



## Application Note: OP-AN-0007 ThingSpeak & parse library

**Device:** Openpicus Flyport

**IDE version:** 2.1

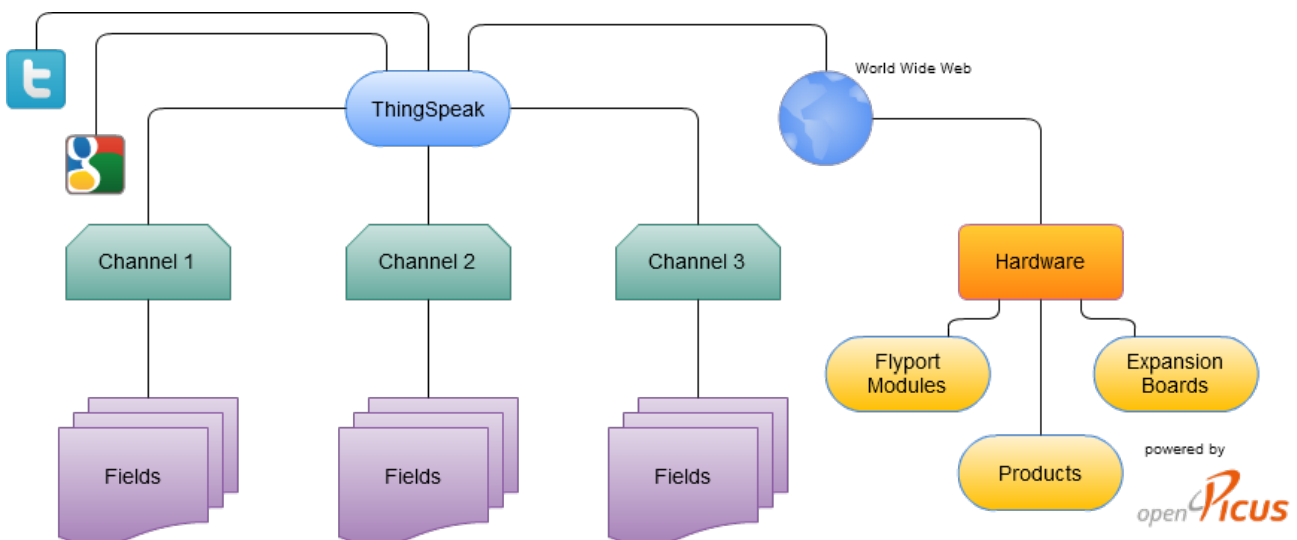
**Author - version:** Simone Marra - v1.0

**External libs:** "Thingspeak" and "parse"

**Connections:** 4 Analog inputs, 4 digital inputs.

### Description:

This Application Note gives access to the Thingspeak services. Thingspeak allows to plot on-line charts, to store and recall values and to use specific APIs to operate with Twitter, or to send HTTP requests directly from the thingspeak servers.



Every thingspeak user can create private or public channels. Every channel can store up to 8 fields (the values) and creates charts with those fields. Every channel has a "Channel ID", a "Name", a "Write API Key" and a "Description".

The *Write API Key* is the most important information of the channel, since it permits to upload or download the field data.

Every field has a name that figures inside the Charts. The charts could be easily integrated inside a website or a blog or seen from the thingspeak.com website.

To use thingspeak.com services for this application note, we created a public channel with "Channel ID"=1609 and link <https://www.thingspeak.com/channels/1609>, with "write API ke" = "Z4B61FNZ4XFD3QMD" and named "OP-AN-0007\_ThingSpeak".

**NOTE:** The channel used is a public channel created for this specific Application Note. Flyport's Firmware is connected to this specific Channel only by the use of Write API Key. Changing those keys allows to use any other thingspeak channel, public or private.

