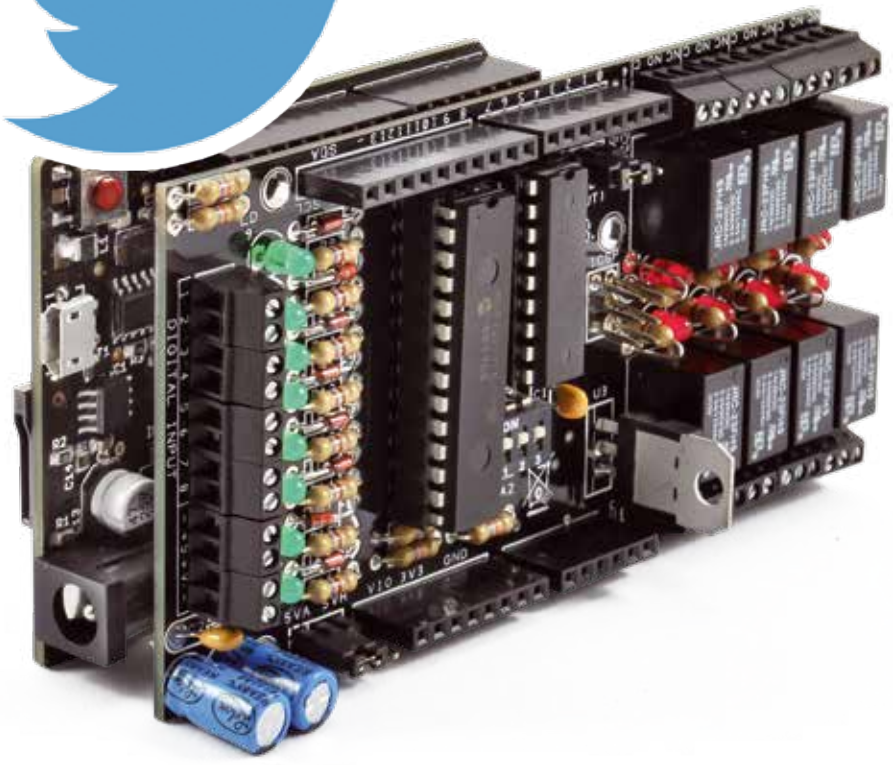
FISH'N TWEET

Controlliamo le uscite di uno shield di espansione Arduino montato sulla nostra Fishino Uno, tramite Twitter.



di MASSIMO DEL FEDELE

Avete conosciuto Fishino, il nostro Arduino dotato di connettività WiFi ed RTC, nel fascicolo n° 199; ora è venuto il momento di vedere un'applicazione pratica della scheda, che permette, in modo molto economico e semplice, la gestione remota di apparecchiature elettriche quali lampadine, caldaie, condizionatori, antifurto, 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 uno o più shield per la gestione dei carichi applicati, che nel nostro caso

Fig. 1

Fig. 2

sono I/O Expander basati su MCP23017 (descritti nel fascicolo n° 181, cui rimandiamo per gli approfondimenti del caso). Questo progetto, non realizzabile con un semplice Arduino nemmeno se dotato di shield WiFi o Ethernet Shield, 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; restare in tema Arduino, attualmente per realizzare un progetto equivalente occorrono una YÙN e un software decisamente più complesso.
Prima di entrare nel merito del progetto vediamo i vari modi per controllare apparecchiature a distanza.

Controllo via SMS
Vantaggi:
- il ricevitore può essere reso portatile ed indipendente da infrastrutture di rete;
- risposta pressoché istantanea;
- gestibile da qualsiasi cellulare, anche non recente;
- portata limitata solo dalla copertura radiomobile.

Svantaggi:
- richiede una SIM sull'apparecchio controllato, che solitamente scade se non usata;
- richiede un dispositivo GSM, non economicissimo, per la ricezione;
- richiede una copertura GSM, cosa non sempre scontata.

Controllo via moduli radio
Vantaggi:
- il ricevitore può essere reso portatile e indipendente da infrastrutture di rete;
- risposta pressoché istantanea;
- molto economico.

Svantaggi:
- portata molto limitata e fortemente influenzata dalla presenza di eventuali ostacoli.

