

How Arduino YUN can send and get tweets using dweet.io service.

Introduction.

It is now normal to hear about the word IoT.

What does word mean?

The Acronym word "IoT" can be expanded as "Internet of Things".

Similar as when anybody post/get a Tweet using Twitter App, or when anybody follows somebody in Twitter, when we talk about the IoT, we could say that the "tweet" is sent by a machine, and the content of this tweet can be any kind of parameters, values, messages, or any other real about a process that can be shared, to be remotely controlled. The process values could be only read remotely or even modified. This open us many possibilities to interact between remote devices by using dweet.io

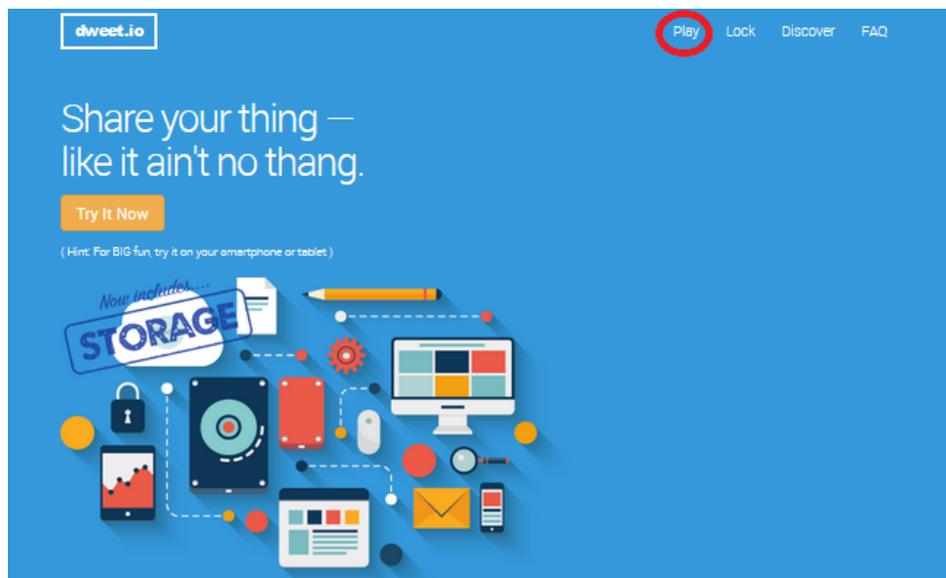
With this short example we will see two methods of how to create a "thing" using the service dweet.io:

1. Create a "thing" using the service dweet.io, directly on their on webpage dweet.io web. The life of the "thing" will be during the computer is on and the web page is open.
2. Create a "thing" inside the code of an Arduino YUN. The life of the "thing" will be during the YUN is powered on and it meanwhile the YUN has WIFI to an access point.

Creating the "thing" at the dweet.io web page for playing.

Steps to create a "thing" at dweet.io:

1. Open <https://dweet.io/> and type Play on the upper right list.



2. Press **POST** and create a **"thing"** named **YUNTEST**, write the **{ }** at field **content** and press **Try-out!**

The screenshot shows the dweet.io API console. At the top, there's a navigation bar with 'dweet.io' and links for 'Play', 'Lock', 'Discover', and 'FAQ'. Below that, a heading says 'Play with dweet.' and a sub-heading says 'Click on one of the operations below in our API console to play with dweet.io.' There are two main sections: 'locks : Lock and unlock your things.' and 'dweets : Create or read dweets in short term cache.' The 'locks' section has three GET operations: '/lock/{thing}', '/unlock/{thing}', and '/remove/lock/{lock}'. The 'dweets' section has four GET operations: '/dweet/for/{thing}', '/get/latest/dweet/for/{thing}', '/get/dweets/for/{thing}', and '/listen/for/dweets/from/{thing}'. The first operation in the 'dweets' section, 'POST /dweet/for/{thing}', is highlighted with a red circle.

The screenshot shows the 'Parameters' section for the 'POST /dweet/for/{thing}' operation. It has a table with columns: Parameter, Value, Description, Parameter Type, and Data Type. The 'thing' parameter is set to 'YUNTEST' and is circled in red. The 'key' parameter is empty. The 'content' parameter is set to '{ }' and is circled in red. Below the table, there's a dropdown menu for 'Parameter content type' set to 'application/json'. At the bottom, there's a 'Try it out!' button circled in red.