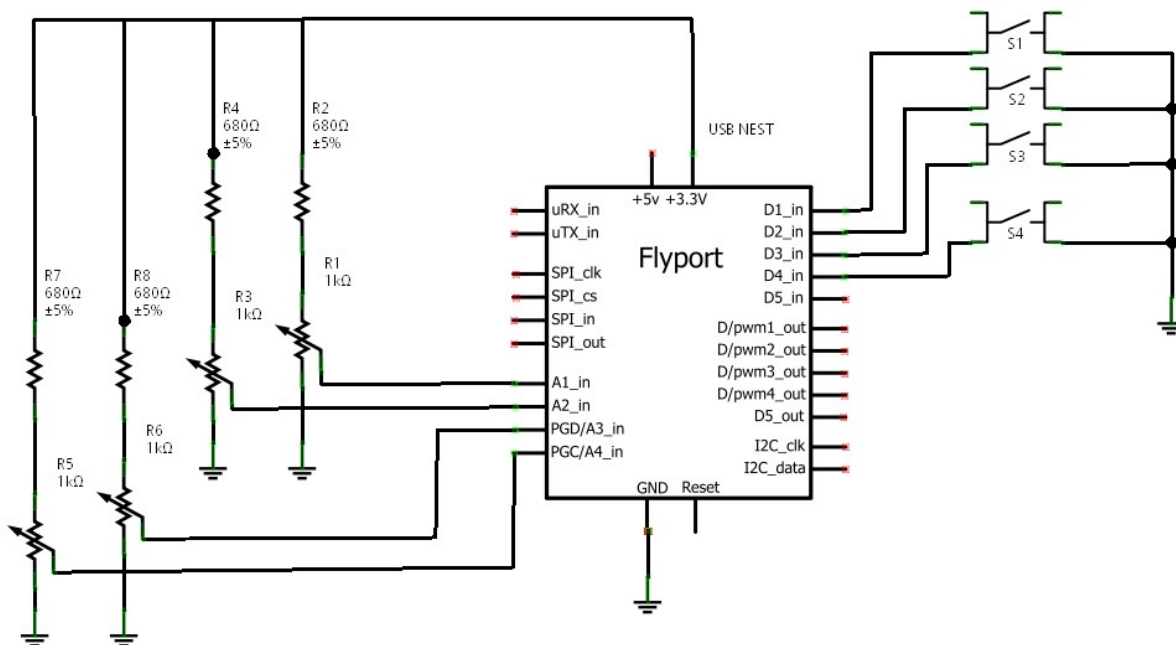
The others Thingspeak services used in this Application Note are thingTweet and thingHTTP.

ThingTweet is a twitter Bridge that *allows* to update twitter status, and thingHTTP permits to send HTTP requests to websites, receiving a pre-parsed reply using the Thingspeak servers. In this example, thingHTTP is used to get the temperature of Rome using the google weather APIs.

### Hardware:

The hardware used is really basic, but it could be easily upgraded with different connections, sensors, etc... Here are used 4 analog inputs with some very simple resistor-pot series to limit the input voltage above 2V, and 4 buttons with internal microcontroller pullup resistors.

The 4 analog values are uploaded to the first 4 fields of thingspeak channel, and the first 3 buttons are uploaded to thingspeak fields 5,6 and 7 as values of 0 for unpressed button and 100 for pressed button. Those values can be freely customized by the user inside the *taskFlyport.c* file.



The switch 4 is responsible of the update of twitter status; if the button is pressed a new message is posted on twitter account.

### Software:

The using of thingspeak services is made easy by the “Thingspeak Library”. It is a ready to use library that allows to write data arrays to channels, or use the thingTWEET and thingHTTP APIs, by the calling of 4 different functions. Those functions automatically prepare the TCP message to send to thingspeak, and handle the receiving of the replies.

To use them it is simply needed to add “thingspeak.h” and “thingspeak.c” reference files from the “External Lib” tool of OpenPicus IDE.

