
TS-test
Connect
Disconnect
Publish
Subscribe
Scripts
Broker Status
Log
channels/12397/subscribe/json/XXXXXXXXXXXXXXX
Subscribe
Now update the channel, and observe the messages in your client.
To unsubscribe from channel updates, click the Unsubscribe or Disconnect buttons.
ThingSpeak MQTT
Connect
Disconnect
Publish
Subscribe
Scripts
Broker Status
Log
channels/12397/subscribe/json
Subscribe
channels/12397/subscribe/json
1
channels/12397/subscribe/json
Dump Messages
Mute
Unsubscribe

```
import sys
import urllib2
from time import sleep
import Adafruit_DHT as dht

# Enter Your API key here
myAPI = '5QTYDNRHSJ5RESA5'
# URL where we will send the data, Don't change it
baseURL = 'https://api.thingspeak.com/update?api_key=%s' % myAPI

def DHT22_data():
        # Reading from DHT22 and storing the temperature and humidity
        humi, temp = dht.read_retry(dht.DHT22, 23)
        return humi, temp

while True:
        try:
                humi, temp = DHT22_data()

                # If Reading is valid
                if isinstance(humi, float) and isinstance(temp, float):
                        # Formatting to two decimal places
                        humi = '%.2f' % humi
                        temp = '%.2f' % temp

                        # Sending the data to thingspeak
                        conn = urllib2.urlopen(baseURL + '&field1=%s&field2=%s' % (temp, humi))
                        print conn.read()
                        # Closing the connection
                        conn.close()

                else:
                        print 'Error'

                # DHT22 requires 2 seconds to give a reading, so make sure to add delay of
above 2 seconds.
                sleep(20)

        except:
                break
```

# THANK YOU

# Sensing and Sending Sensor Data to Cloud (ThingSpeak)

# Sending Data to ThingSpeak Cloud

```
import sys
import Adafruit_DHT as dht
import urllib2

myAPI = 'IUKV2ZRBQW9MV407Q'
ThingsURL = 'https://api.thingspeak.com/update?api_key=%s' % myAPI

def DHT22_data():
        humidity, temperature = dht.read_retry(dht.DHT22, 23)
        return humi, temp

humidity, temp = DHT22_data()
h = '%.2f ' % humidity
t = '%.2f ' % temp

fields = '&field1=%s&field2=%s' % (t, h)
coms = urllib2.urlopen(ThingsURL + fields)
content = coms.read()
print content
coms.close()
```