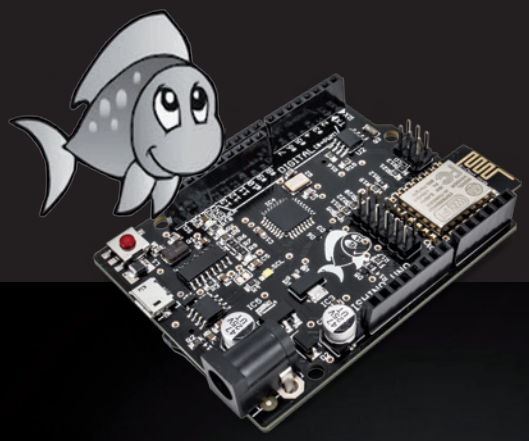


Realizziamo un'applicazione, eseguibile sia su PC che su un terminale Android, in grado di controllare tramite WiFi moduli a LED multicolore connessi a una scheda Fishino. Lo faremo utilizzando le Qt, che da questa prima puntata vi faremo conoscere.

1

di MASSIMO DEL FEDELE

Le librerie Qt consistono in una serie completa di strumenti software per lo sviluppo di applicazioni multipiattaforma in C++ e sono state sviluppate inizialmente dalla Trolltech, una software house finlandese acquisita successivamente dalla Nokia ed ora nota come Qt-Company. Dopo varie vicissitudini sulle quali sorvoliamo, le librerie sono ora disponibili all'utente con due tipi di licenza: GPL per software open source, gratuita, ed a pagamento per sviluppo di applicazioni closed source.
Grazie alle Qt sono state sviluppate innumerevoli applicazioni open source, tra cui per esempio il desktop KDE per i vari sistemi operativi Linux, che hanno portato alla disponibilità di un'enorme quantità di codice di ottima qualità.
Proprio perché il codice disponibile basato sulle Qt è a dir poco infinito, in questo corso ci limiteremo alla loro installazione e ad un esempio pratico di utilizzo per sviluppare un'applicazione, eseguibile sia su PC desktop che su un terminale Android, in grado di controllare tramite WiFi uno o più LED multicolore (RGBW, ossia rosso, verde, blu e bianco) connessi alle schede Fishino che vi abbiamo presentato nei precedenti fascicoli di Elettronica In.
Come accennato sopra, per lo sviluppo di codice open source e/o ad uso personale le Qt sono scaricabili gratuitamente ed utilizzabili senza alcun limite; purtroppo va detto che i costi delle licenze per lo sviluppo di software proprietario non sono proprio abbordabili per uno sviluppatore singolo...
Le Qt sono composte da una completa serie di librerie software, un sistema di sviluppo (IDE), chiamato Qt-Creator, contenente al suo interno

SDK Tools Only

If you prefer to use a different IDE or run the tools from the command line or with build scripts, you can instead download the stand-alone Android SDK Tools. These packages provide the basic SDK tools for app development, without an IDE. Also see the SDK tools release notes.

Platform	Package	Size	SHA-1 Checksum
Windows	installer_r24.4.1-windows.exe (Recommended)	151659917 bytes	f9b59d72413649d31e633207e31f456443e7ea0b
	android-sdk_r24.4.1-windows.zip	199701062 bytes	66b6a6433053c152b22bf8cab19c0f3fef4eba49
Mac OS X	android-sdk_r24.4.1-macosx.zip	102781947 bytes	85a9cccb0b1f9e6f1f616335c5f07107553840cd
Linux	android-sdk_r24.4.1-linux.tgz	326412652 bytes	725bb360f0f7d04eaccff5a2d57abdd49061326d

Fig. 1 – Ricerca dell'SDK per QT.

un sistema per la gestione dei progetti (un po' come l'IDE di Arduino), un editor grafico per le schermate delle applicazioni, un sistema per passare da una piattaforma all'altra (per esempio, da desktop ad Android) e molto altro. Le librerie disponibili coprono praticamente qualsiasi settore dell'informatica, ed in rete si trovano componenti aggiuntivi installabili nell'IDE nel caso se ne presenti la necessità. È anche possibile estendere il numero di widget (controlli) disponibili tramite plugin (estensioni), cosa che tratteremo in un futuro articolo, essendo l'installazione dei medesimi non proprio semplicissima.

Fishino & Colibrì
Controllo di LED RGBW tramite dispositivi mobili

Con questo progetto mostriamo l'utilizzo di vari tool di sviluppo applicati alla nostra scheda Fishino la quale, abbinata ad uno o più moduli Colibrì (si tratta di driver per LED RGBW che abbiamo descritto nel fascicolo n° 202) consente il controllo di luci o qualsiasi altro apparecchio tramite un cellulare Android oppure un'applicazione su Personal Computer connesso in rete.
Cogliamo l'occasione per sfruttare il protocollo di comunicazione UDP, recentemente introdotto nel firmware e nelle librerie di Fishino, che presenta notevoli vantaggi rispetto al protocollo TCP utilizzato abitualmente, in particolar modo se si opera in una rete locale. L'applicazione consiste in 2 moduli, uno che gira su Desktop/Android, sviluppato tramite le ottime librerie Qt, che andremo a descrivere abbastanza in dettaglio, e l'altro consistente in un semplicissimo sketch che gira sul nostro Fishino. L'App è in grado di rilevare autonomamente tutti i Fishini connessi in rete ed elencarne sul cellulare le luci connesse, permettendo la selezione del dispositivo su cui

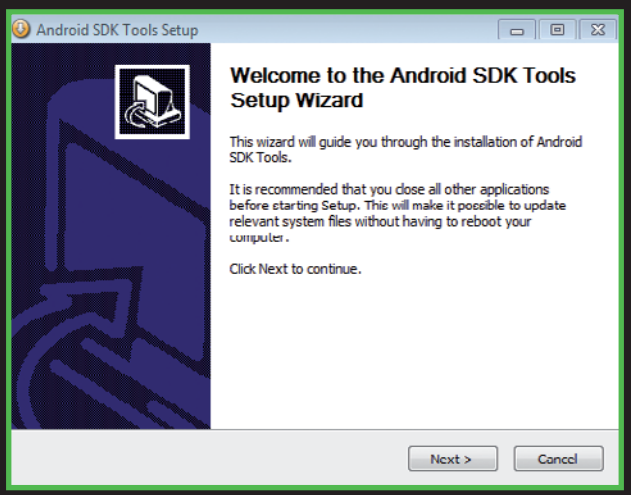


Fig. 2 – Avvio dell'installazione dell'SDK.

operare e, relativamente a questo, il cambio di luminosità dei 4 canali tramite 4 sliders.
L'installazione del sistema di sviluppo Qt + Android risulta abbastanza laboriosa ma, se ci seguirete fino in fondo, vi accorgerete che una volta superato lo scoglio iniziale lo strumento è efficacissimo per creare applicazioni facilmente trasportabili tra i vari dispositivi al prezzo di un semplice clic.
Potrete quindi sviluppare la vostra App sul desktop, collaudarla, eseguirne il debug e una volta pronta, caricarla sul cellulare e/o un emulatore senza dover cambiare una sola riga di programma.