Controllo via Internet
Vantaggi:
- può essere reso portatile al costo di un ricevitore GPRS;
- può essere effettuato pressoché da qualsiasi PC o telefono cellulare;
- molto economico, se presso la destinazione è presente un'infrastruttura di rete;
- può diventare bidirezionale senza costi aggiuntivi e senza particolare difficoltà.

Svantaggi:
- 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, dal lato sia hardware che software;
- non è richiesto alcun IP pubblico né 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 (aggirabili) sull'invio di tweet duplicati.
Vediamo innanzitutto l'utilizzo, 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 tweet 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". A ciò si può ovviare 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.

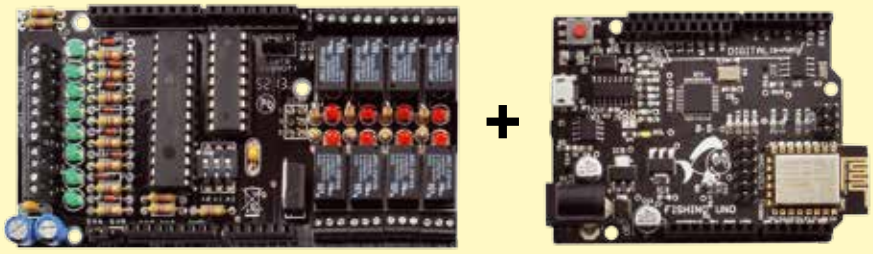
FUNZIONAMENTO

Partiamo dall'invio del tweet, che viene memorizzato e propagato 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 (**Fig. 1**).
Il risultato della ricerca è qualcosa di simile alla **Fig. 2**.
Come si nota, vengono mostrati i tweet 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);
- nessuno, uno o più hashtag

L' **hardware** occorrente

Il progetto si propone di controllare utilizzatori elettrici o gestibili elettricamente tramite l'invio di tweet contenenti le istruzioni per governare i relé di schede di I/O applicabili a Fishino: si tratta di shield FT1079 della Futura Elettronica, sviluppati intorno all'I/O Expander MCP23017 della Microchip. Questo integrato dispone in totale di 16 pin (suddivisi a livello logico in 2 diverse porte A e B) configurabili individualmente come ingresso o uscita. Nello shield gli 8 pin della porta A sono impostati come uscite e gli 8 pin della porta B come ingressi. Le uscite pilotano gli ingressi di un line-driver ULN2803, progettato appositamente per pilotare carichi induttivi come le bobine dei relé (infatti al suo interno dispone di diversi stadi a transistor dotati di diodi di protezione della giunzione base-collettore). Gli ingressi sono invece collegati ad apposite sezioni circuitali che realizzano gli input digitali. Ogni ingresso dispone di un diodo di protezione, una resistenza di pull-up e un LED che ne segnala lo stato. Comunque in questa applicazione non usiamo gli input.
La gestione dell'MCP23017 avviene da I²CBus, realizzato con i piedini 12 (SCL) e 13 (SDA), i quali sono collegati ai relativi pin di Fishino.
L'MCP23017 offre la possibilità di generare un interrupt a fronte della variazione di almeno un pin programmato come input; nel nostro caso non abbiamo sfruttato questa funzione perché non ci serve acquisire lo stato degli ingressi. L'indirizzo I²C dell'MCP23017 viene selezionato tramite il dip-switch SW1.
Lo shield si alimenta tramite il Vin di Fishino, grazie ad un regolatore interno, basato su un L78L05, che ricava 5 V stabilizzati; per attivare questa opzione, il jumper JP5V va chiuso su 5VR. Chiudendolo su 5VA, lo shield riceve i 5 V direttamente dalla linea +5V di Fishino; ciò è possibile solo se il numero di schede è limitato a due.



