

Fish'n Tweets

Con questo progetto vogliamo mostrare un'applicazione della scheda **Fishino** che permette in modo molto economico e semplice la gestione remota di apparecchiature elettriche quali lampadine, caldaie, condizionatori, antifurti, eccetera.

L'unicità del sistema è costituita dal fatto di non richiedere apparecchiature di rete o server remoti esterni, ma solo una scheda **Fishino UNO** ed eventualmente una o più schede per la gestione dei carichi applicati. Questo progetto, non realizzabile con un semplice Arduino nemmeno se dotato di shield WiFi o Ethernet, è stato reso possibile grazie alla capacità di **Fishino** di effettuare connessioni sicure con il protocollo HTTPS, disponibili solitamente su apparecchi di livello superiore e decisamente più costosi; per restare nel mondo Arduino, attualmente occorre una YUN per realizzare un progetto equivalente, ed un software decisamente più complesso.

Esistono vari modi per controllare apparecchiature a distanza, ognuna con vantaggi e svantaggi. Vediamone un breve ed incompleto elenco:

- Controllo via SMS :
Vantaggi:
 - Il ricevitore può essere reso portatile ed indipendente da infrastrutture di rete
 - Risposta pressochè istantanea
 - Gestibile da qualsiasi cellulare, anche non troppo recente
 - Portata pressochè illimitataSvantaggi:
 - Richiede una sim sull'apparecchio controllato, che solitamente scade se non usata
 - Richiede un dispositivo GSM, non economicissimo, in ricezione
 - Richiede una copertura GSM, cosa non sempre scontata
- Controllo via moduli radio :
Vantaggi:
 - Il ricevitore può essere reso portatile ed indipendente da infrastrutture di rete
 - Risposta pressochè istantanea
 - Molto economicoSvantaggi:
 - Portata molto limitata
 - La portata è fortemente influenzata da ostacoli
- Controllo via internet :
Vantaggi:
 - Può essere reso portatile al costo di un ricevitore GPRS
 - Può essere controllato pressochè da qualsiasi PC o telefono cellulare
 - Molto economico se presso la destinazione è presente un'infrastruttura di rete
 - Può essere reso facilmente bidirezionale senza costi aggiuntiviSvantaggi:
 - Richiede la presenza di un'infrastruttura di rete in ricezione o un costo aggiuntivo per il GPRS

- La stabilità dipende dalla connessione internet
- I tempi di accesso possono essere variabili a seconda della connessione e/o del metodo utilizzato
- Spesso richiede ricevitori complessi e/o un computer di appoggio
- Richiede un IP pubblico sull'apparecchio ricevente o l'uso di un servizio intermedio di terze parti.
- Possono esserci problemi di sicurezza che richiedono accorgimenti particolari per evitare utilizzi non autorizzati

Con questo progetto abbiamo cercato di unire i vantaggi del controllo via Internet eliminando alcuni degli svantaggi di tale tecnologia, sfruttando un servizio ben noto a tutti : Twitter.

In pratica, i dispositivi vengono azionati inviando un semplice Tweet con un formato particolare, dall'utente preimpostato (o più utenti, con qualche piccola modifica al software) .

Con questo sistema risolviamo le seguenti problematiche del controllo via Internet :

- La stabilità della connessione. Nel caso la connessione sia momentaneamente assente il comando viene solo posticipato ed eseguito al suo ripristino.
- Il ricevitore risulta largamente semplificato, sia dal lato hardware che da quello software.
- Non è richiesto alcun IP pubblico nè un servizio di rete intermedio
- La sicurezza è garantita dall'infrastruttura di Twitter, che lavora con connessioni sicure.

Come per tutte le cose, anche questo sistema non è esente da svantaggi, che sono principalmente la non immediatezza dell'attuazione (il tempo di propagazione di un tweet è di qualche secondo fino ad un paio di decine di secondi) ed alcune limitazioni (aggraviabili) sull'invio di tweets duplicati.

Vediamo innanzitutto l'utilizzazione per poi passare alla descrizione del sistema.

Il controllo avviene tramite l'invio di un Tweet con il seguente formato :

#hashtag porta ON/OFF

ove '**#hashtag**' è un hashtag impostabile nel software (nel nostro caso abbiamo preimpostato **#fishino**), '**porta**' è un numero di porta da attivare (vedremo i dettagli in seguito) e '**ON**' o '**OFF**' sono i comandi di accensione/spegnimento della medesima.

Esempio:

#fishino 24 ON	Accende l'uscita 24
#fishino 10 OFF	Spegne l'uscita 10

Come detto sopra, tra l'invio del tweet e l'esecuzione del comando passa un tempo stimabile in 5-20 secondi, dipendente dalla congestione della rete di Twitter.

L'altro svantaggio di cui parlavamo sopra è l'impossibilità di inviare tweets duplicati in breve sequenza; per esempio:

#fishino 24 ON	Accende l'uscita 24
#fishino 24 OFF	Spegne l'uscita 24
#fishino 24 ON	Accende l'uscita 24

Il terzo tweet non viene accettato ma segnalato con un "hai già inviato questo tweet".
La cosa è ovviabile molto semplicemente aggiungendo qualche carattere a caso dopo la scritta ON:

#fishino 24 ON 123456 Accende l'uscita 24

I caratteri supplementari (**123456**) vengono trascurati dal ricevitore.