Parameter	Value	Description	Parameter Type	Data Type
thing	YUNTEST	A unique name of a thing. It is recommended that you use a GUID as to avoid name collisions.	path	string
key		A valid key for a locked thing. If the thing is not locked, this can be ignored.	query	string
content	{ }	The actual content of the string. Can be any valid JSON string.	body	string

The new "thing" YUNTEST is now ready to be read.

Creating a “thing” at the dweet.io just directly inside the Arduino YUN code.

This is the code in an Arduino YUN to create a “thing” called **YUN_ANALOG_IN_DWEETING** in dweet.io using the instruction `client.get` to monitorize two analog inputs.

See complete code for the YUN

```
#include <Bridge.h>
#include <HttpClient.h>
int analogIn = A0; int analogVal = 0; int analogIn1 = A1; int analogVal1 = 0;
void setup() {
  // Bridge takes about two seconds to start up
  // it can be helpful to use the on-board LED
  // as an indicator for when it has initialized
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  Bridge.begin();
  digitalWrite(13, HIGH);
}

void loop() {
  // Initialize the client library
  analogVal = analogRead(analogIn); analogVal1 = analogRead(analogIn1);
  HttpClient client;
  // Make a HTTP request: To send analog input values of A0 and A1
  client.get("http://www.dweet.io/dweet/for/YUN_ANALOG_IN_DWEETING?AnalogInput_A0="+String(analogVal)
 +"&AnalogInput_A1="+String(analogVal1));
  // from the server, read response and print at the console:
  int i=0; // for degugging
  delay(1000);
  while (client.available()) {
    char c = client.read();
  }
  delay(1000);
}
```

This is the meaning of the call to the dweet.io url:

```
client.get("http://www.dweet.io/dweet/for/YUN_ANALOG_IN_DWEETING?AnalogInput_A0="+String(analogVal)+"&AnalogInput_A1="+String(analogVal1));
```

The YUN as an httpClient must send an string composed of the following parts:

`http://www.dweet.io/dweet/for/` → url that must be called

`YUN_ANALOG_IN_DWEETING` → Name of the "thing"

`?AnalogInput_A0=` → Label for the next variable value. In this case the label is "Analoginput_A0"

`+String(analogVal)` → Variable value to associate to the previous label. In this example it corresponds to the variable `analogVal` converted to an String.

`+"&AnalogInput_A1=` → Label for the next variable value. In this case the label is "Analoginput_A1". **IMPORTANT IS THE CHARACTER &** this is needed when we want to add more than one variable to the same "thing", in our case this character is used only once, but it can be added as many times we need if we want to add 5 variable to the same "thing"

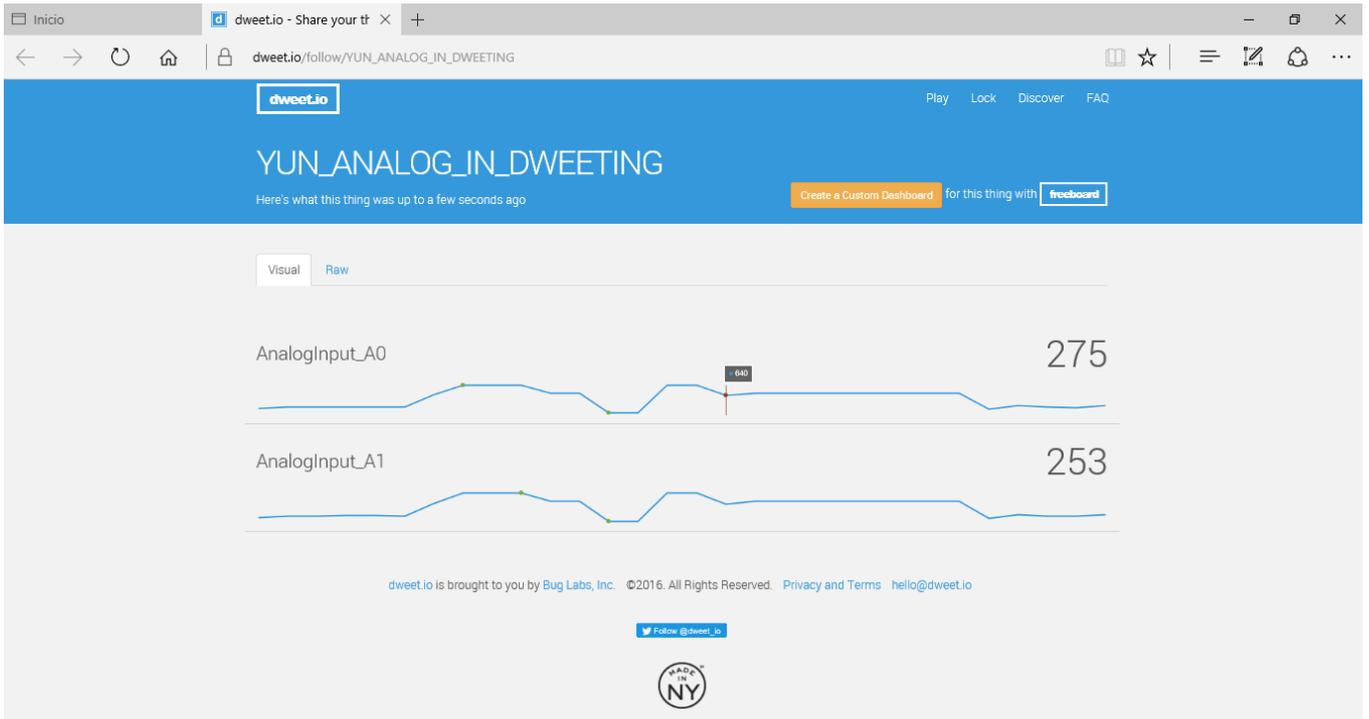
`+String(analogVal1)` → Variable value to associate to the previous label. In this example it corresponds to the variable `analogVal1` converted to an String.