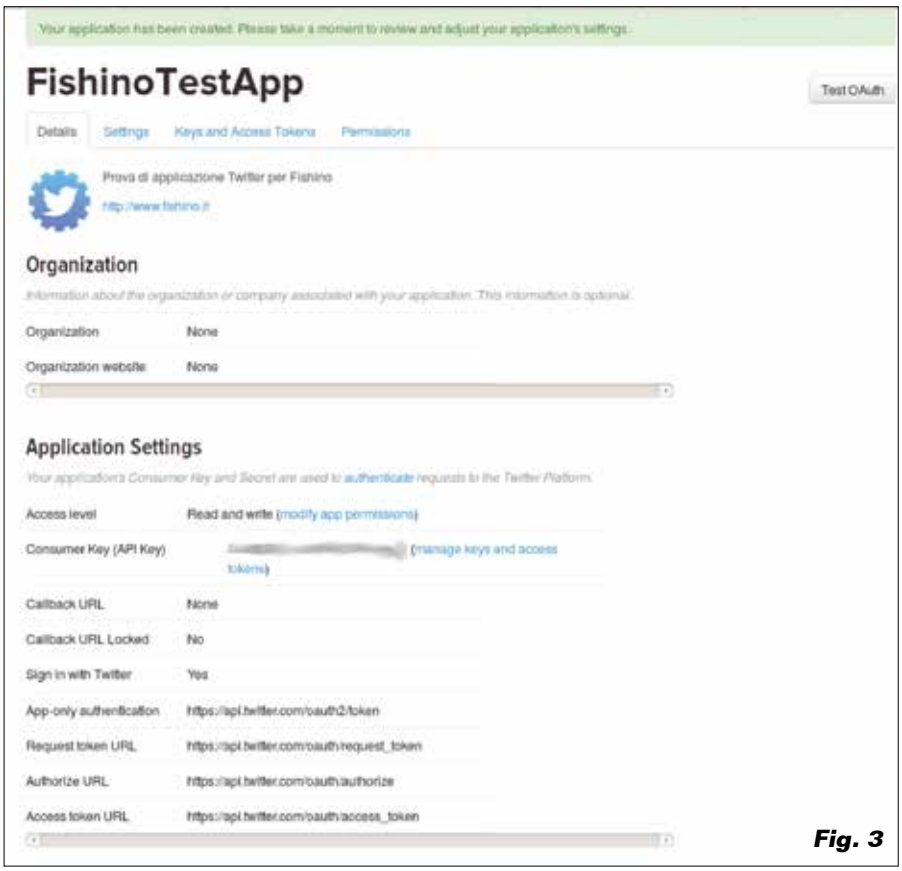


Fig. 3

(**#fishino** in questo caso);
- un messaggio di testo, comprendente gli hashtag e altro;
- un timestamp, ovvero data ed ora di invio.

Il tweet può anche contenere riferimenti ad altre persone, immagini, URL, eccetera.
Nel nostro sistema sono importanti il **mittente**, utilizzato per consentire o negare l'accesso, gli **hashtag**, utilizzati per differenziare un tweet normale da uno di comando al sistema, il testo, che contiene il comando vero e proprio, ed infine il **timestamp**, o data e ora 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 tweet.

LE API DI TWITTER

Le API in questione sono molto estese, permettendo di effettuare moltissime operazioni, sia di ricerca che di invio di tweet tramite software. Attualmente, per il nostro controller ci interessano solo le API di ricerca.
Il funzionamento di tutte le API è comunque abbastanza omogeneo e, una volta capiti i meccanismi di un loro sottoinsieme, risulta facilissimo utilizzarle.
Iniziamo con il superare il primo scoglio incontrato 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 tweet: la sicurezza.
Twitter utilizza, per rendere sicuro il suo sistema, principalmente due meccanismi i quali prevedono che:
- la connessione al server delle API debba avvenire con protocollo sicuro https;
- l'utente delle API debba autenticarsi in uno dei due modi che vedremo in seguito.

Se il secondo requisito è relativamente semplice da soddisfare, il primo impone -per contro- l'utilizzo di un sistema in grado di effettuare connessioni con protocollo sicuro HTTPS. Questo esclude automaticamente gli Ethernet Shield e WiFi Shield di Arduino, per esempio, che non sono in grado di utilizzarlo. Invece Fishino è in grado di eseguire connessioni https anche se col limite di una connessione per volta, vista la grande richiesta di risorse di memoria necessaria per la crittografia.
Vediamo, nel **Listato 1**, una

Listato 1

```
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 AAAAAAAAAAAAAAAAAAAA(altri caratteri della chiave di autorizzazione)
```

