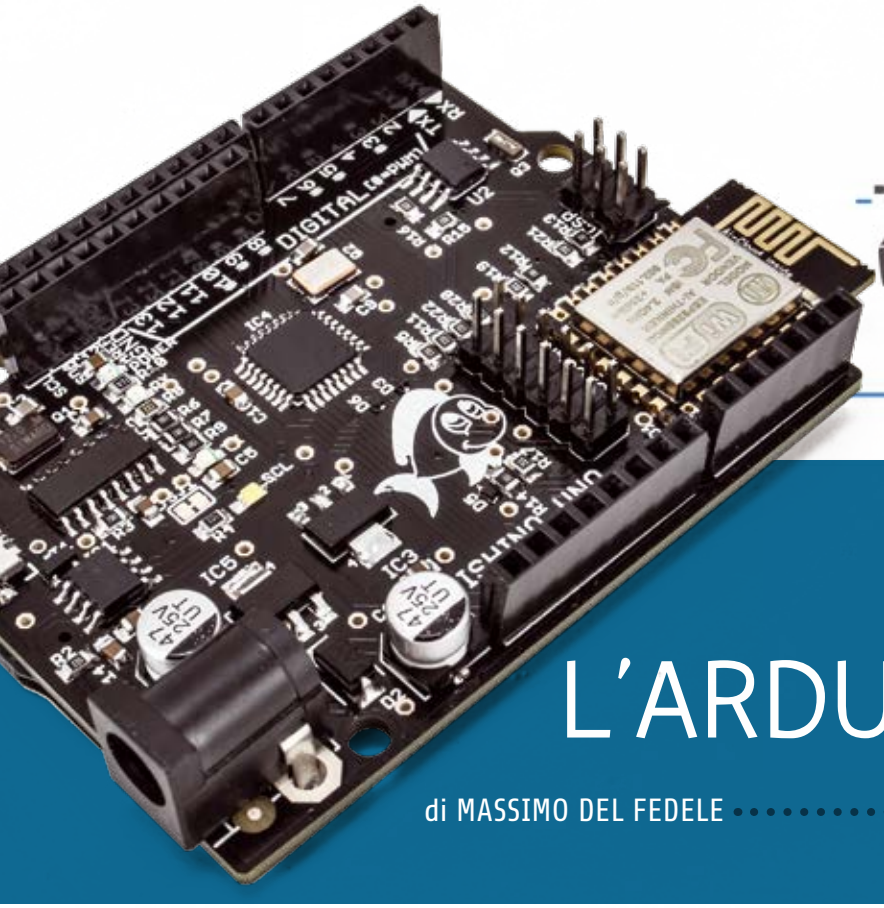
Continuamo la presentazione della board Fishino, mostrando le principali funzioni delle librerie e degli esempi d'uso. Seconda puntata.



FISHINO, L'ARDUINO DIVENTA WIRELESS

di MASSIMO DEL FEDELE

Nel numero scorso abbiamo presentato la scheda **Fishino UNO**, una board compatibile con la diffusissima Arduino UNO dotata però di connettività WiFi, slot per microSD ed RTC incorporati.
In questo secondo articolo iniziamo la descrizione delle librerie software disponibili, mostrando le principali funzioni con alcuni semplici esempi d'uso.
Come anticipato, sia il firmware della scheda che le librerie software sono in continua fase di sviluppo, quindi consigliamo di eseguire spesso l'aggiornamento di entrambi.

LE LIBRERIE

Per poter sfruttare tutte le caratteristiche di **Fishino** occorre ovviamente disporre di una serie di librerie software che ne gestiscano tutti i componenti aggiuntivi.
Se per la scheda SD card e il Real Time Clock (RTC) esistono già nella suite di Arduino le corrispondenti librerie, questo non vale per il modulo WiFi ESP12, per il quale ne abbiamo sviluppate di apposite.
Inizieremo quindi da queste ultime, fornendo comunque successivamente anche qualche dettaglio su quelle già disponibili nell'IDE.