For this Application Note we will use also another library: the parse lib. This helper lib provides some functions to “select” a text inside some complex string like a TCP/IP reply of a webserver. The parse lib is used to take the Temperature in °C from the reply of the thingHTTP function. To use the parse lib, add “parse.h” and “parse.c” with the IDE tool.

In the attached software provided, user have to *change only the Wi-Fi parameters to its router ssid and password*, since a internet connection is needed for the correct function of this firmware. The adhoc connection should not be used. So, simply open the project in the openPicus IDE, launch the Wizard to change connection parameters, compile and download the firmware inside the Flyport.

Another customization could be to change thingspeak's channel, thingTWEET allowed twitter account, and thingHTTP kind of request. Those customizations in Flyport's firmware are done by changing the API keys of thingspeak.com.

Accessing to thingspeak.com website, user can get his own keys and handle all the charts customization as well.

#### Getting the API Keys:

In the taskFlyport.c there are 3 different API keys:

- `char thingKey [] = "Z4B61FNZ4XFD3QMD";`
- `char thingTweetKey [] = "SXHF1RYHZBURVFDO";`
- `char thingHTTPKey [] = "RMB8KBQGSBGL3K2D";`

The channel key *thingKey* is provided by thingspeak.com when the user creates a new Channel, in both public or private mode.

### Channels » Channel 1609 » Edit

<b>Name</b>	<input type="text" value="OP-AN-0007_ThingSpeak"/>	
<b>Description</b>	<input type="text" value="This is the OpenPicus Thingspeak Application Note demo channel"/>	
<b>Tags</b>	<input type="text" value="Flyport, op-an-0007"/>	
<b>Latitude</b>	<input type="text"/>	
<b>Longitude</b>	<input type="text"/>	
<b>Elevation</b>	<input type="text"/>	
<b>Make Public?</b>	<input checked="" type="checkbox"/>	
<b>URL</b>	<input type="text" value="1609/charts"/>	
<b>Video ID</b>	<input type="text"/>	<input type="radio"/> YouTube <input type="radio"/> Vimeo
<b>Field 1</b>	<input type="text" value="AnalogIn1"/>	<input type="checkbox"/> remove field
<b>Field 2</b>	<input type="text" value="AnalogIn2"/>	<input type="checkbox"/> remove field
<b>Field 3</b>	<input type="text" value="AnalogIn3"/>	<input type="checkbox"/> remove field
<b>Field 4</b>	<input type="text" value="AnalogIn4"/>	<input type="checkbox"/> remove field
<b>Field 5</b>	<input type="text" value="DigitalIn1"/>	<input type="checkbox"/> remove field
<b>Field 6</b>	<input type="text" value="DigitalIn2"/>	<input type="checkbox"/> remove field
<b>Field 7</b>	<input type="text" value="DigitalIn3"/>	<input type="checkbox"/> remove field
<b>Field 8</b>	<input type="text" value="Rome temp"/>	<input type="checkbox"/> remove field
<input type="button" value="Update Channel"/>		

To generate the thingHTTP key, simply go to <https://thingspeak.com/apps/thinghttp>, create a new request with:

- URL: <http://www.google.com/ig/api?weather=Roma&hl=en>
- METHOD: GET
- parse string: current\_conditions

In this way an API key will be provided by thingspeak.

To generate the thingTWEET key, simply go to <https://thingspeak.com/apps/thingtweet>, and link a twitter account. Thingspeak will provide API key.