Funzionamento

Partiamo dall'invio del Tweet, che viene memorizzato e propagato dalla rete di Twitter. Chiunque può vedere, ricevere e cercare il tweet nella rete da qualsiasi parte del mondo.
Utilizzando un PC o un telefono cellulare si può effettuare la ricerca, per esempio, impostando come parametri il mittente del tweet e l'hashtag:

Ricerca avanzata

Parole

Tutte queste parole ☐

Questa frase esatta ☐

Una di queste parole ☐

Nessuna di queste parole ☐

Questi hashtag

Scritto in

Persone

Da questi utenti

Il risultato della ricerca è qualcosa di simile all'immagine seguente:



Come si nota, vengono mostrati i tweets corrispondenti al criterio di ricerca, dall'ultimo arrivato a quelli precedenti.

Analizziamo un po' più in dettaglio un singolo tweet; ogni messaggio contiene:

- un mittente (**@mdelfede** in questo caso)
- zero o più hashtags (**#fishino** in questo caso)
- un messaggio di testo, comprendente gli hashtags e altro
- un timestamp, ovvero data ed ora di invio
- può contenere riferimenti ad altre persone, immagini, url, eccetera

Nel nostro sistema sono importanti il **mittente**, utilizzato per permettere o negare l'accesso, gli **hashtags**, utilizzati per differenziare un tweet normale da uno di comando al sistema, ed il **testo**, che contiene il comando vero e proprio, ed infine il **timestamp**, o data di invio, che serve per discriminare i comandi già eseguiti da quelli ancora da eseguire.

Ovviamente non possiamo utilizzare una ricerca simile tramite un PC per controllare i nostri dispositivi, ma abbiamo bisogno di qualcosa di più "digeribile" da un sistema automatico, ed il più semplice possibile.

Per questo ci vengono in aiuto le **API** (Application Programming Interfaces) fornite da Twitter che permettono di effettuare, tra le altre cose, delle ricerche sui tweets.

Le API di Twitter

Le **API** in questione sono molto estese, permettendo di effettuare moltissime operazioni, sia di ricerca che di invio di tweets tramite software.

Attualmente, nel nostro controller siamo interessati solo ad una piccola categoria delle medesime, ovvero alle **API** di **ricerca**.

Il funzionamento di tutte le **API** è comunque abbastanza omogeneo e, una volta capiti i meccanismi di un sottoinsieme delle medesime, risulta facilissimo utilizzarle tutte.

Iniziamo con il primo scoglio riscontrato durante lo sviluppo, e che è lo scoglio più grosso che ha obbligato molti progetti analoghi ad utilizzare sistemi molto più complessi di **Fishino** e/o PC di appoggio per la ricerca dei tweets : la sicurezza.

Tweeter utilizza, per rendere sicuro il suo sistema, 2 principali meccanismi :

- la connessione al server delle API DEVE avvenire con protocollo sicuro HTTPS
- l'utente delle API DEVE autenticarsi in uno dei 2 modi che vedremo in seguito

Se il secondo requisito è relativamente semplice, il primo impone per contro l'utilizzo di un sistema in grado di effettuare connessioni con protocollo sicuro HTTPS.

Questo esclude automaticamente gli shields Ethernet/WiFi di Arduino, per esempio, che non sono in grado di utilizzarlo.

Fishino, per contro, è in grado di eseguire connessioni HTTPS anche se col limite di una connessione per volta, vista la grande richiesta di risorse di memoria necessarie per la crittografia.

Vediamo qui di seguito una richiesta https tipica alle API di Twitter, per poi esaminarla in dettaglio :

```
GET /1.1/search/tweets.json?count=1&q=%23fishino+from%3Amdelfede&result_type=recent&max_id=647068135569387520&since_id=646038066440896517 HTTP/1.1
Host: api.twitter.com
User-Agent: Fishino Twitter Client
Authorization: Bearer AAAAAAAAAAAAAAAAAAAAAA(altri caratteri della chiave di autorizzazione)
```

La prima riga è la richiesta vera e propria; esaminiamola in dettaglio:

- **GET /1.1/search/tweets.json?**
è la richiesta alle API di ricerca tweets
- **count=1**
è il numero di risultati richiesti; ne vogliamo 1 per volta

- **q=%23fishino+from%3Amdelfede**
cerchiamo l'hashtag (%23, #) **fishino** proveniente (%3A,@) da **mdelfede**
- **result_type=recent**
vogliamo i risultati in ordine di tempo e non per popolarità
- **max_id=647068135569387520**
gli IDs sono numeri di sequenza contenuti in ogni messaggio Twitter, e che forniscono la sequenza temporale dei medesimi. In questo caso vogliamo cercare un messaggio **non successivo** al numero impostato
- **since_id=646038066440896517**
Vogliamo un messaggio **successivo** al numero impostato
- **HTTP/1.1**
chiude la query HTTP specificando il formato del protocollo

La seconda e la terza riga indicano rispettivamente l'host a cui viene effettuata la richiesta (**api.twitter.com**) e l'applicazione che la sta effettuando (**Fishino Twitter Client**); quest'ultima è arbitraria e serve solo ad identificare l'applicazione che utilizza le **API**.