Le librerie fornite e liberamente scaricabili dal sito sono:

- Libreria '**Fishino**'
- Libreria '**FishinoWebServer**'
- Libreria '**Flash**'

quest'ultima libreria, che abbiamo inserito per comodità nel download pur essendo reperibile in rete, è necessaria per il funzionamento delle due precedenti.

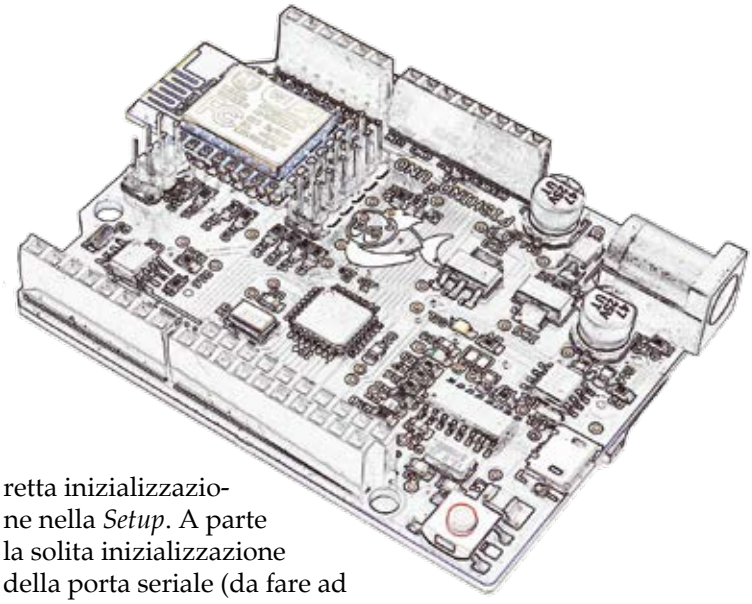
LIBRERIA 'FISHINO'

Partiamo con la descrizione della libreria (che potete scaricare dal sito della rivista *www.elettronicain.it*) che contiene tutta la gestione a basso e medio livello del modulo

WiFi di **Fishino**.
Questa definisce 3 classi:

- **FishinoClass** e la relativa variabile globale Fishino
- **FishinoClient**
- **FishinoServer**

Il progetto è in continua evoluzione e ben supportato da una comunità molto attiva anche su FaceBook (*https://www.facebook.com/groups/fishino*); anche grazie a questa in seguito verranno aggiunte la **FishinoUdp**, **FishinoAnalog**, **FishinoDigital** e **FishinoSerial** per gestire rispettivamente le comunicazioni internet tramite socket Udp, l'ingresso analogico, gli I/O digitali e la porta seriale hardware aggiuntiva presenti sul modulo WiFi.
Iniziamo la descrizione della classe **FishinoClass** (istanza singola nella variabile globale **Fishino**), con degli esempi pratici di utilizzo delle varie funzioni.
Con **bool Fishino.reset()** si inizializza il modulo WiFi inviandogli un reset software. Obbligatorio ad inizio sketch per garantire un avvio corretto del modulo. Ritorna TRUE se il modulo è stato correttamente inizializzato, FALSE altrimenti.
La funzione di reset esegue inoltre un controllo sulla versione del firmware installata. In caso di versione troppo datata viene inviato un messaggio di errore alla porta seriale ed il programma viene bloccato. Nel **Listato 1** vediamo un esempio di corretta inizializzazione nella *Setup*. A parte la solita inizializzazione della porta seriale (da fare ad inizio setup), si notano:



- l'inizializzazione dell'interfaccia SPI
- il **Fishino.reset()** dell'inizializzazione del modulo

La prima è stata lasciata volutamente manuale per poter cambiare la velocità di comunicazione, nel caso siano presenti altri shields che utilizzano la stessa interfaccia. In questo caso si è impostata la massima velocità disponibile.
La sezione contenente la chiamata **Fishino.reset()** inizializza il modulo e visualizza un messaggio di corretta inizializzazione sulla seriale o, in caso di problemi, visualizza l'errore e blocca lo sketch.
Attenzione, il modulo WiFi **NON** parte senza que-