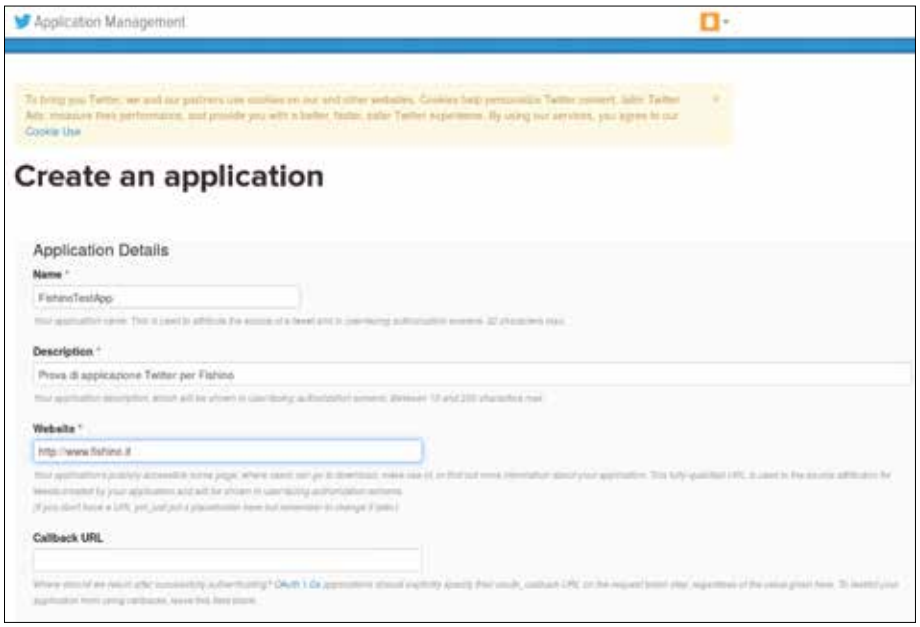


Fig. 4

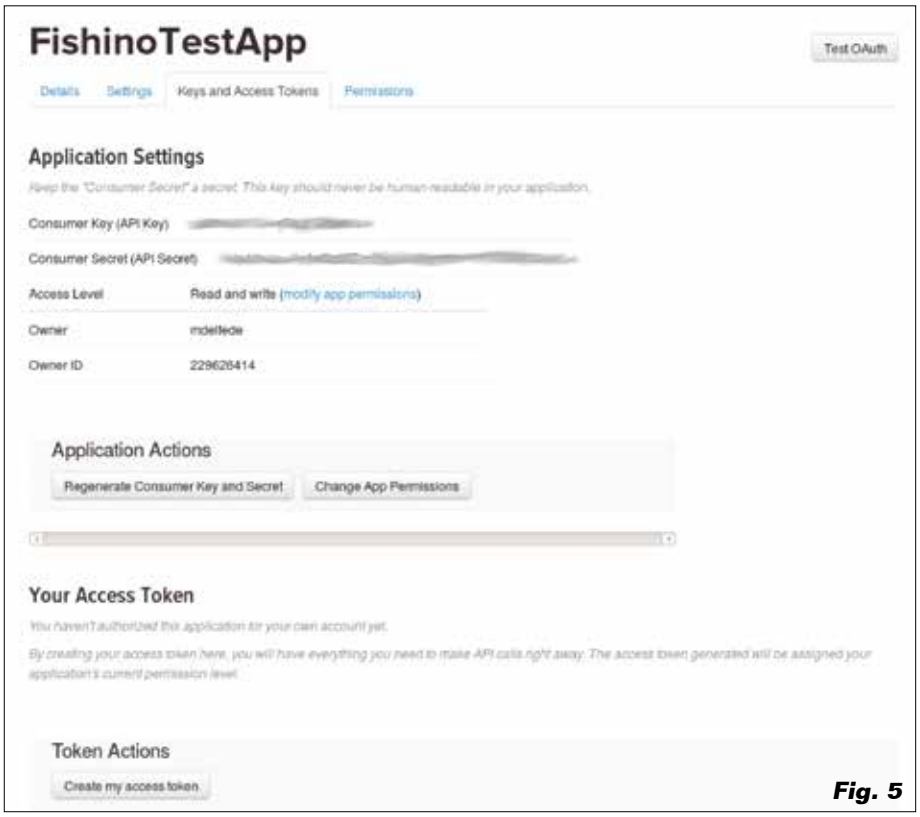
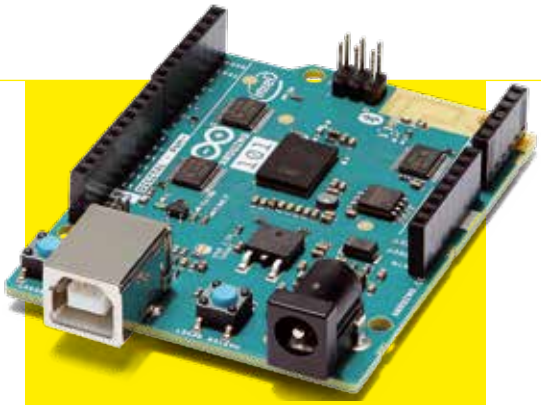