La quarta riga è molto importante, perchè è quella utilizzata per l'autenticazione della richiesta sulla rete twitter :

Authorization: Bearer AAAAAAAAAAAAAAAAAAAAAA(altri caratteri della chiave di autorizzazione)

La rete Twitter, come detto in precedenza, utilizza 2 metodi per rafforzare la sicurezza delle connessioni : il protocollo HTTPS (che impedisce gli attacchi tipo "man in the middle") e delle chiavi di autenticazione.

Queste ultime sono di 2 tipi : **personali** o di **applicazione (application-only)**.

Con il primo tipo di autenticazione il programma accede all'**API** come se fosse un utente specifico di Twitter ed agisce in suo nome; può, per esempio, inviare tweets come se fossero stati inviati da un PC. Il secondo tipo di autenticazione è legato per contro all'applicazione, quindi parzialmente anonimo. Non permette l'invio di messaggi ma solo alcune operazioni, per esempio la ricerca dei tweets. Nel nostro progetto abbiamo utilizzato il secondo metodo, che da un lato è più limitato, per un motivo molto valido : Twitter pone dei limiti sull'utilizzo delle **API**, limiti temporali; non è possibile eseguire oltre ad un certo numero di operazioni in un determinato intervallo di tempo.

Questi limiti, attualmente sono di **180** operazioni ogni **15 minuti**, separatamente per l'utilizzo come utente o come applicazione. Quindi, se eseguiamo 180 richieste in 15 minuti come applicazione ce ne rimangono altrettante a disposizione come utente.

Superare questi limiti vuol dire rischiare di farsi bloccare l'account Twitter, quindi è opportuno evitare richieste troppo rapide; 180 richieste ogni 15 minuti equivalgono ad un intervallo di 5 secondi tra 2 richieste.

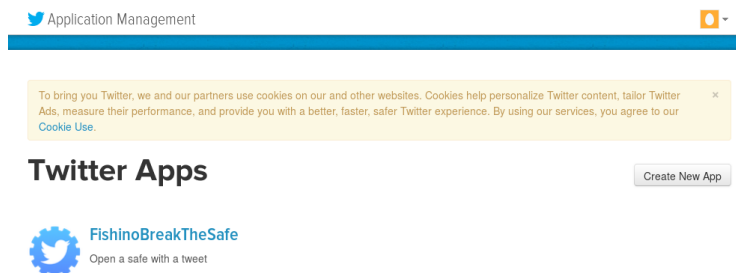
Ottenere una chiave 'Bearer' di autorizzazione

Di seguito spieghiamo come fare a creare un'applicazione Twitter e ad ottenerne la relativa chiave di autorizzazione.

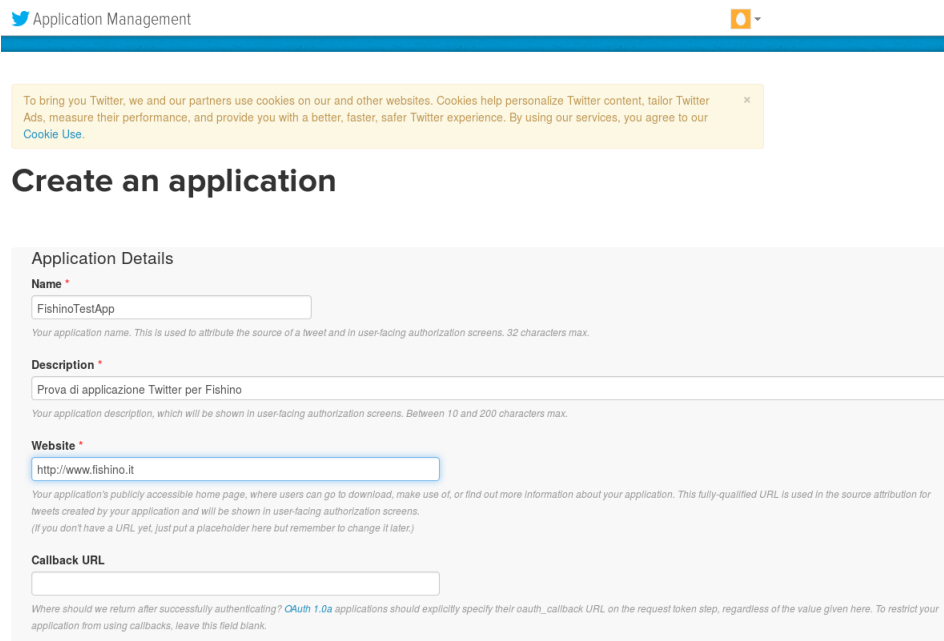
Innanzitutto occorre essere registrati a Twitter, accedere ed andare al link seguente :

<https://apps.twitter.com/>

Vi apparirà una schermata simile alla seguente :



In questo caso si vede un'applicazione già creata (**FishinoBreakTheSafe**) ed il pulsante '**Create New App**'. Premendolo appare la seguente schermata :



Application Management

To bring you Twitter, we and our partners use cookies on our and other websites. Cookies help personalize Twitter content, tailor Twitter Ads, measure their performance, and provide you with a better, faster, safer Twitter experience. By using our services, you agree to our [Cookie Use](#).