Listato 1

```
void setup()
{
  // apre la porta seriale e ne attende l'apertura
  // consigliabile da eseguire come primo comando per poter visualizzare
  // eventuali messaggi di errore sul monitor seriale
  Serial.begin(115200);

  // attende l'apertura della porta seriale.
  // Necessario solo per le boards Leonardo
  while (!Serial)
        ;

  // inizializza il modulo SPI
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV2);

  // resetta e testa il modulo WiFi
  if(Fishino.reset())
        Serial.println("Fishino WiFi RESET OK");
  else
  {
        Serial.println("Fishino RESET FAILED");

        // attende per sempre
        while(true)
              ;
  }

  Serial.println("Fishino WiFi web server");

  .....<resto dello sketch>...
```

Listato 2

```
...<parte precedente dello sketch>...

Fishino.setMode(STATION_MODE);

...<resto dello sketch>...
```

Listato 3

```
...<parte precedente dello sketch>...
while(true)
{
  if(Fishino.begin("MIO_SSID","MIA_PASSWORD")) {
   Serial.println("Connected to MIO_SSID");
   break;
  }
  else {
   Serial.println("Failed connecting to MIO_SSID");
   Serial.println("Retrying.....");
  }
}
...<resto dello sketch>...
```

sto comando.
Le funzioni **bool Fishino.setMode(uint8_t mode)** e **uint8_t Fishino.getMode(void)** impostano (o leggono) la modalità di funzionamento del modulo (**Listato 2**), che può essere una delle seguenti:

- **STATION_MODE**
  modalità stazione. Richiede la presenza di un router WiFi a cui connettersi. È la modalità normale.
- **SOFTAP_MODE**
  Permette la creazione di un access point a cui connettersi. Utile in mancanza di un'infrastruttura di rete esistente.
- **STATIONAP_MODE**
  Modalità doppia, il modulo funziona sia da stazione, collegandosi ad un router esistente, che da access point.

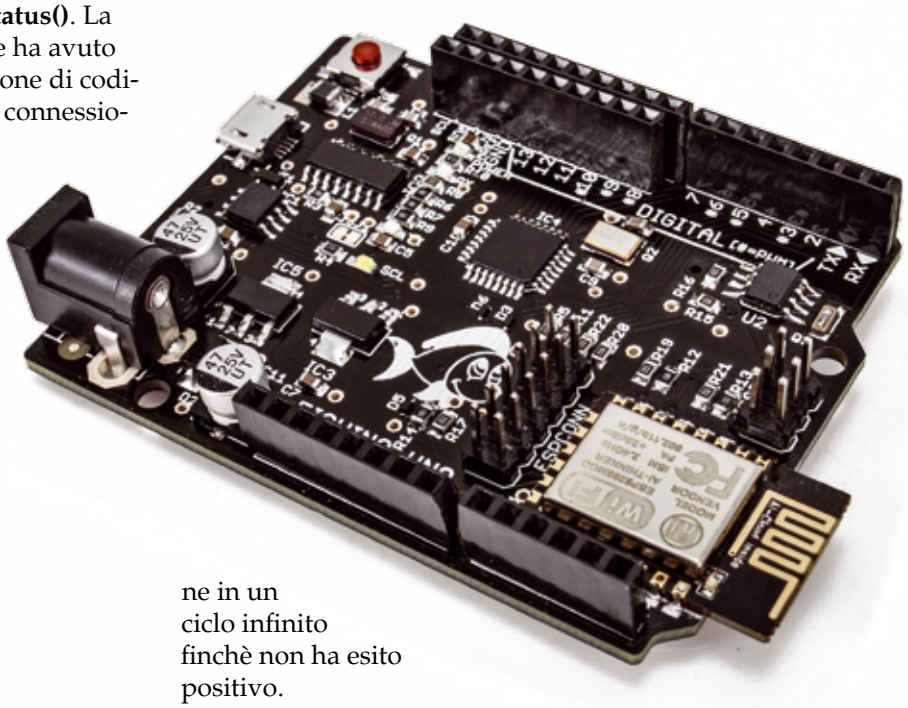
Per eseguire la connessione all'access point/router si utilizza **bool Fishino.begin(SSID, PASSWORD)**, dove al posto di SSID va inserito il punto di accesso e al posto di PASSWORD la chiave per accedervi (quest'ultima può essere una stringa vuota se non è richiesta).
Per controllare se la board Fishino è correttamente connessa il comando è **uin8_t Fishino.status()**. La funzione ritorna TRUE se la connessione ha avuto successo, FALSE altrimenti. Nello spezzone di codice presente nel **Listato 3** viene tentata la connessione in un ciclo infinito finchè non ha esito positivo.
Questo tipo di connessione (eseguita nella *Setup*) è adatto ad una postazione fissa, ovviamente. Nel caso si utilizzi il Fishino in mobilità, è consigliabile spostare la connessione nel *loop()* e tentarla ogni tanto mentre si fanno altre attività.
In questo modo è possibile, ad esempio, raccogliere