`Client.get(" string ");` → This is the instruction that needs to send one string to the http service in our case to the service `dweet.io`

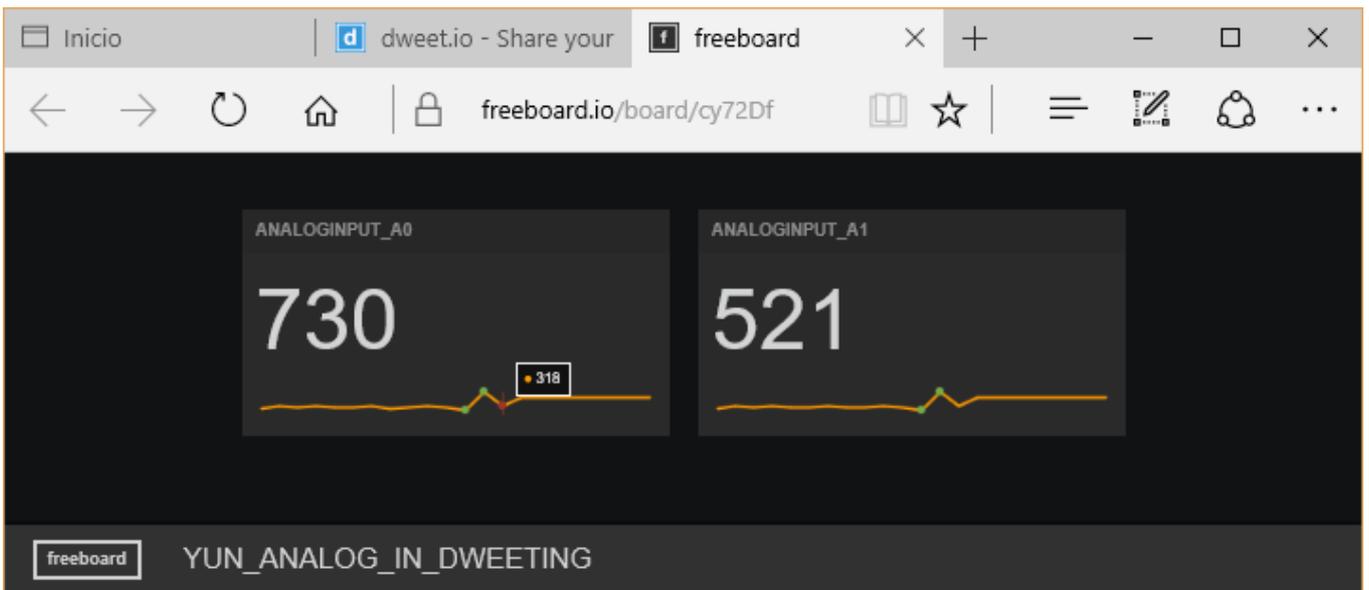
Reading the dweet sent by the YUN in a remote computer

(Arduino YUN must be connected to a WIFI access point).

https://dweet.io/follow/YUN_ANALOG_IN_DWEETING



Select "Create a Custom Dashboard" to obtain a simple panel with the parameters of the "thing" automatically.



<https://freeboard.io/board/cy72Df>

Autor: rjuarez7@gmail.com

Issued: March 2016