Twitter Apps

Create New App

FishinoBreakTheSafe
Open a safe with a tweet

Create an application

Application Details

Name *
FishinoTestApp
Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *
Prova di applicazione Twitter per Fishino
Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *
<http://www.fishino.it>
Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL
Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Qui occorre inserire alcuni campi obbligatori quali :

- **Nome applicazione (Name)**; in questo caso **FishinoTestApp**
- **Descrizione applicazione (Description)**; in questo caso **"Prova di applicazione Twitter per Fishino"**
- **Sito web (Website)** : un sito che dovrebbe contenere i dettagli dell'applicazione. Obbligatorio, ma può essere anche solo un segnaposto; in questo caso **www.fishino.it**
- **Callback URL** : non necessaria per i nostri scopi (lasciare in bianco)

In fondo alla pagina ci sono le usuali condizioni del servizio da accettare (selezionare la checkbox) ed il pulsante per creare l'applicazione (Create your Twitter application) :

Developer Agreement

Effective: May 18, 2015.

This Twitter Developer Agreement ("Agreement") is made between you (either an individual or an entity, referred to herein as "you") and Twitter, Inc. and Twitter International Company (collectively, "Twitter") and governs your access to and use of the Licensed Material (as defined below).

PLEASE READ THE TERMS AND CONDITIONS OF THIS AGREEMENT CAREFULLY, INCLUDING WITHOUT LIMITATION ANY LINKED TERMS AND CONDITIONS APPEARING OR REFERENCED BELOW, WHICH ARE HEREBY MADE PART OF THIS LICENSE AGREEMENT. BY USING THE LICENSED MATERIAL, YOU ARE AGREEING THAT YOU HAVE READ, AND THAT YOU AGREE TO COMPLY WITH AND TO BE BOUND BY THE TERMS AND CONDITIONS OF THIS AGREEMENT AND ALL APPLICABLE LAWS AND REGULATIONS IN THEIR ENTIRETY WITHOUT LIMITATION OR QUALIFICATION. IF YOU DO NOT AGREE TO BE BOUND BY THIS AGREEMENT, THEN YOU MAY NOT ACCESS OR OTHERWISE USE THE LICENSED MATERIAL. THIS AGREEMENT IS EFFECTIVE AS OF THE FIRST DATE THAT YOU USE THE LICENSED MATERIAL ("EFFECTIVE DATE").

IF YOU ARE AN INDIVIDUAL REPRESENTING AN ENTITY, YOU ACKNOWLEDGE THAT YOU HAVE THE APPROPRIATE AUTHORITY TO ACCEPT THIS AGREEMENT ON BEHALF OF SUCH ENTITY. YOU MAY NOT USE THE LICENSED MATERIAL AND

☐ Yes, I agree

Create your Twitter application

Premendo il pulsante appare la pagina seguente :

Your application has been created. Please take a moment to review and adjust your application's settings.

FishinoTestApp


Test OAuth

Details

Settings

Keys and Access Tokens

Permissions



Prova di applicazione Twitter per Fishino
<http://www.fishino.it>

Organization

Information about the organization or company associated with your application. This information is optional.

Organization

None

Organization website

None

Application Settings

Your application's Consumer Key and Secret are used to [authenticate](#) requests to the Twitter Platform.

Access level

Read and write ([modify app permissions](#))

Consumer Key (API Key)

([manage keys and access tokens](#))

Callback URL

None

Callback URL Locked

No

Sign in with Twitter

Yes

App-only authentication

<https://api.twitter.com/oauth2/token>

Request token URL

https://api.twitter.com/oauth/request_token

Authorize URL

<https://api.twitter.com/oauth/authorize>

Access token URL

https://api.twitter.com/oauth/access_token

L'applicazione è stata creata e viene fornita (qui l'abbiamo oscurata) la 'consumer API key', che è una delle chiavi necessarie all'utilizzo delle API.

Poichè noi vogliamo usare la '**Bearer token**', ovvero la chiave necessaria in modalità '**Application only**', serve un ulteriore passo; in alto cliccate sul tab '**Keys and access tokens**'; vi apparirà la schermata seguente :

FishinoTestApp Test OAuth

[Details](#) [Settings](#) [Keys and Access Tokens](#) [Permissions](#)

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)

Consumer Secret (API Secret)

Access Level Read and write ([modify app permissions](#))

Owner mdelfede

Owner ID 229626414

Application Actions

Regenerate Consumer Key and Secret Change App Permissions

Your Access Token

You haven't authorized this application for your own account yet.

By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.

Token Actions

Create my access token

In cui vi si dice che l'applicazione non è ancora stata autorizzata (voce sotto '**Your Access Token**'). Premendo il pulsante '**Create my access token**' in basso si ottiene :

FishinoTestApp Test OAuth

[Details](#) [Settings](#) [Keys and Access Tokens](#) [Permissions](#)

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)

Consumer Secret (API Secret)

Access Level Read and write ([modify app permissions](#))