Listato 4

```
uint32_t connecTime;
void setup()
{
    .....<parte precedente dello sketch>....
    connectTime = millis();
}

void loop()
{
  // controlla la connessione ogni 10 secondi
  if(millis() - connectTime > 10000) {

    // resetta il tempo
    connectTime = millis();

    // controlla se connesso
    uint8_t stat = Fishino.status();

    // se non connesso, tenta la riconnessione
    if(stat != STATION_GOT_IP) {
        if(Fishino.begin("MIO_SSID", "MIA_PASSWORD"))
            stat = STATION_GOT_IP;
    }

    // se connesso, salva i dati sul server
    if(stat ==  STATION_GOT_IP) {
        salvaDatiSulServer(); // funzione da definire!!!
  }

  // qui legge i sensori e li memorizza sulla SD
  leggiSensoriEMemorizza(); // funzione da definire!!!
}
```

Listato 5

```
Fishino.config(IPAddress(192, 168, 1, 251));
```



CARATTERISTICHE TECNICHE

- **Alimentazione: 12 Vcc o USB**
- **Completamente compatibile con Arduino Uno**
- **Scheda WiFi a bordo, con possibilità di funzionamento in modalità stazione, access point o entrambe contemporaneamente**
- **Interfaccia per schede di memoria MicroSD a bordo**
- **RTC (Real Time Clock) a bordo con batteria al litio di mantenimento**
- **Sezione di alimentazione a 3,3 V potenziata**
- **Connettore aggiuntivo sfalsato in modo da risolvere il problema dell'incompatibilità di Arduino con le schede millefori.**

Listato 6

```
Serial.print("Il mio IP è : ");
Serial.println(Fishino.localIP());
```