Fig. 5

richiesta https tipica alle API di Twitter, che andremo a esaminare in dettaglio. La prima riga è la richiesta vera e propria:

- **GET /1.1/search/tweets.json?** è la richiesta alle API di ricerca dei tweet;
- **count=1**: è il numero di risultati richiesti; ne vogliamo uno per volta;
- **q= %23fishino+from %3Amdelfede** significa che cerchiamo l'hashtag (%23, #) fishino proveniente (%3A,@) da mdelfede;
- **result_type=recent** significa



Arduino sposa Intel

All'ultimo Rome Maker Faire, Intel e Massimo Banzi hanno presentato Genuino 101, una scheda di sviluppo pensata per la scuola, il mondo dei maker e per quanti si avvicinano per la prima volta alla programmazione. Genuino 101 è basata su Intel Curie, un SoC a basso consumo dalle dimensioni ridottissime, nato per applicazioni wearable; esso implementa un modulo Bluetooth LE e un accelerometro giroscopio a 6 assi. Il core è un Quark x86 a 32 bit con clock a 32 MHz nel quale gira un sistema operativo RTOS ViperOS. La board utilizza anche un core ARC (Argonaut RISC Core) a 32 bit anch'esso con clock a 32 MHz. Per il resto la basetta ha lo stesso form-factor di Arduino UNO e dispone di 14 pin digitali di I/O (di cui 4 possono essere utilizzati come uscite PWM), 6 ingressi analogici, un connettore USB per la comunicazione seriale e per il caricamento degli sketch, un jack di alimentazione e un connettore ICSP con SPI e I²C. Le linee di I/O funzionano a 3,3V ma sono protette contro le sovratensioni fino a 5 V. Genuino 101 sarà adottata da Creative Technologies in the Classroom, programma di physical computing attualmente implementato in oltre 300 scuole.