Installazione del sistema di sviluppo per Android

Le Qt si appoggiano, per quanto riguarda lo sviluppo Android, sull' SDK di Google, reperibile sul web all'indirizzo *https://developer.android.com/sdk/index.html#Other*.
Poichè utilizzeremo le Qt per sviluppare le nostre applicazioni, non ci interessa il pacchetto Android Studio ma soltanto l' SDK; selezioneremo quindi la voce SDK Tools Only e, di questa, la versione raccomandata (recommended) che dipende dal sistema operativo in uso (**Fig. 1**).
Selezioniamo quindi la prima voce, quella con la scritta (*Recommended*). Apparirà l'usuale schermata con le condizioni del servizio, da accettare, e quindi un pulsante di download che avvierà lo scaricamento dell'installatore.
Salviamo anche qui l'installer sul desktop ed eseguiamolo (**Fig. 2**).

L'installer rileverà la versione attuale di Java installata nel computer e se precedente a quella richiesta, fornirà l'avviso visibile nella finestra di dialogo in **Fig. 3**. Vi verrà ora chiesta la tipologia di installazione: per evitare di dover eseguire in futuro l'applicazione di configurazione come amministratore, vi suggeriamo di selezionare la voce **Install just for me**; in questo modo l'**SDK** sarà utilizzabile solo dall'utente attuale (**Fig 4**). Verrà quindi richiesto il percorso di installazione; potete lasciare tranquillamente quello proposto, comunque prendete nota del percorso di installazione perché vi servirà in seguito.
Successivamente verrà chiesta la posizione nel menu delle applicazioni; anche qui si può lasciare la scelta suggerita. Facendo clic su **Install** verrà quindi avviata l'installazione vera e propria, che potrà durare pochi minuti.
Una volta completata, fate clic su **Next** e successivamente su **Finish**, lasciando selezionata la casella **Start SDK Manager** in modo da poter configurare l' **SDK** al passo successivo; verrà quindi avviato il gestore dell' **SDK** (**Fig. 5**). Suggeriamo di lasciare tutto come preselezionato, ovvero di accettare l'installazione tipica; sarebbe possibile eliminare alcuni package non utilizzati nelle nostre esercitazioni, ma per semplicità eviteremo di avvalerci di questa opzione.
Ora fate clic sul pulsante **Install 19 packages** (il numero può variare in base ai pacchetti selezionati) e apparirà l'usuale schermata

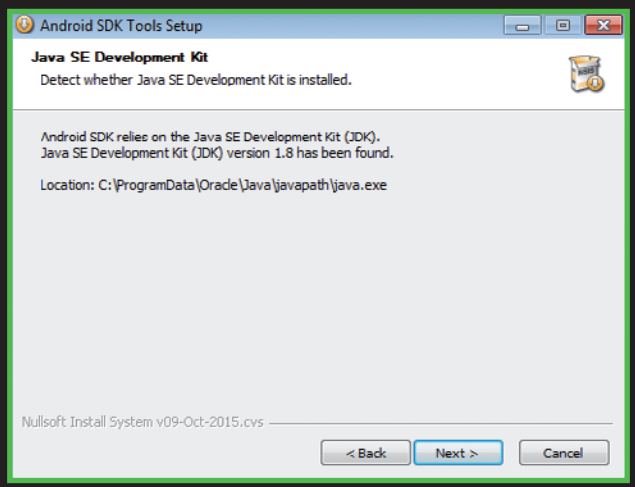


Fig. 3 – Verifica della versione di Java presente nel computer.

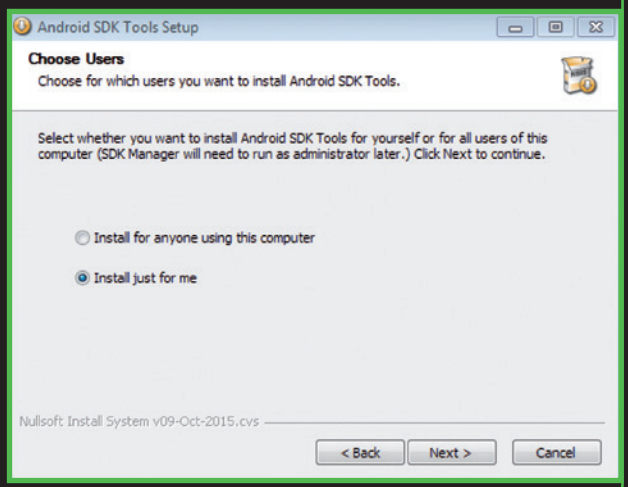


Fig. 4 – Scelta dell'utente che utilizzerà il software.