#### Using the thingspeak APIs:

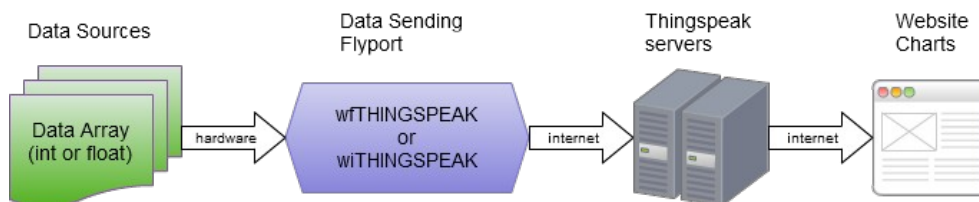
In the taskFlyport.c there are used 3 of the 4 functions of thingspeak library:

- wfTHINGSPEAK
- thingTWEET
- thingHTTP

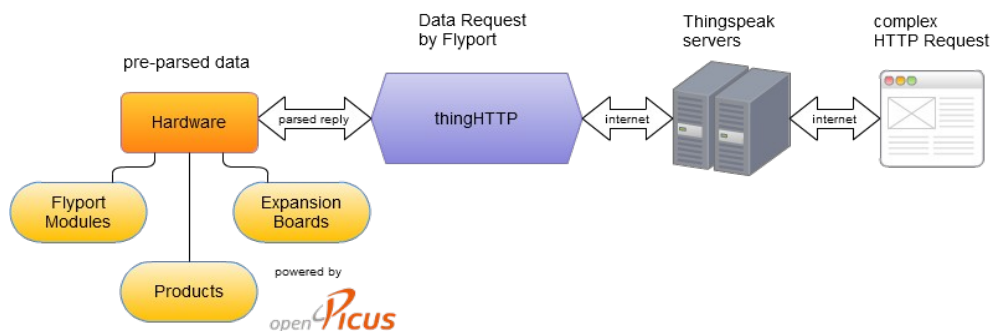
The “*wfTHINGSPEAK*” is a function that writes an array of float data to the specified channel... Thingspeak stores those values and plots them as timed charts, that could be used also outside the thingspeak service inside a blog or a website. In this application note the function is used to write 4 analog inputs and 3 digital inputs values at the first 7 fields of the channel, and the last one field is the current temperature of Rome (the Italian City).

The 4 analog values are read by openPICUS framework, and depend on the state of the potentiometers. The 3 digital values are related to the state of 3 push-buttons (pressed or not pressed). The last value is updated using the thingHTTP feature.

The “*wfTHINGSPEAK*” function uploads float type data, but using the “*wiTHINGSPEAK*” user can uploads int type of data.



The “*thingHTTP*” is a function that sends a TCP request to thingspeak, and receives a parsed reply. This kind of request is very simple, since the message format is just prepared inside the function, the user should only worry about the API key. The request is prepared inside the thingspeak website, and also a parsing string could be used.



For this application note, the thingHTTP request stored inside thingspeak is:

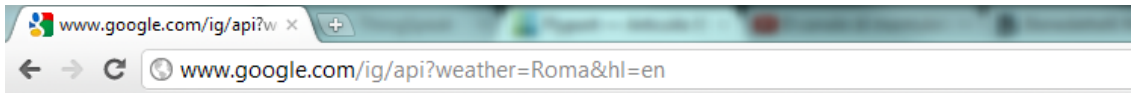
<http://www.google.com/ig/api?weather=Roma&hl=en> where and the parse is done at *current\_conditions* parameter.

### Apps » ThingHTTP 537 » Edit

<b>Name</b>	<input type="text" value="googleWeatherRoma"/>
<b>API Key</b>	RMB8KBQGSBGL3K2D
<b>URL</b>	<input type="text" value="http://www.google.com/ig/ap"/>
<b>HTTP Auth Username</b>	<input type="text" value="http://www.google.com/ig/api?weather=Roma&amp;hl=en"/>
<b>HTTP Auth Password</b>	<input type="text"/>
<b>Method</b>	GET <input type="button" value="v"/>
<b>Content Type</b>	<input type="text"/>
<b>HTTP Version</b>	1.1 <input type="button" value="v"/>
<b>Host</b>	<input type="text"/>
<b>Headers</b>	Name <input type="text"/> Value <input type="text"/> <a href="#">remove header</a> <a href="#">add new header</a>
<b>Body</b>	<input type="text"/>
<b>Parse String</b>	<input type="text" value="current_conditions"/>
<input type="button" value="Update Request"/>	

In this way, Flyport receives only data related to the “current\_conditions”, and the other information are not received by the module, with a high reduction of memory space needed for the GENERIC\_TCP\_CLIENT Rx Buffer (650 bytes are enough). A more precise parse could be done using a parse parameter of “current\_conditions.temp\_c.data” that returns only the temperature in Celsius Degrees.