Owner mdelfede

Owner ID 229626414

Application Actions

Regenerate Consumer Key and Secret Change App Permissions

Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token

Access Token Secret

Access Level Read and write

Owner mdelfede

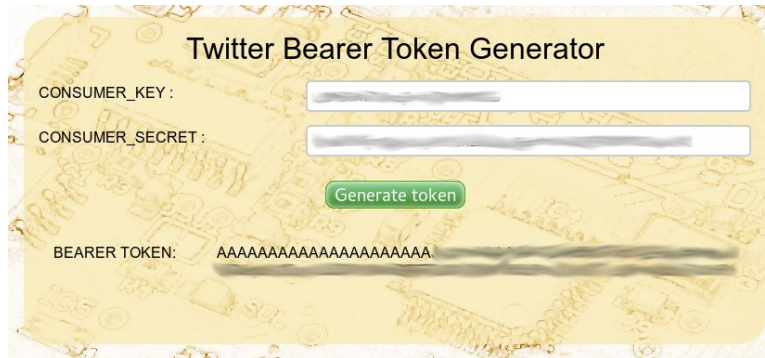
Owner ID 229626414

Sotto la voce '**Your Access Token**' appaiono ora 2 chiavi che vanno utilizzate per l'uso

dell'applicazione in modalità normale (**non Application-only**), e quindi facendola funzionare come se foste voi ad eseguire i comandi. L'ultimo passo per ottenere la chiave **Application-only**, che è poi quella che useremo nella nostra applicazione, sarebbe il più complicato visto che nelle pagine della documentazione delle **API** viene spiegato come fare ma, purtroppo, non viene fornito un sistema per farlo in modo semplice. Abbiamo quindi provveduto a realizzare una pagina all'indirizzo

www.fishino.it/bearer.php

che permette di ottenere in modo facilissimo la cosiddetta '**bearer token**':

The image shows a web form titled "Twitter Bearer Token Generator" on a yellow background with a faint circuit pattern. It contains two input fields labeled "CONSUMER_KEY :" and "CONSUMER_SECRET :". Below these is a green button labeled "Generate token". At the bottom, there is a label "BEARER TOKEN:" followed by a text area containing a long string of 'A' characters, which is partially obscured by a greyed-out area.

È sufficiente inserire la '**consumer_key**' e la '**consumer_secret**' dell'applicazione, ricavate in precedenza, e premere il pulsante '**Generate token**' per ottenere la chiave da copiare ed incollare nello sketch. Ultima nota : la chiave appare sullo schermo in 2 linee per motivi di larghezza di pagina, ma è da copiare ed incollare in **una sola linea**.

Lo sketch

Passiamo ora ad esaminare lo sketch, partendo dall'inizio del file dove ci sono le poche cose da configurare, così da dar modo agli impazienti di provare subito il sistema :

```
////////////////////////////////////
////////////////////////////////////
// CONFIGURATION DATA           -- ADAPT TO YOUR NETWORK !!!
// DATI DI CONFIGURAZIONE -- ADATTARE ALLA PROPRIA RETE WiFi !!!

////////////////////////////////////
////////////////////////////////////
// CONFIGURATION DATA           -- ADAPT TO YOUR NETWORK !!!
// DATI DI CONFIGURAZIONE -- ADATTARE ALLA PROPRIA RETE WiFi !!!

// here put SSID of your network
// inserire qui lo SSID della rete WiFi
#define MY_SSID      "CASA"

// here put PASSWORD of your network. Use "" if none
// inserire qui la PASSWORD della rete WiFi -- Usare "" se la rete non | protetta
#define MY_PASS      "unapassword"
```

```
// here put required IP address of your Fishino
// comment out this line if you want AUTO IP (dhcp)
// NOTE : if you use auto IP you must find it somehow !
// inserire qui l'IP desiderato per il fishino
// commentare la linea sotto se si vuole l'IP automatico
// nota : se si utilizza l'IP automatico, occorre un metodo per trovarlo !
#define IPADDR 192, 168, 1, 251

// here put your application Bearer authorization key
// see https://dev.twitter.com/oauth/application-only for details
// inserire qui la chiave "bearer" di autenticazione dell'applicazione
// vedere https://dev.twitter.com/oauth/application-only per dettagli
#define BEARER "AAAAAAAAAAAAAAAAAAAAAAAA....."

// here put your Twitter user name; if empty the application will react to ANY user
// inserire qui il nome utente ai cui tweets il programma si attiva; se vuoto, il programma si attiva con TUTTI gli utenti
#define TWITTER_USER "pippo"

// here put twitter hashtag needed to trigger the event (without the # char!)
// inserire qui l'hashtag necessario ad avviare l'applicazione (senza il carattere #!)
#define TWITTER_HASHTAG "fishino"

// here put the delay in milliseconds between requests
// there is a limit of about 180 requests every 15 minutes, so don't put a too short time here
// inserire qui il ritardo tra due richieste successive
// c'è un limite di circa 180 richieste ogni 15 minuti, quindi non inserire un tempo troppo breve
#define REQUESTS_INTERVAL 5000
```

Le impostazioni sono poche e semplici; chi già conosce le applicazioni dimostrative di **Fishino** noterà che lo stile è sempre lo stesso ed i contenuti variano di poco; in particolare, verso la fine del gruppo di impostazioni si notano quelle rilevanti per Twitter, ovvero la **Bearer Token** di cui abbiamo parlato al paragrafo precedente (**BEARER**), il nome utente (**TWITTER_USER**) da cui devono provenire i tweets, l'hashtag che attiva il programma (**TWITTER_HASHTAG**), da inserire **SENZA** il carattere di cancelletto (#) e l'intervallo tra 2 richieste al sistema. Per quanto riguarda quest'ultimo, come accennato ad inizio articolo, Twitter pone dei limiti abbastanza stretti sulla frequenza delle richieste; se si superano l'applicazione viene bloccata, a volte temporaneamente ed a volte definitivamente. Attualmente il limite è di 180 richieste ogni 15 minuti, quindi una richiesta ogni 5 secondi circa. Sugeriamo di non inserire un valore inferiore ai 5000 millisecondi impostati se non per brevi prove.