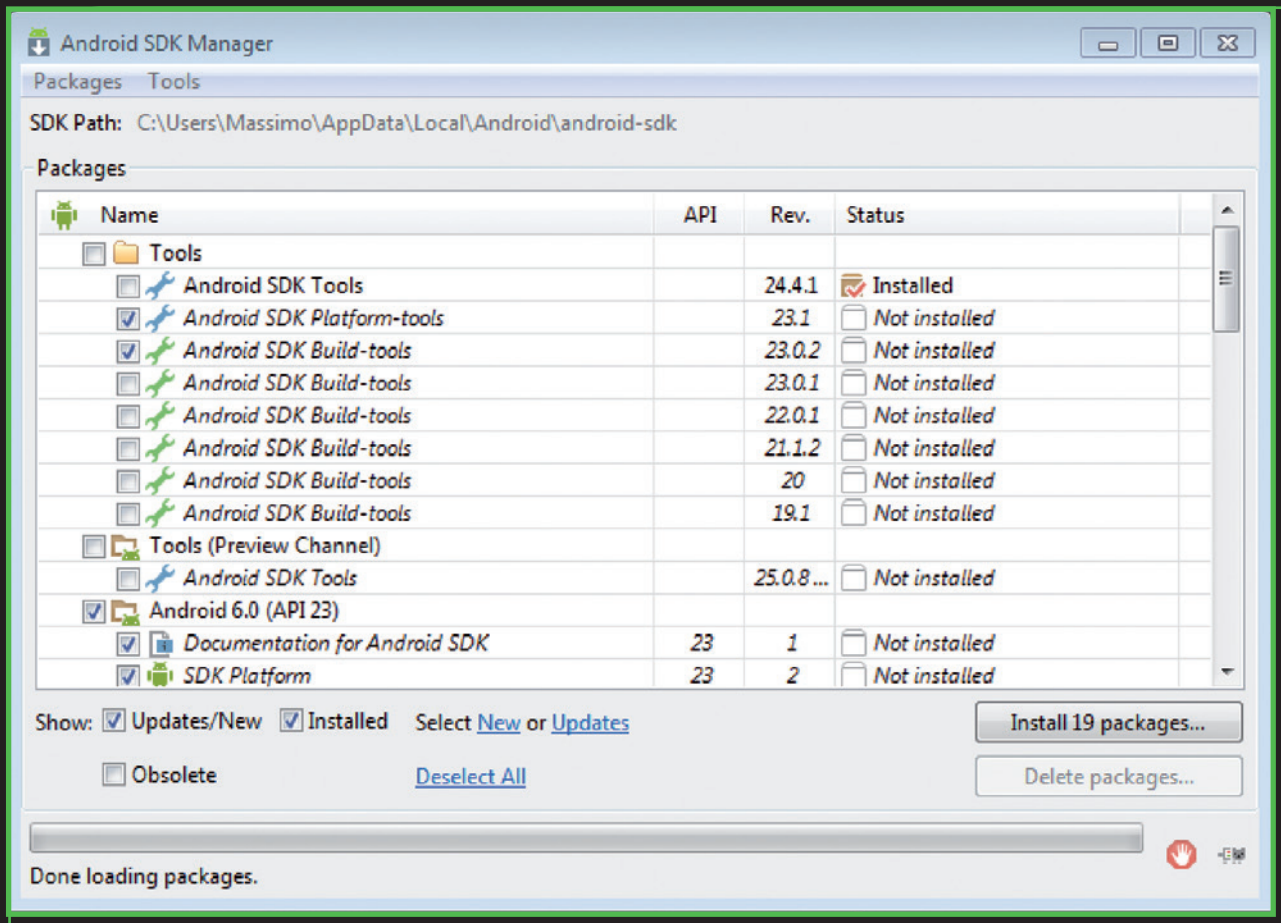


Fig. 5 – Installazione in corso da SDK Manager.

con le licenze, che dovrete accettare facendo successivamente clic sul pulsante **Install**. Verrà quindi avviato lo scaricamento dei pacchetti software, che potrà durare parecchio tempo.

Installazione dell'NDK

Diversamente da Processing, per esempio, le Qt non utilizzano Java per le applicazioni ma il C++, similmente ad Arduino. Poichè il linguaggio standard del toolkit di Android è il Java, occorre installare un toolkit di sviluppo aggiuntivo (NDK) che permette la programmazione in C++. Come "effetto collaterale" otterremo delle applicazioni compilate direttamente in linguaggio macchina nativo sul nostro Android, quindi leggermente più veloci rispetto alle applicazioni Java: non male, come effetto...
L' NDK (Native Development Toolkit) è scaricabile alla pagina web *http://developer.android.com/ndk/downloads/index.html*. Sono presenti diverse versioni: scegliete quella che corrisponde al vostro sistema operativo (Windows, in questo tutorial, 32 o 64 bit a seconda della vostra versione). A differenza dell' SDK, il file non contiene un installer ma un archivio auto-scompattante; occorre quindi scaricarlo e salvarlo "vicino" al percorso di installazione dell' SDK (di cui avete preso nota precedentemente). Aprite quindi la cartella e fate doppio clic sul file appena scaricato: si aprirà un terminale provvisorio dove si vedrà scorrere una sterminata lista di file mentre vengono estratti, in una cartella del nome simile a questo (il nome esatto dipende dalla versione dell'NDK): android-ndk-r10e.
Per comodità (vi tornerà utile in un secondo momento) rinominate la cartella togliendo il numero di versione, quindi come **android-ndk**. Il percorso completo dell' NDK sarà quindi del tipo *C:\Users\nomeutente\AppData\Local\Android\android-ndk*, dove "nomeutente" è il nome dell'utente sotto cui avete installato il software.
Prendete nota anche di questo percorso, perché

vi servirà per la fase finale di configurazione.

Installazione di ANT

Ant è un tool di compilazione simile al Make di Linux, ma scritto in Java. È necessario per la generazione degli eseguibili per Android, quindi va installato. Allo scopo basta scaricare il file in formato zip (il primo della lista) reperibile su *http://ant.apache.org/bindownload.cgi* e decomprimerlo in una cartella a scelta; per comodità vi consigliamo anche qui la cartella dove avete messo l' SDK ed il NDK di Android. Anche in questo caso prendete nota del percorso, perché vi servirà in seguito.

Installiamo QT

Finalmente potete installare QT: aprite la pagina *www.qt.io*, quindi fate clic sul pulsante "Get Started" al centro dello schermo; apparirà una schermata (**Fig. 6**) in cui viene chiesta la tipologia di installazione, ossia commerciale (*Commercial Deployment*), sviluppo interno/uso personale/studenti (*In-house deployment, private use or student use*) oppure open source sotto licenze LGPL o GPL (*Open Source under a LGPL or GPL license*). Nel nostro caso, volendo sviluppare programmi open source, sceglieremo la terza opzione. Le QT sono infatti gratuite per sviluppare programmi open source. Apparirà quindi una finestra di dialogo in cui viene chiesto se siamo certi di voler sviluppare applicazioni open: confermiamo con l'opzione *Yes*, allorché apparirà quindi una terza schermata in cui vi si chiederà se sarete in grado di rispettare gli obblighi relativi alle licenze LGPL o GPL, che sono piuttosto restrittivi, imponendo la pubblicità di tutto il sorgente delle applicazioni che svilupperete con le QT. Anche qui, scegliete *Yes*. Finalmente apparirà il pulsante di *Download*, insieme ad altre opzioni che però non ci interessano, in quanto utilizzeremo l'installatore on-line, che scaricherà ed installerà tutti gli elementi necessari (**Fig. 7**). Facendo clic sul pulsante **Download Now** verrà avviato il download dell'installer in questione, che dovrete salvare da qualche parte, ad esempio sul desktop (**Fig. 8**).
Salvate il file ed eseguitelo tramite il consueto doppio clic sull'icona corrispondente, allorché vi verrà richiesta la solita conferma da parte di