dei dati da un sensore, memorizzarli sulla scheda SD e, quando viene rilevata una connessione funzionante, inviarli ad un computer remoto (**Listato 4**). In questo esempio (volutamente abbreviato), nella *setup()* viene letto il tempo corrente e salvato nella variabile *connectTime*; successivamente nel *loop()* viene controllato il tempo passato *(millis() - connectTime)* e quando questo supera i 10 secondi viene eseguito un test sulla connessione; se non connesso si tenta la connessione al server e, in caso di successo viene eseguita una funzione (da definire) che salva in rete i dati letti in precedenza.
Il loop continua successivamente tramite un'altra funzione (anch'essa da definire) che legge qualche sensore e memorizza i dati localmente, ad esempio su una scheda SD.
Con uno sketch simile è possibile quindi realizzare un semplice datalogger che non solo legge e memorizza su scheda SD i dati ma che, in presenza di una connessione di rete, è in grado di salvarli in modo totalmente automatico ad intervalli di tempo predefiniti.
Per configurare un IP statico ed eventualmente i servers DNS, il gateway e la subnet della rete locale si utilizzano queste funzioni:
bool Fishino.config(IPAddress local_ip)
bool Fishino.config(IPAddress local_ip, IPAddress dns_server)
bool Fishino.config(IPAddress local_ip, IPAddress dns_server, IPAddress gateway)
bool Fishino.config(IPAddress local_ip, IPAddress dns_server, IPAddress gateway, IPAddress subnet)
In pratica, la prima è adoperata per impostare un IP statico, se necessario.
Nel **Listato 5** vediamo come è possibile impostare un IP statico su 192.168.1.251
Se non utilizzata l'IP sarà richiesto dinamicamente al router.
E' ovviamente possibile anche disconnettersi dalla rete WiFi. Il comando per eseguire questa operazione è **bool Fishino.disconnect(void)**.

Qui di seguito, invece, alcune funzioni utilizzate per controllare i parametri della connessione, in particolare per leggere il MAC del modulo WiFi la funzione è **const uint8_t* Fishino.macAddress(void)**
Mentre per la lettura dell'IP acquisito dal modulo WiFi (utile nel caso si sia impostato un IP dinamico) il comando da richiamare è **IPAddress Fishino.localIP()** come mostrato ad esempio nel **Listato 6**.
Per leggere la maschera della sottorete e dell'indirizzo IP del gateway potete richiamare queste funzioni:
IPAddress Fishino.subnetMask()
IPAddress Fishino.gatewayIP()
Le funzioni indicate sopra sono state nominate in modo assolutamente simile a quelle delle analoghe funzioni delle librerie Ethernet e WiFi di Arduino,

Listato 8

```
Serial.print("Sono connesso a : ");
Serial.println(Fishino.SSID());
```

per poter semplificare il porting del codice esistente.
Tuttavia le potenzialità superiori di **Fishino**, ed in particolar modo la possibilità di funzionare anche in modalità Access Point senza bisogno di un'infrastruttura esistente, hanno reso necessario studiare nuove funzioni per quanto riguarda la modalità **Stazione** tra cui:
bool Fishino.setStaIP(IPAddress ip)
bool Fishino.setStaMAC(uint8_t const *mac)
bool Fishino.setStaGateway(IPAddress gw)
bool Fishino.setStaNetMask(IPAddress nm)
Mentre per la modalità **Access Point** sono state create:
bool Fishino.setApIP(IPAddress ip)
bool Fishino.setApMAC(uint8_t const *mac)
bool Fishino.setApGateway(IPAddress gw)
bool Fishino.setApNetMask(IPAddress nm)
bool Fishino.setApIPInfo(IPAddress ip, IPAddress gateway, IPAddress netmask)
In particolare, l'ultima permette di impostare tutti i parametri IP del **Fishino** utilizzato come router

Listato 7

```
Fishino.setApIPInfo(
        IPAddress(192, 168, 100, 1),            // IP del Fishino
        IPAddress(192, 168, 100, 1),            // Gateway del Fishino, solitamente come l'IP
        IPAddress(255, 255, 255, 0)             // Netmask (maschera di sottorete) del Fishino

   );
```

Listato 9

```
uint8_t n = Fishino.scanNetworks();
if(n) {
      Serial.print("Trovate ");
      Serial.print(n);
      Serial.println(" reti wifi:");
      for(int i = 0; i < n; i++) {
               Serial.print("Rete #");
               Serial.print(i);
               Serial.print(" : ");
               Serial.println(Fishino.SSID(i));
      }
}
else
      Serial.println("Nessuna rete WiFi trovata");
```