Descrizione dello sketch

Una buona parte dello sketch è comune agli altri esempi presentati nei 2 articoli precedenti sul **Fishino**, e si occupa dell'inizializzazione della scheda, della connessione alla rete WiFi, eccetera. Per brevità rimandiamo a tali articoli per chi fosse interessato ad approfondire e vediamo in dettaglio solo le parti più interessanti dello sketch, ovvero la gestione della connessione sicura alle API di Twitter e l'utilizzo delle schede di espansione IO. Per quanto riguarda la connessione alle API, notiamo subito una differenza rispetto agli sketch visti in precedenza :

```
// The web secure client used to access twitter API
// il client https usato per l'accesso all'API di Twitter
FishinoSecureClient client;
```

Qui si nota l'utilizzo del FishinoSecureClient al posto del precedente FishinoClient. Questo client viene utilizzato per le connessioni sicure HTTPS.

La connessione alle API avviene tramite questo spezzone di codice :

```
if (!client.connected() && !client.available())
{
    Serial.println("\nStarting connection to server...");
    if(client.connect("api.twitter.com", 443))
    {
        Serial.println("connected to server");

        // run the query looking from given hashtag and user
        doQuery(client, "%23" TWITTER_HASHTAG);
    }
}
```

Prima viene controllata se il client è ancora connesso dal loop precedente e/o se ha ancora dei dati da leggere, visto che i dati sono mantenuti in un buffer nel client anche dopo la chiusura dello stesso. In caso contrario viene aperta la connessione al sito api.twitter.com sulla porta 443 che è quella utilizzata nelle connessioni https.

Se la connessione ha esito positivo viene inviata la richiesta vera e propria tramite la funzione doQuery(), che ha come parametri il client e l'hashtag da cercare.

La funzione non fa altro che inviare i dati come visto durante la spiegazione delle API :

```
// run a query on twitter
// esegue la query a Twitter
void doQuery(FishinoSecureClient &client, const char *query)
{
    // if walkback is finished (nextID == -1) we reset sinceID to next found maximum ID
    // otherwise we let it as it is and continue the walk back
    // se il walkback è finito (nextID == -1) re-impostiamo il sinceID a dopo il massimo ID trovato
    // altrimenti lasciamo tutto come sta e continuiamo col walkback
    if(nextID == (uint64_t)-1)
        sinceID = maxID;

    client.print("GET /1.1/search/tweets.json?count=1&q=");
    client.print(query);
    if(strlen(TWITTER_USER))
    {
        client.print("+from%3A");
        client.print(TWITTER_USER);
    }
    client.print("&result_type=recent");
    if(nextID != (uint64_t)-1)
    {
        client.print("&max_id=");
        client << nextID;
    }
    client.print("&since_id=");
}
```

```

client << sinceID;
client.println(" HTTP/1.1");

client.println("Host: api.twitter.com");
client.println("User-Agent: Fishino Twitter Client");

// send auth key to server
// invia la chiave di autorizzazione al server
client.println("Authorization: Bearer " BEARER);
client.println();

// reset nextID - it will be filled by parser if more results are pending
// azzera il nextID - verrà riempito dal parser se ci sono altri risultati da leggere
nextID = -1;
}

```

La particolarità di questa funzione, a parte l'invio dei dati di ricerca veri e propri, sta nella gestione dei campi **since_id** e **max_id** che consentono di restringere la ricerca in ben precisi limiti temporali. Per motivi di spazio e velocità ad ogni richiesta scegliamo infatti di ricevere un solo tweet, che è l'ultimo presente in rete tra i due ids forniti. Giocando sui suddetti IDs è possibile andare a ritroso richiedendo i tweets precedenti, fino all'esaurimento dei medesimi. Finita la ricerca all'indietro viene reimpostato il **since_id** in modo che nelle richieste successive vengano selezionati solo i tweets più nuovi.

Una volta inviata la richiesta, occorre ricevere ed analizzare i dati. Le API di Twitter rispondono con una serie di dati in formato JSON e di dimensioni molto variabili, anche piuttosto grandi. Vista la limitatezza delle risorse di memoria della scheda, una memorizzazione di questi dati è impensabile, ed occorre quindi analizzarli al volo man mano che arrivano. Per far questo abbiamo scritto una libreria completa per il parsing (scansione) dei dati in arrivo in modalità streaming, ovvero carattere per carattere, senza memorizzazioni intermedie. Questa libreria, JSONStreamingParser, è diversa dalla maggior parte delle librerie analoghe presenti in rete visto che, oltre all'analisi dei dati al volo, non memorizza nemmeno i risultati, se non quello corrente.