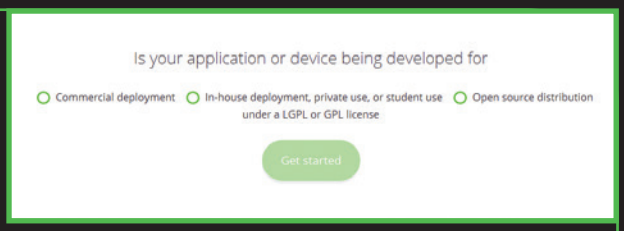


Fig. 6 – Scelta della licenza di QT

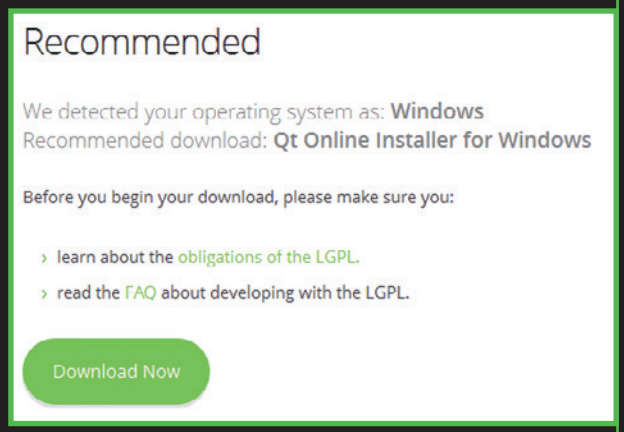


Fig. 7 – Download dell'installer di QT.

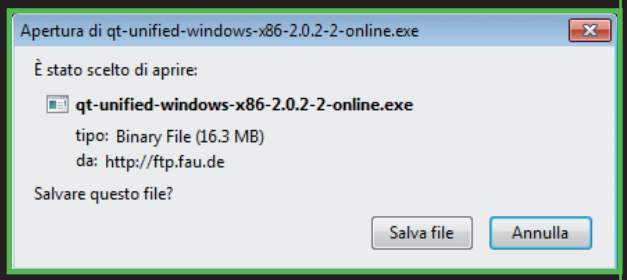


Fig. 8 – Salvataggio dell'installer nel computer.

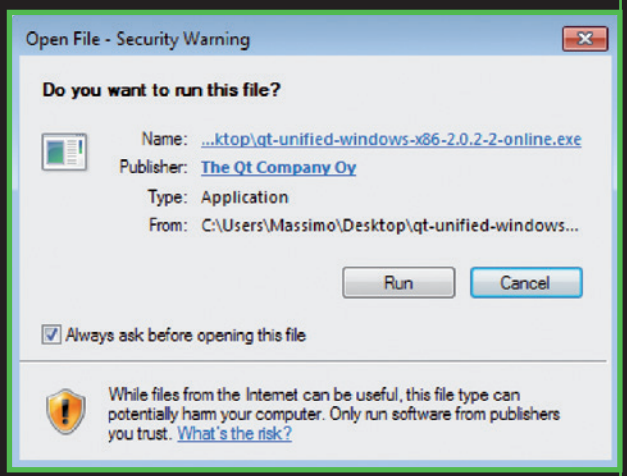


Fig. 9 – Finestra di dialogo per l'avvio dell'esecuzione dell'installer di QT.

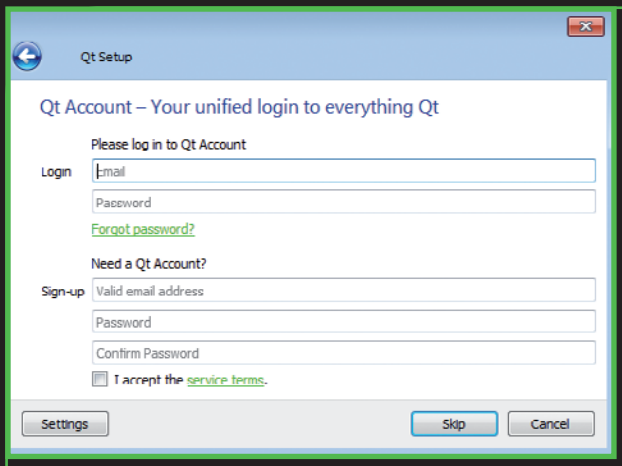


Fig. 10 – Finestra per login o registrazione del vostro account.

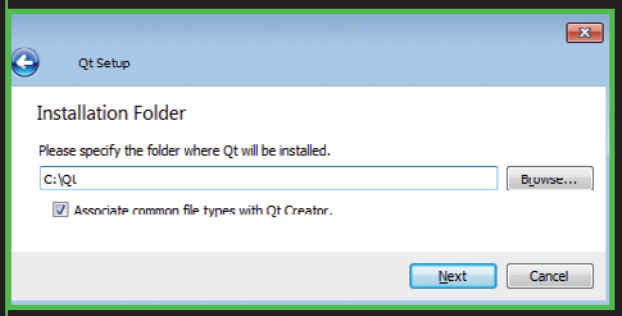


Fig. 11 – Scelta della directory di installazione di QT.

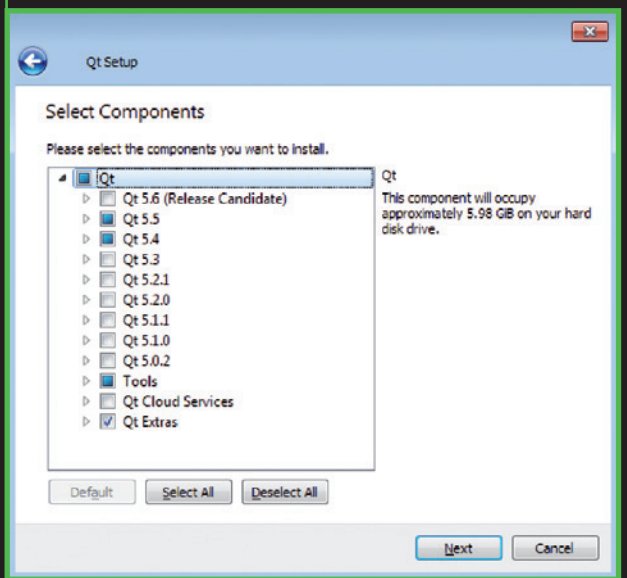


Fig. 12 – Scelta dei componenti da installare.

Windows (**Fig. 9**). Fate clic su **Run** e vi apparirà la schermata di benvenuto dell'installazione, in cui si comunica che è necessaria una registrazione al sito, che potrà essere anche fatta al volo nel passo successivo. In questa fate clic sul pulsante **Next** e vi apparirà la richiesta di login, da usare se siete già registrati al sito, oppure i dati per la registrazione (**Fig. 10**). Sceglieremo qui la seconda opzione, inserendo un'indirizzo e-mail e una password a scelta, senza dimenticarci di accettare i le condizioni tramite l'apposita casella.