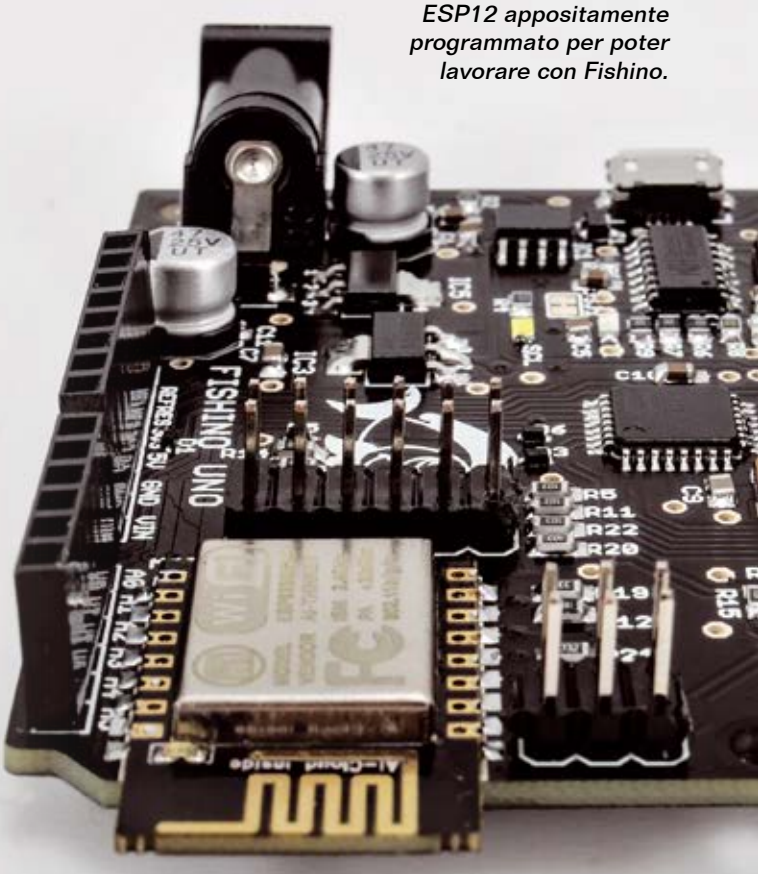
WiFi in un comando singolo (**Listato 7**).
Vedremo come usarle a fine articolo con un esempio completo. Per poter leggere i dati della connessione WiFi, quali lo SSID del router a cui ci si è connessi, il MAC del medesimo (BSSID), la potenza in dBm del segnale (RSSI) ed il tipo di protezione della rete, potete utilizzare queste funzioni:
const char* Fishino.SSID()
const uint8_t* Fishino.BSSID()
int32_t Fishino.RSSI()
uint8_t Fishino.encryptionType()
Come mostrato ad esempio nel **Listato 8**.
Esistono poi alcune funzioni utilizzate per eseguire una lista delle reti WiFi disponibili con le loro caratteristiche:
uint8_t Fishino.scanNetworks()
Questa operazione esegue una scansione delle reti WiFi disponibili e ritorna il numero di reti trovate.
Una volta eseguita la *scanNetworks*, è possibile utilizzare le seguenti funzioni, che hanno come parametro il numero della rete da esaminare (numero di reti ritornate da **scanNetworks()** - 1).
La funzione **const char* Fishino.SSID(uint8_t networkItem)** ritorna invece lo SSID, ovvero il nome della rete richiesta come mostrato nel **Listato 9**.
Questo esempio stampa sulla seriale un'elenco delle reti wireless trovate.
Per sapere il tipo di protezione della rete il comando da usare è **uint8_t Fishino.encryptionType(uint8_t networkItem)**.
E' possibile anche sapere la potenza del segnale della rete richiesta con **int32_t Fishino.RSSI(uint8_t networkItem)**.
Nella classe **FishinoClass** sono presenti altre funzioni meno utilizzate che tralasciamo per brevità.
Il codice della libreria è comunque ben commentato e di facile interpretazione.