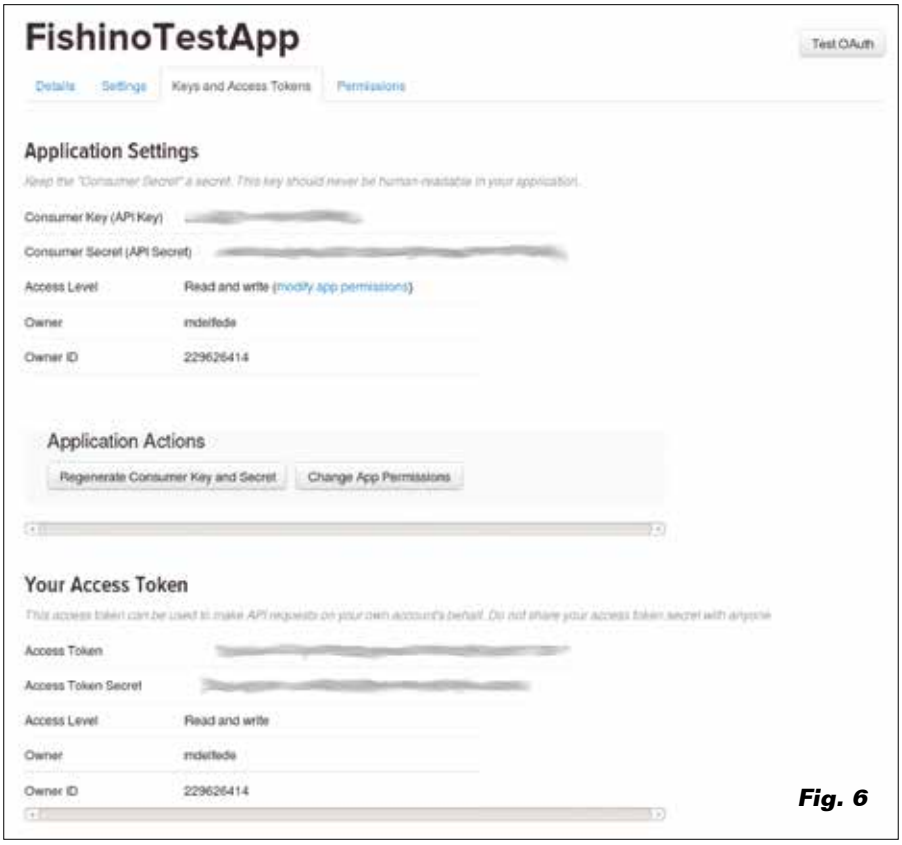


Fig. 6

che vogliamo i risultati in ordine di tempo e non di popolarità;

- **max_id= 647068135569387520**; gli ID 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= 46038066440896517** vuol dire che 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 cui viene inviata 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, perché è quella utilizzata per l'autenticazione della richiesta sulla rete twitter:

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

Come già detto Twitter utilizza due 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 due 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 tweet come se fossero stati inviati da un PC. Il secondo tipo di autenticazione è invece legato all'applicazione, quindi parzialmente anonimo; non permette l'invio di messaggi ma solo alcune operazioni: per esempio la ricerca dei tweet. Nel nostro progetto abbiamo utilizzato il secondo metodo, che da un lato è più limitato perché Twitter pone dei limiti temporali all'utilizzo delle API: non è possibile eseguire oltre 180 operazioni ogni 15 minuti come utente o come applicazione. Quindi, se eseguiamo 180 richieste in 15 minuti come applicazione ce ne rimangono altrettante a disposizione come utente. 180 richieste ogni 15 minuti equivalgono ad un intervallo di 5 secondi tra 2 richieste.
Superare questi limiti vuol dire rischiare di farsi bloccare l'account Twitter, quindi è opportuno evitare richieste troppo frequenti.

Fig. 7

Listato 2

```
////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
// 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 pur 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 "AAAAAAAAAAAAAAAAAAAAA.........."

// 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 requsts
// 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
```

OTTENERE UNA CHIAVE 'BEARER' DI AUTORIZZAZIONE

Vediamo come creare un'applicazione Twitter e ottenere la relativa chiave di autorizzazione. Innanzitutto occorre essere registrati a Twitter e aprire l'indirizzo *https://apps.twitter.com/*. Facendo clic sul pulsante Create New App appare la schermata di **Fig. 3**. Qui occorre riempire 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)** è un sito che dovrebbe contenere i dettagli dell'applicazione; è obbligatorio, ma può essere anche solo un segnaposto; in questo caso è *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) e il pulsante per creare l'applicazione (Create your Twitter application); cliccando appare la pagina di **Fig. 4.**
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. Poiché 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 in **Fig. 5**, che informa che l'applicazione

Listato 3

```
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);
          }
       }
```

Listato 4

```
// 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;
}
```

non è ancora stata autorizzata (voce sotto '**Your Access Token**'). Facendo clic sul pulsante 'Create my access token' in basso si ottiene la schermata di **Fig. 6**.
Sotto la voce '**Your Access Token**' appaiono ora due chiavi che vanno utilizzate per l'uso dell'applicazione in modalità normale (**non Application-only**), vale a dire 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**':
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 due linee per motivi di larghezza di pagina, ma è da copiare e incollare in una sola linea.

LO SKETCH PER FISHINO

Passiamo ora ad esaminare lo sketch (**Listato 2**) partendo dall'inizio del file, dove abbiamo scelto di configurare pochi elementi per dare modo agli "impazienti" di provare subito il sistema.
Le impostazioni sono poche e semplici; chi già conosce le applicazioni dimostrative di Fishino noterà che lo stile è sempre lo stesso e 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 tweet, l'hashtag che attiva il programma (**TWITTER_HASHTAG**), da inserire senza il carattere di cancelletto (#) e l'intervallo tra due richieste al sistema. Per quanto riguarda quest'ultimo, per rispettare il mimite di 180 richieste in 15 minuti abbiamo impostato una richiesta ogni 5 secondi circa (#define REQUESTS_INTERVAL 5000). Suggeriamo di non scendere sotto il valore 5.000 (millisecondi) se non per brevi prove.