Il passaggio di registrazione si potrebbe saltare, ma spesso nell'utilizzo viene comodo avere accesso al sito per aggiornamenti, help, eccetera, quindi conviene farlo. Facendo clic su **Next** (che apparirà al posto dello **Skip** una volta compilato il form) si passerà alla schermata successiva e contestualmente vi arriverà una e-mail per la conferma della registrazione. Qui, facendo clic su **Next** inizierà (finalmente!) la parte finale dell'installazione e vi verrà chiesto dove volete installare le **QT**; suggeriamo di mantenere la scelta proposta, come mostrato nella **Fig. 11**.

Nella finestra di dialogo, facendo clic sul pulsante **Next** apparirà l'importante schermata di selezione dei componenti da installare (**Fig. 12**); qui suggeriamo, per non appesantire troppo l'installazione, di deselezionare la versione 5.4, cliccando due volte consecutive sul quadratino corrispondente (la prima volta diventa un simbolo di spunta, selezionando tutti i sottocomponenti, mentre al secondo clic li deseleziona tutti).

È inoltre indispensabile aprire la sezione 5.5 (cliccando sul triangolino a sinistra della voce) e selezionare i componenti per Android, oltre a quelli desktop, visto che vorremo sviluppare applicazioni anche per i cellulari; è sufficiente per la stragrande maggioranza dei casi selezionare la versione Android per Armv7 (**Fig. 13**).

Senza quest'ultima opzione non sarete in grado di sviluppare applicazioni Android, a meno di non aprire l'utilità di gestione delle QT successivamente, quindi suggeriamo caldamente di attivarla subito. La versione armv7 è quella usata nella gran parte dei casi perché è su tale architettura che si basa la stragrande maggioranza dei dispositivi mobile funzionanti con Android.

Lasciate pure le altre opzioni come sono state preimpostate e fate nuovamente clic su Next;

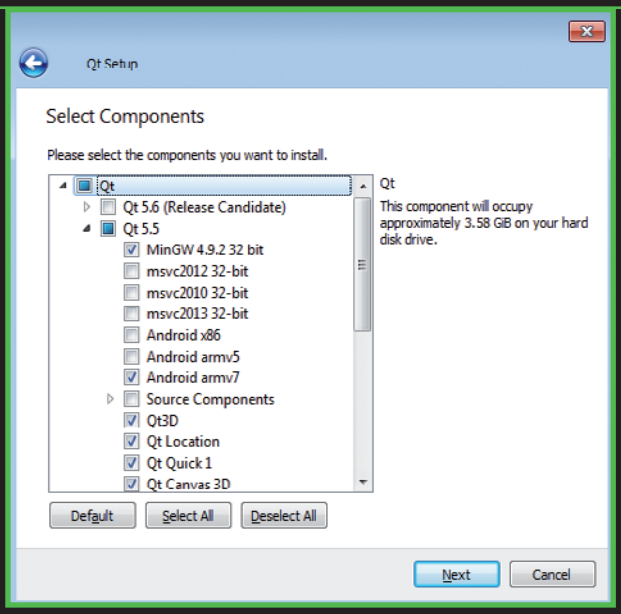


Fig. 13 – Selezione della versione Android per Armv7.

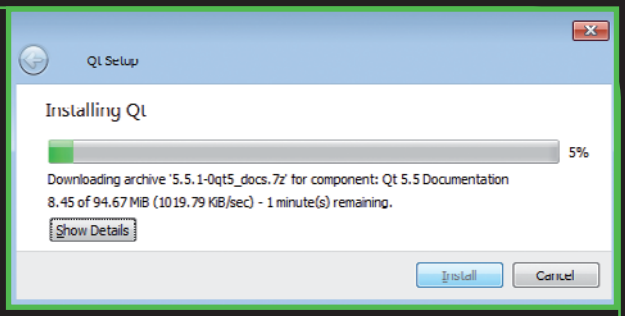


Fig. 14 – Installazione di Qt in corso.

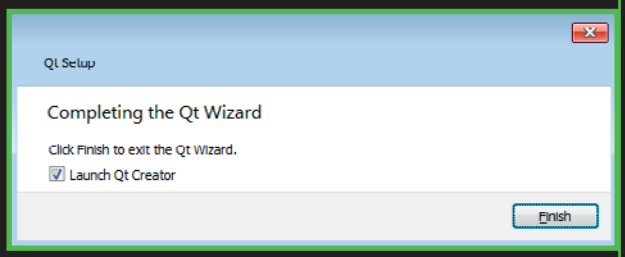


Fig. 15 – Conclusione delll'installazione di Qt.

apparirà un'ulteriore richiesta di accettazione delle condizioni, che qui omettiamo per brevità. È sufficiente selezionare la casella di accettazione e fare nuovamente clic su **Next**.
Vi verrà ora chiesto il nome del menu in cui si troveranno le applicazioni Qt; anche qui ometteremo la schermata, è sufficiente premere **Next** accettando il valore proposto (Qt).
Finalmente, nell'ultima schermata apparirà la richiesta di conferma per l'avvio dell'installazione.
È sufficiente fare clic su **Install** per proseguire.
Ora inizierà lo scaricamento dei pacchetti selezionati, che potrà richiedere anche parecchio tempo a seconda delle prestazioni del vostro computer e della velocità della rete (**Fig. 14**). Ad installazione completata apparirà la finestra di dialogo conclusiva nella quale, lasciando selezionata la casella **Launch Qt Creator**, verrà eseguita l'applicazione principale di **Qt** nella quale potremo iniziare a scrivere i nostri programmi (**Fig. 15**).
In questa finestra, facendo clic su **Finish** verrà quindi lanciato **Qt Creator**, la cui finestra di lavoro è illustrata nella **Fig. 16**.
L'applicazione **Qt Creator** dispone di un tutorial, che per brevità non descriveremo in questa sede; potete provarlo da voi facendo clic sul pulsante **Get Started Now**.
Noi ci limiteremo, nelle prossime puntate di questo corso, a creare il nostro primo progetto d'esempio, chiamato **Colibrì**.