In pratica, i dati JSON hanno un formato simile a questo :

```

{
  nome1 : dato1,
  nome2 : {nome3 : dato3, nome4 : dato4},
  nome5 : [dato6, dato7, dato8]
}

```

Ovvero, i dati possono contenere valori semplici (stringhe, numeri, eccetera) oppure liste di dati oppure array di dati. Il formato, arrivando ai dati semplici, è comunque sempre lo stesso :

```
nome : dato
```

Il parser quando determina un nome esegue la scansione del dato; se questo è un tipo semplice, chiama una funzione predefinita con i seguenti parametri :

```
void callback(uint8_t filter, uint8_t level, const char *name, const char *value)
```

ove il parametro 'filter' è previsto per future espansioni, il parametro 'level' indica il livello di

nidificazione del dato ed i parametri '**name**' e '**value**' indicano nome e valore del dato. Se il dato NON è semplice ma costituito da una lista o da un array, il campo '**value**' assume il valore "<LIST>" o "<ARRAY>" e la scansione prosegue al suo interno.

Creando opportune funzioni di **callback** è quindi possibile analizzare al volo il contenuto del **JSON** ricevuto ed impostare le variabili opportune nello sketch.

L'attuale funzione di callback è questa :

```
// analyze data and act on devices
// analizza i dati in arrivo da Twitter
void parserCallback(uint8_t filter, uint8_t level, const char *name, const char *value)
{
    // check for user name
    // controlla il nome utente
    if(level == 3 && !strcmp(name, "screen_name") && !strcmp(value, "\"" TWITTER_USER "\""))
        gotUser = true;

    // check twitter message - MUST contain the requested hashtag, a port number and word ON or OFF
    // controlla il testo del messaggio - DEVE contenere l'hashtag richiesto, un numero di porta e le parole ON o OFF
    else if(level == 2 && !strcmp(name, "text") && value && value[0] && !strnicmp(value + 1, "#"
TWITTER_HASHTAG, strlen(TWITTER_HASHTAG) + 1))
    {
        // hashtag found, skip it
        // trovato l'hashtag, lo salta
        const char *p = value + strlen(TWITTER_HASHTAG) + 3;

        // skip spaces after hashtag
        // salta gli spazi dopo l'hashtag
        while(*p && isspace(*p))
            p++;

        // read port number - sets it to 0xff if not found
        // legge il numero di porta - lo imposta a 0xff se non trovato
        if(isdigit(*p))
        {
            port = 0;
            while(isdigit(*p))
                port = port * 10 + *p++ - '0';
        }
        else
            port = 0xff;

        // if port not found, just leave
        // se non trova il numero di port ritorna
        if(port == 0xff)
            return;

        // skip spaces after port number
        // salta gli spazi dopo il numero di porta
        while(*p && isspace(*p))
            p++;

        // chek for ON or OFF keywords - if not found sets port to 0xff and leave
        // cerca le parole chiave ON o OFF - se non trovate imposta la porta a 0xff ed esce
        if(!strnicmp(p, "on", 2))
```

```

        command = true;
    else if(!strnicmp(p, "off", 3))
        command = false;
    else
        port = 0xff;
}

// look if there are previous tweets to handle
// cerca se deve elaborare tweets precedenti
else if(level == 1 && !strcmp(name, "next_results"))
{
    // get next tweet id to handle
    const char *s = strstr(value, "\"?max_id=");
    if(s)
    {
        s += 9;
        nextID = readID(s);
    }
}

// get max id of search, needet to go further after walk back
// cerca il massimo ID corrispondente alla ricerca, necessario per proseguire dopo la ricerca all'indietro
else if(level == 2 && !strcmp(name, "id"))
{
    uint64_t id = readID(value);
    if(id > maxID)
        maxID = id;
}
}

```

La funzione sembra complessa ma in realtà non fa altro che sfruttare i parametri '**level**' e '**name**' per determinare se sono presenti o meno i campi richiesti nel tweet e, nel caso, ne estrae ed analizza i valori. Qui si nota che viene controllato l'**utente** che ha inviato il tweet, gli **hashtags** per vedere se c'è quello impostato, il parametro '**text**' per leggere i comandi inviati e gli **IDs** utilizzati per controllare la sequenza temporale dei tweets.

Come dicevamo, la funzione viene richiamata dal **JSONStreamingParser** ogni volta che questo legge un valore dallo stream. Il parser a sua volta viene utilizzato dalla funzione **doReceive()** :

```

// waits for data from twitter and parse them - return true on sucess, false if no data
bool doReceive(FishinoSecureClient &client, JSONStreamingParser &parser)
{
    // wait 1 second max for data from server
    // attende al massimo 1 secondo i dati dal server
    unsigned long tim = millis() + 1000;
    while(!client.available() && millis() < tim)
        ;

    // if no data is available, close the connection
    // se non ci sono dati, chiude la connessione
    if(!client.available())
        return false;

    // skip data up to 2 consecutive cr
    bool cr = false;

```

```

while(client.available())
{
    if(cr)
    {
        if(client.read() == 0x0d && client.read() == 0x0a)
            break;
        else
            cr = false;
    }
    else if(client.read() == 0x0d && client.read() == 0x0a)
        cr = true;
}

while(client.available())
{
    char c = client.read();
    parser.feed(c);
}
return true;
}

```

Questa funzione non fa altro che leggere i dati dal client, saltare il preambolo HTTP (la cui fine è indicata da una linea vuota) ed inviare i restanti caratteri, uno alla volta, al parser per l'analisi.