DESCRIZIONE DELLO SKETCH

Una buona parte dello sketch si occupa dell'inizializzazione della

Listato 5

```
// 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;
     }
}
```

Listato 6

```
// 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;
}
```

Listato 7

```
    // 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);
         }
      }
    }
```

scheda, della connessione alla rete WiFi, eccetera.
Per brevità rimandiamo agli articoli di presentazione di Fishino; qui 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 I/O.
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 lo spezzone di codice contenuto nel **Listato 3**: prima viene controllato 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 (**Listato 4**).
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 ID forniti; giocando su questi ultimi è possibile andare a ritroso richiedendo i tweet 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 tweet 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; occorre quindi analizzarli 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 in tempo reale, 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
```

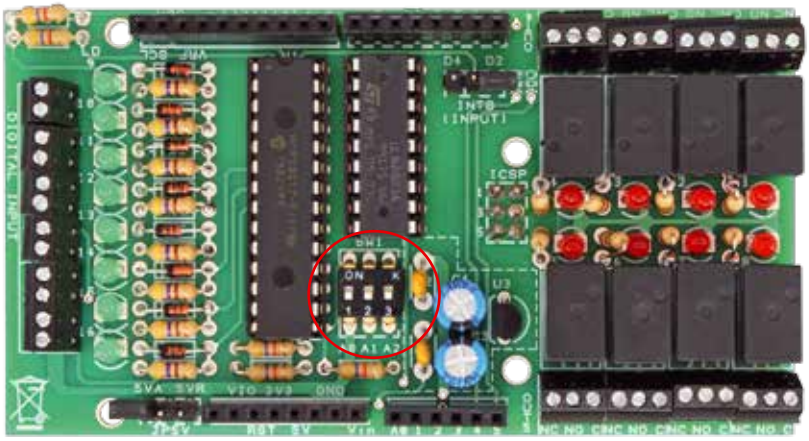
Listato 8

```
// 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");
}
```

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
```

dove 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 contenuta nel **Listato 5**.
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 hashtag per vedere se c'è quello impostato, il parametro '**text**' per leggere i comandi inviati e gli ID utilizzati per controllare la sequenza temporale dei tweet.
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()**



Con SW1 si imposta l'address I²C dello shield I/O Expander: notate la corrispondenza tra A0, A1, A2 e i singoli dip.

come mostrato nel **Listato 6**. 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 vorremo portare l'uscita a, valore HIGH, oppure false se la vorremo a LOW.

Vediamo la parte che attiva l'uscita, interessante per il fatto che prevede la possibilità di applicare gli I/O Expander a Fishino (**Listato 7**). 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 pin utilizzati per altri scopi ed agisce su quello.

UTILIZZO DELLE SCHEDE DI ESPANSIONE I/O

Lo sketch prevede, in modo completamente automatico, il riconoscimento di eventuali schede I/O Expander 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, nel setup(), se sono presenti le schede e le numera in base all'indirizzo I²C. Se ne trova, assegna i port 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 alcuna scheda di espansione, lo sketch va in modalità standalone 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 lo spezzone di codice contenuto nel setup (**Listato 8**).

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 tweet alla scheda quando alcuni pin di ingresso cambiano di stato, cosa utilissima per esempio per realizzare degli allarmi o monitorare eventi in un luogo remoto.

per il MATERIALE

Tutto il materiale necessario per realizzare questo progetto è disponibile presso Futura Elettronica. Lo shield Arduino I/O expander in kit (cod. FT1079K) è disponibile a Euro 34,00. La board Fishino (cod. FISHINOUNO) viene fornita montata e collaudata al prezzo di Euro 36,00. Software e sketck sono scaricabili gratuitamente sul sito *www.elettronicain.it*. I prezzi si intendono IVA compresa.

Il materiale va richiesto a:
Futura Elettronica, Via Adige 11,
21013 Gallarate (VA)
Tel: 0331-799775 • Fax: 0331-792287
http://www.futurashop.it