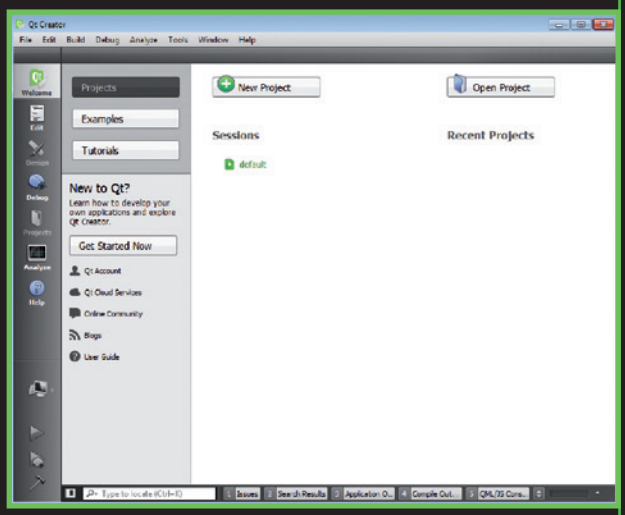


Fig. 16 – Finestra di avvio di Qt Creator.

Impostazioni di Qt Creator

L'ultima fase dell'installazione consiste nel fornire a Qt Creator i percorsi dei due toolkit per Android precedentemente installati, del toolkit di sviluppo Java (JDK) e di Ant.
Aprite quindi il menu **Tools** di Qt Creator e in esso impartite il comando **Options**: vi apparirà la schermata illustrata nella **Fig. 17**, dove occorre selezionare -nella lista proposta a sinistra- la voce Android, in modo da visualizzare le relative impostazioni nella parte di destra.
Le prime tre righe in alto sono i percorsi che dovremo introdurre; il primo (JDK location)

dovremo ricercarlo tra le cartelle del PC, nel caso sia già installato; in caso contrario è sufficiente premere l'icona con la freccia in giù, a destra del corrispondente tasto Browse, per aprire Internet Explorer sulla pagina di scaricamento del JDK.
Per brevità tralasciamo le eventuali istruzioni di installazione, che sono comunque disponibili nel sito di riferimento; occorre ricordarsi solo di installare il pacchetto corretto (JDK) nella versione più recente e per il sistema operativo utilizzato (Windows a 32 o 64 bit nel nostro caso). Anche qui, si tratta di scaricare un installer ed eseguirlo. A fine installazione, nella nostra macchina il percorso è risultato *C:\Program Files\Java\jdk1.8.0_74*. Nella versione Italiana di Windows si troverà probabilmente in *C:\Programmi\Java\jdkxx.yy.zz_tt*, ove xx, yy, zz e tt sono il numero della versione di Java installato. Fate quindi clic sul pulsante **Browse** accanto alla prima casella e ricercare il percorso che, una volta dato l'**OK**, apparirà nella casella JDK location; in alternativa è possibile digitarlo per esteso nella medesima casella. Gli altri tre percorsi risultano più semplici, visto che ne abbiamo preso nota precedentemente; è sufficiente quindi scriverli nelle relative caselle. Ad impostazione completata, la schermata sarà simile a quella mostrata in **Fig. 18**.
Con questo passaggio abbiamo finalmente terminato la laboriosa installazione del sistema di sviluppo.
Tutti i progetti realizzati tramite **Qt Creator** si troveranno nella cartella **Documenti**; è comunque possibile modificarla nella sezione '**Build and Run**' del menu impostazioni di **Qt Creator**.
Per tenere tutto in ordine suggeriamo di creare una sottocartella chiamata **Qt** dentro ai **Documenti** e modificare le impostazioni in modo che i progetti vengano salvati dentro questa.
Abbiamo dunque concluso la parte iniziale di questo corso; vi diamo appuntamento alla prossima puntata, nella quale entreremo nel vivo dello sviluppo descrivendo la realizzazione della nostra prima applicazione d'esempio, per poi passare, nelle puntate successive, alla creazione dell'applicazione obiettivo del corso.

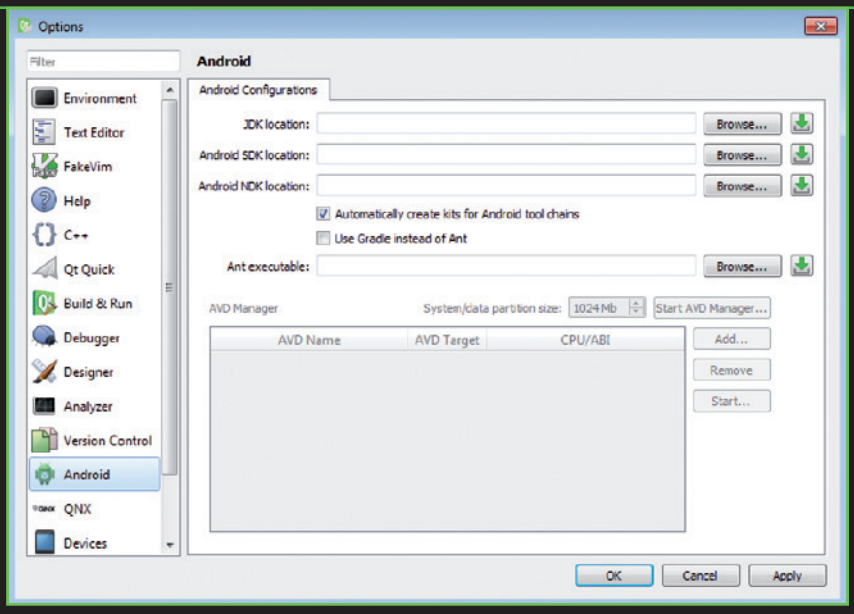


Fig. 17 – Configurazione Android.

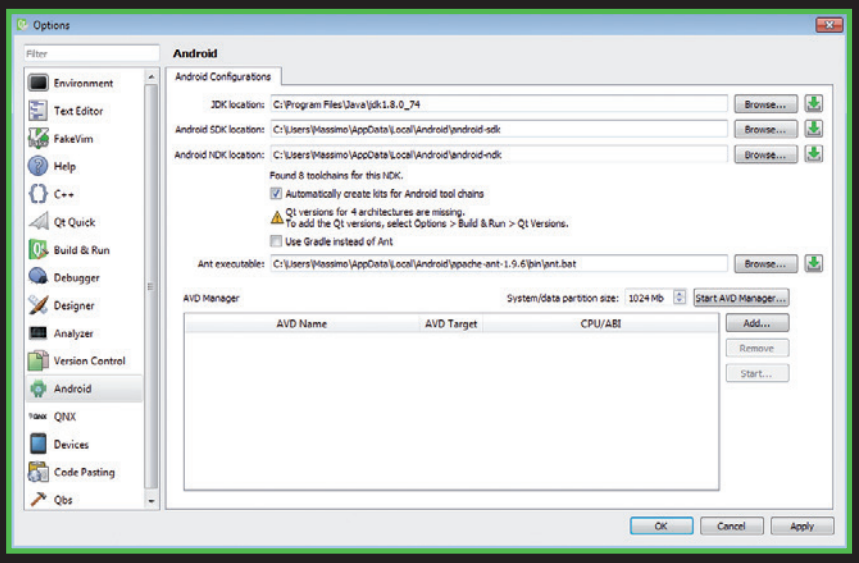
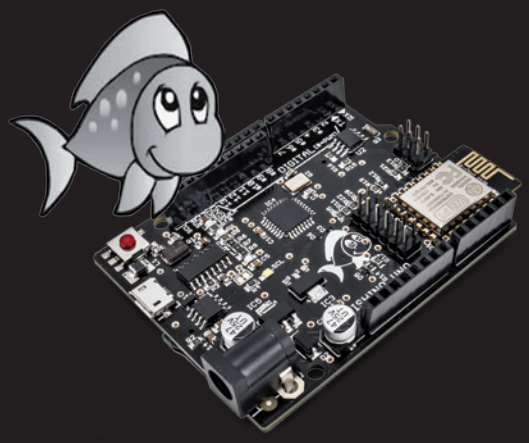


Fig. 18 – Impostazione Android in Qt completata.