Una volta completata l'analisi dello stream di dati, in caso di comando valido, la variabile globale '**port**' conterrà il numero di porta da modificare e la variabile globale '**command**' (booleana) sarà impostata a **true** se vogliamo mettere l'uscita a valore **HIGH** oppure **false** se la vogliamo mettere a **LOW**. Vediamo la parte che attiva l'uscita, interessante per il fatto che prevede la possibilità di collegare gli **IOEXPANDERS** al Fishino :

```

// if port has been set, change its output
// se la porta è stata impostata, ne modifica l'output
if(port != 0xff)
{
    // if no expanders found, check if pin is free on Fishino
    // and set its output (must be changed on Mega!)
    if(!numExpanders)
    {
        if(port == 4 || port == 7 || port >= 10)
            Serial << F("INVALID PORT #") << port << F(" -- COMMAND DISCARDED\n");
        else
        {
            Serial << "\n\nSETTING PORT " << port << " TO " << (command ? "HIGH" : "LOW")
<< "\n";

            pinMode(port, OUTPUT);
            digitalWrite(port, command ? HIGH : LOW);
        }
    }
    else
    {
        uint8_t board = port / 8;
        if(board >= numExpanders)
            Serial << F("INVALID PORT #") << port << F(" -- COMMAND DISCARDED\n");
    }
}

```

```

        else
        {
            Serial << "\n\nSETTING PORT " << port << " TO " << (command ? "HIGH" : "LOW")
<< "\n";

            port %= 8;
            expanders[board]->digitalWrite(15-port, command ? HIGH : LOW);
        }
    }
}

```

qui si nota per prima cosa il controllo sulla validità della porta (port); se è a 0xff il comando è da scartare, altrimenti viene proseguita l'analisi; successivamente il codice agisce in modo differente se sono presenti o meno le espansioni.

Nel primo caso, determina il numero di scheda di espansione (port / 8) e la porta sulla medesima (port %8). Per esempio, un valore di port pari a 33 corrisponde alla scheda n. 4 (33 / 8) ed alla sua porta n.1 (33 % 8 = 33 - 4 · 8 = 1). L'operatore '%' fornisce infatti il resto della divisione.

Nel caso non siano presenti le espansioni, lo sketch utilizza il numero di porta direttamente come I/O digitale del Fishino; controlla che sia un valore valido e che non vada a modificare pins utilizzati per altri scopi ed agisce su quello.

Utilizzo delle schede di espansione IOEXPANDER

Lo sketch prevede, in modo completamente automatico, il riconoscimento di eventuali schede IOEXPANDER anche connesse in cascata, fino ad un massimo di 8. L'importante è avere l'accortezza di fornire indirizzi i2C differenti ad ogni scheda.

All'avvio lo sketch controlla, nella setup(), se sono presenti le schede e le numera in base all'indirizzo i2C. Se ne trova, assegna i ports di uscita in sequenza : da 0 a 7 per la prima scheda, da 8 a 15 per la seconda e così via.

Nel caso non venga rilevata nessuna scheda di espansione, lo sketch va in modalità stand-alone utilizzando tutti gli I/O liberi di **Fishino**, e quindi le uscite **0, 1, 2, 3, 5, 6, 8 e 9**, essendo le altre riservate al funzionamento delle periferiche incluse. In questo caso il numero di porta corrisponde al numero dell'I/O digitale di **Fishino**.

La numerazione delle schede avviene tramite questo spezzone di codice nella setup :

```

// find all available IOExpander boards
void findIOExpanders(void)
{
    numExpanders = 0;
    for(uint8_t addr = 0; addr < 8; addr++)
    {
        IOExpanderMCP23017 *exp = new IOExpanderMCP23017;

        // set expander address
        exp->begin(addr);

        // this is because expander board hasn't reset connected to arduino one
        // and it checks register 0 upon init to see if board is present
        // (register should be 0xff, all A port as input)
        for(int i = 0; i < 16; i++)

```



```

        exp->pinMode(i, INPUT);

    if(exp->init())
    {
        // expander found, add to list and setup pins
        expanders[numExpanders++] = exp;

        // set pin modes
        for(int i = 0; i < 8; i++)
            exp->pinMode(15-i, OUTPUT);
    }
    else
        // not found, delete expander object
        delete exp;
}
if(numExpanders)
    Serial << F("Found ") << numExpanders << F(" I/O expander boards\n");
else
    Serial << F("No expander boards found - using Fishino's I/O pins\n");
}

```

Si nota il ciclo che prova tutti gli indirizzi I2C possibili (da 0 a 7); per ogni indirizzo crea una variabile **IOExpanderMCP23017** (in modo dinamico, come puntatore), ne esegue il test di presenza e, nel caso funzioni la memorizza in un'array di puntatori. Se la scheda non è presente elimina la variabile e riparte con la successiva.

In questo modo le schede vengono rilevate in ordine di indirizzo I2C.

Con questo abbiamo concluso la descrizione dell'applicazione **Fish'n Tweets!**

In un prossimo articolo vedremo come fare l'operazione inversa, ovvero far inviare dei tweets alla scheda quando alcuni pins di ingresso cambiano di stato, cosa utilissima per esempio per realizzare degli allarmi o monitorare eventi in un luogo remoto.