For a more flexibility of the firmware, it was chosen to parse less information with thingHTTP, and execute a post parse inside the Flyport. The function used is “PARSEbetween”, and the parameter is “temp\_c data=\\”, but changing it to “temp\_f data=\\” permits to have the Fahrenheit Degrees, without changes inside the thingspeak website. Inside Flyport's firmware it could be also parsed any other information returned by thingHTTP function (since these are stored inside a char array...), like the weather condition, the wind or the humidity.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

▼<xml_api_reply version="1">
  ▼<weather module_id="0" tab_id="0" mobile_row="0" mobile_zipped="1" row="0" section="0">
    ▼<forecast_information>
      <city data="Rome, Lazio"/>
      <postal_code data="Roma"/>
      <latitude_e6 data=""/>
      <longitude_e6 data=""/>
      <forecast_date data="2012-01-18"/>
      <current_date_time data="1970-01-01 00:00:00 +0000"/>
      <unit_system data="US"/>
    </forecast_information>
    ▼<current_conditions>
      <condition data="Cloudy"/>
      <temp_f data="43"/>
      <temp_c data="6"/>
      <humidity data="Humidity: 70%"/>
      <icon data="/ig/images/weather/cloudy.gif"/>
      <wind_condition data="Wind: N at 0 mph"/>
    </current_conditions>
    ▼<forecast_conditions>
      <day_of_week data="Wed"/>
      <low data="36"/>
      <high data="52"/>
      <icon data="/ig/images/weather/sunny.gif"/>
      <condition data="Clear"/>
    </forecast_conditions>
    ▼<forecast_conditions>
      <day_of_week data="Thu"/>
      <low data="48"/>
      <high data="52"/>
      <icon data="/ig/images/weather/mostly_sunny.gif"/>
      <condition data="Partly Sunny"/>
    </forecast_conditions>
    ▼<forecast_conditions>
      <day_of_week data="Fri"/>
      <low data="37"/>
      <high data="55"/>
      <icon data="/ig/images/weather/chance_of_rain.gif"/>
      <condition data="Chance of Rain"/>
    </forecast_conditions>
    ▼<forecast_conditions>
      <day_of_week data="Sat"/>
      <low data="37"/>
      <high data="50"/>
      <icon data="/ig/images/weather/sunny.gif"/>
      <condition data="Clear"/>
    </forecast_conditions>
  </weather>
</xml_api_reply>

```

In fact, opening the link <http://www.google.com/ig/api?weather=Roma&hl=en> inside a webbrowser, it could be seen the data available inside the <current\_conditions> ... </current\_conditions> tags, and choose what kind of data to parse inside the Flyport firmware.

The *“thingTWEET”* is the function that allows to update a twitter status with a defined message.

Since twitter needs to have a different message every time the content of the status is updated, inside the message it is written also the date parsed from the reply of the TCPMessage from the thingspeak server. In this way every message is different from another and brings also the date (GMT) of the thingspeak reply.

To launch the update it should be pressed the button 4 at the remaining digital input.

**Conclusions:**

With a minimal effort from user, Flyport module and all the hardware based on openPICUS framework is ready to download updated weather information, upload data to cloud servers, give access to online data charting functionalities and set custom statuses of twitter account.

Thanks to thingspeak library the queries to websites are “transparent” to user firmware, providing an easy to write and easy to read code, just like the serial communications