Creiamo la struttura della nostra prima applicazione con Qt attraverso l'utilizzo dei tool scaricati. Seconda puntata.

Qt Code less. Create more. Deploy everywhere.

2

di MASSIMO DEL FEDELE

Abbiamo imparato, nella puntata precedente di questo corso, che cosa sono e a cosa servono le librerie Qt e con esse ci siamo proposti di realizzare un controllo da smartphone attraverso la connessione WiFi di un modulo driver per LED RGBW (a luce rossa, verde, blu e bianca) sfruttando la connettività wireless di una scheda Fishino. Per farlo abbiamo studiato i tool software di contorno allo sviluppo, tra cui l'SDK Android (indispensabile perché il nostro proposito è realizzare un'app per smartphone basati sul diffusissimo sistema operativo di Google) del toolkit di sviluppo aggiuntivo (NDK) e di ANT, che è un tool di compilazione simile al Make di Linux, ma scritto in Java.
Arrivati a questo punto, possiamo iniziare a creare l'applicazione: apriamo Qt Creator e nella finestra principale facciamo clic sul pulsante **New Project**, il che determinerà l'apertura della schermata visibile in **Fig . 1**.
Come si può vedere esistono varie tipologie di progetti; noi sceglieremo **Application** (Applicazione) sulla sinistra e **Qt Widgets Application** (applicazione grafica Qt) in centro; sulla destra vediamo una breve descrizione del tipo di applicazione e le piattaforme supportate (**Desktop Android**).
Qui si nota già un grosso vantaggio delle **Qt** : è possibile creare un'applicazione in modalità **Desktop**, collaudarla e debuggarla come tale, senza i tempi lunghi di caricamento nel device Android e/o nell'emulatore e, una volta funzionante, basta cambiare modalità per generare l'App definitiva.
Facciamo clic sul pulsante **Choose** e ci apparirà una schermata (**Fig. 2**) per definire

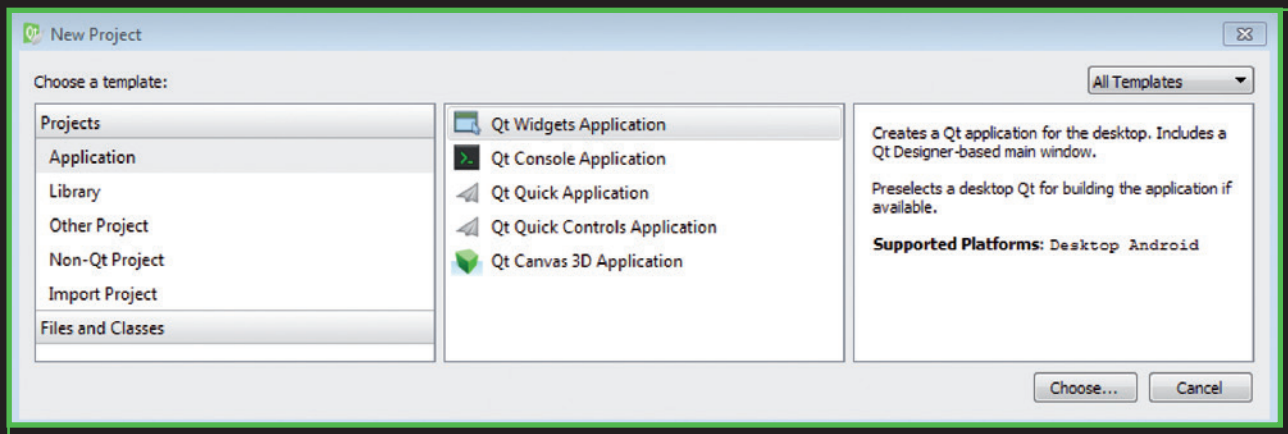


Fig. 1 – Finestra per la creazione di un nuovo progetto.

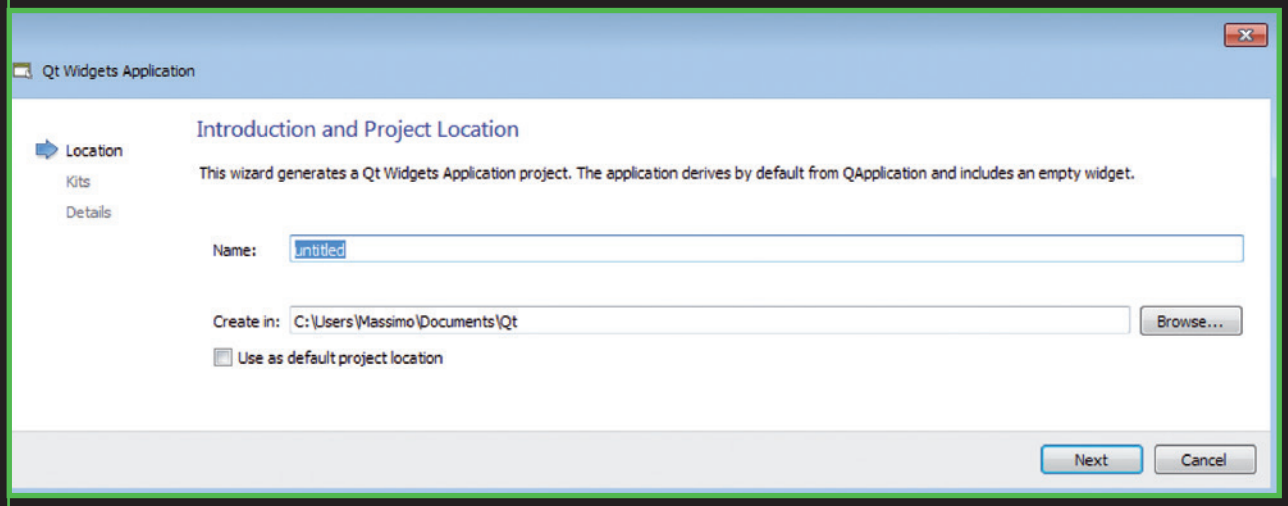


Fig. 2 – Definiamo nome e percorso in cui salvare l'appicazione.