Listato 10

```
// tenta la connessione al server
FishinoClient client;
if (client.connect("www.google.com", 80)) {

 Serial.println("connected to server");

 // esegue un request Http
 client.println("GET /search?q=arduino HTTP/1.1");
 client.println("Host: www.google.com");
 client.println("Connection: close");
 client.println();

 // legge la risposta finchè il client resta connesso
 do {
      // finchè ci sono bytes in arrivo.....
      while (client.available()) {

               // legge un carattere dal server
               char c = client.read();

               // .... e lo invia alla seriale
               Serial.write(c);
      }
      }
      while(client.connected());
      Serial.println("Client disconnected");
 }
}
```

CLASSI FISHINOCLIENT E FISHINOSERVER

Queste due classi sono l'equivalente delle EthernetClient/WiFiClient ed EthernetServer/WiFiServer delle shield ethernet e WiFi di Arduino, e l'uso è praticamente identico.
Ad esempio, per inviare una richiesta ad una pagina web e stampare sulla seriale la risposta vediamo il **Listato 10**.



Dettaglio del modulo WiFi ESP12 appositamente programmato per poter lavorare con Fishino.

Listato 11

```
#include <Flash.h>
#include <FishinoUdp.h>
#include <FishinoSockBuf.h>
#include <Fishino.h>
#include <SPI.h>

/////////////////////////////////////////////////////////////////////
//     CONFIGURAZIONE SKETCH -- ADATTARE ALLA PROPRIA RETE WiFi   //
// WiFi SSID e PASSWORD
// potete cambiarle entrambe, verranno utilizzate
// per la creazione dell'infrastruttura WiFi
#define My_SSID       "FISHINO"
#define My_PASS       ""
//                      FINE CONFIGURAZIONE                      //
/////////////////////////////////////////////////////////////////////

// crea un server in ascolto sulla porta 80 (HTTP standard)
FishinoServer server(80);

void setup()
{
  // apre la porta seriale
  Serial.begin(115200);

  // attende l'apertura della porta seriale.
  // Necessario solo per le boards Leonardo
  while (!Serial)
    ;


  // inizializza il modulo SPI
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV2);

  // resetta e testa il modulo WiFi
  if(Fishino.reset())
    Serial << F("Fishino WiFi RESET OK\r\n");
  else
  {
    Serial << F("Fishino RESET FAILED\r\n");

    // attende per sempre in caso di errore
    while(true)
          ;
  }

  Serial << F("Fishino WiFi AP web server\r\n");

  // imposta la modalitè SOFT AP (crea una rete autonoma)
  Fishino.setMode(SOFTAP_MODE);

  // ferma il server DHCP, necessario per impostare l'IP della rete
  Fishino.softApStopDHCPServer();

  // imposta i parametri IP dell'access point
  // in questo caso la rete viene creata su 192.168.100.0-255
  // ed il Fishino assume l'IP 192, 168, 100, 1
  Fishino.setApIPInfo(
    IPAddress(192, 168, 100, 1),                    // IP
    IPAddress(192, 168, 100, 1),                    // gateway
    IPAddress(255, 255, 255, 0)                     // netmask
  );


  // imposta i parametri di connessione WiFi, ovvero nome della rete(SSID)
  // e password. Se non avete modificato l'esempio, la rete sarà chiamata FISHINO
  // e sarà una rete aperta, senza password
  Fishino.softApConfig(My_SSID, My_PASS, 1, false);

  // riavvia il server DHCP in modo da poter fornire gli indirizzi
  // in automatico a tutte le stazioni che si connettono
  Fishino.softApStartDHCPServer();

  // inizia l'attesa delle connessioni
  server.begin();
}


void loop()
{
  // attende nuovi clienti
  FishinoClient client = server.available();

  if (client)
  {
    Serial.println("new client");

    // ogni richiesta http termina con una linea vuota
    boolean currentLineIsBlank = true;
    while (client.connected())
    {
          if (client.available())
          {
                    char c = client.read();
                    Serial.write(c);
```

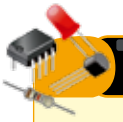
A conclusione dell'articolo, presentiamo un esempio completo che mostra una delle caratteristiche più interessanti di **Fishino**, ovvero la possibiltà di creare una propria infrastruttura di rete senza bisogno di un router esterno, alla quale connettersi in mobilità, per esempio con un cellulare. Un'applicazione simile potrebbe essere usata, ad esempio, per monitorare alcuni sensori all'aperto

Fig. 1

Fig. 2

```
                    // se si è arrivati a fine linea (carattere 'newline' ricevuto
                    // e la linea è vuota, la richiesa http è terminata
                    // quindi è possibile inviera una risposta
                    if (c == '\n' && currentLineIsBlank)
                    {
                            // invia uno header standard http
                            client.println("HTTP/1.1 200 OK");
                            client.println("Content-Type: text/html");
                            client.println("Connection: close");  // la connessione verrà chiusa automaticamente una volta inviata la risposta
                            client.println("Refresh: 5");  // aggiorna la pagina automaticamente ogni 5 secondi
                            client.println();
                            client.println("<!DOCTYPE HTML>");
                            client.println("<html>");

                            // invia il valore di tutti i pins analogici
                            for (int analogChannel = 0; analogChannel < 6; analogChannel++)
                            {
                                    int sensorReading = analogRead(analogChannel);
                                    client.print("analog input ");
                                    client.print(analogChannel);
                                    client.print(" is ");
                                    client.print(sensorReading);
                                    client.println("<br />");
                            }
                            client.println("</html>");
                            break;
                    }
                    if (c == '\n')
                    {
                            // inizio di una nuova linea
                            currentLineIsBlank = true;
                    }
                    else if (c != '\r')
                    {
                            // sono stati ricevuti dei caratteri nella linea corrente
                            currentLineIsBlank = false;
                    }
            }
    }
    // lascia tempo al browser per ricevere i dati
    delay(1);

    // chiudi la connessione
    client.stop();
    Serial.println("client disonnected");
  }
}
```

tramite un cellulare da una certa distanza, realizzando così dispositivi completamente portatili.
Un'altra interessante applicazione potrebbe essere un comando remoto via WiFi sempre tramite browser web sul cellulare.
L'esempio, nel **Listato 11** crea una rete WiFi "volante", con nome (SSID) '**FISHINO**', senza password (rete open) ed avvia un piccolo server che su richiesta fornisce una lettura dei sei ingressi analogici di Fishino. Una volta lanciato lo sketch, occorre selezionare la rete wireless **FISHINO** tra le reti disponibili (**Fig. 1**) e aprendo l'indirizzo **192.168.100.1** sul browser si ottiene il risultato visualizzato in **Fig. 2**.
Gli esempi qui riportati sono comunque contenuti, insieme ad altri, nella libreria **Fishino**.
Un'ultima nota sugli I/O occupati dalle estensioni, e che non vanno utilizzati come I/O quando sono attivi i componenti aggiuntivi. Il modulo **WiFi** utilizza i seguenti pins: **7, 10, 11, 12 e 13**. È disattivabile completamente con un ponticello tra il pin **CH_PH** del connettore **ESP** e la massa. La scheda **microSD** utilizza i seguenti pins: **4, 11, 12 e 13** ed impone che il pin **7** sia impostato ad output digitale. Per liberare i ports usati basta non inserire alcuna scheda nel connettore. Il modulo **RTC** comunica via i2c utilizzando i pins **SCL** e **SDA**, abbinati nell'UNO ai ports analogici **A4** ed **A5**.
Continueremo in un prossimo articolo con la descrizione della libreria **FishinoWebServer** che permette la realizzazione di un piccolo ma completo server web, che è la base dell'esempio di Home Automation mostrato in breve nel numero scorso.



per il MATERIALE

La board Fishino (cod. FISHINOUNO) viene fornita montata e collaudata. Può essere acquistata presso Futura Elettronica al prezzo di Euro 36,00. Il prezzo si intende 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