il nome dell'applicazione. Scriviamo **Colibri** nell'apposita casella e accettiamo la destinazione predefinita (è comunque possibile specificarne una diversa) quindi facciamo clic su **Next**.
Adesso appare una nuova finestra di dialogo (**Kit Selection**) che permette di scegliere tra i kit installati, ovvero dei sistemi di sviluppo da utilizzare per la nostra applicazione (**Fig. 3**). In pratica, ogni progetto può utilizzare diversi kit che corrispondono alle diverse piattaforme supportate; nel nostro caso, avendo installato le piattaforme per applicazioni **Desktop** ed **Android** ci troveremo quelle voci. Selezioniamole entrambe, visto che vogliamo sviluppare e collaudare la nostra App sul computer e successivamente caricarla nel nostro terminale **Android**.
Scelto il kit facciamo clic su **Next** e ci apparirà la finestra di dialogo mostrata in **Fig. 4**, nella quale potremo scegliere il nome per le **classi C++** che **Qt Creator** genererà automaticamente per noi, e la classe su cui si baserà la nostra App. Cambiamo solo il nome della classe in **Colibri**, come si vede nella schermata in **Fig. 4**; il resto va lasciato inalterato. Il checkbox **Generate form** indicherà a **Qt Creator** di creare la finestra principale dell'applicazione. Facendo clic su **Next** apparirà una finestra di riepilogo che non dovete modificare; cliccando su **Finish** verrà quindi generato lo scheletro della nostra App (**Fig. 5**).
Sulla sinistra potete notare l'elenco dei file del progetto, divisi per categorie:

- Colibri.pro è il progetto; al momento non è necessario aprire il relativo file;
- Headers contiene tutti i files di include (.h);
- Sources contiene tutti i files sorgente (.cpp), dei quali al momento ci interessa colibri.cpp;
- Forms contiene le schermate della nostra

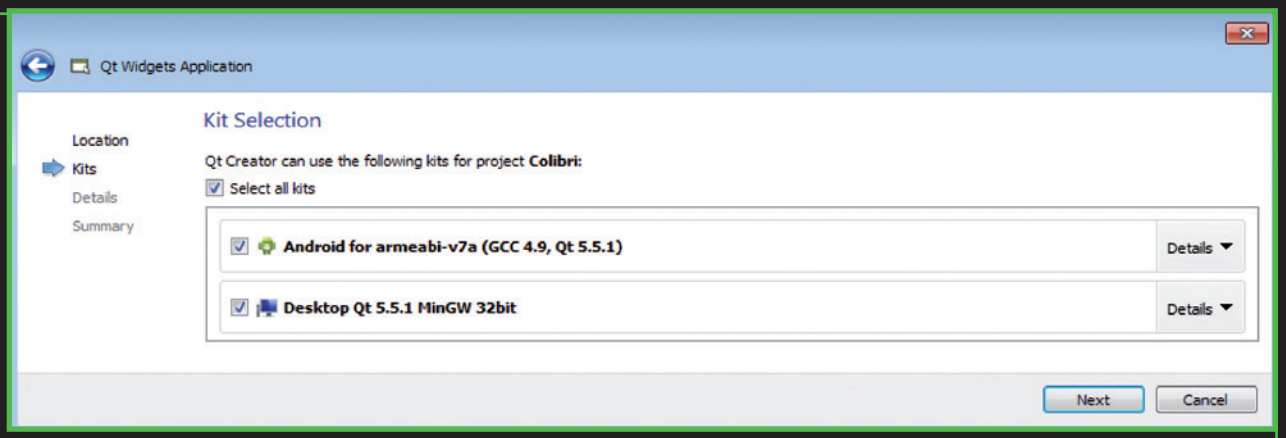


Fig. 3 – Selezione dei sistemi di sviluppo con cui creare l'applicazione.

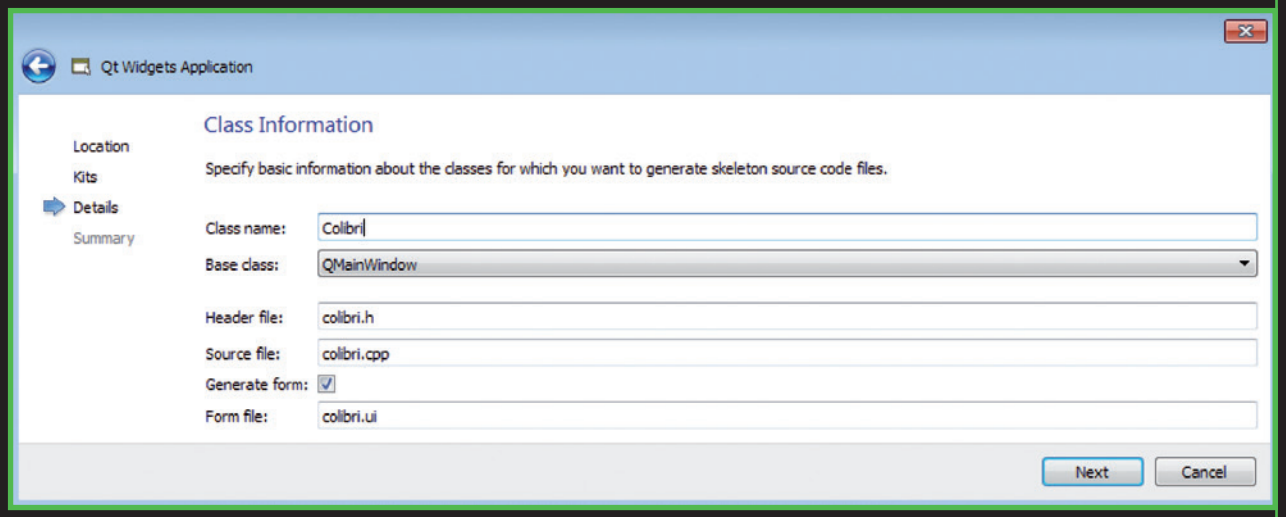


Fig. 4 – Assegnazione del nome alle classi.

applicazione; aprendolo e facendo doppio clic su colibri.ui, si apre il Form Editor, che permette di modificare graficamente l'aspetto della nostra app, inserendo i vari Widget (controlli) nella medesima (**Fig. 6**).

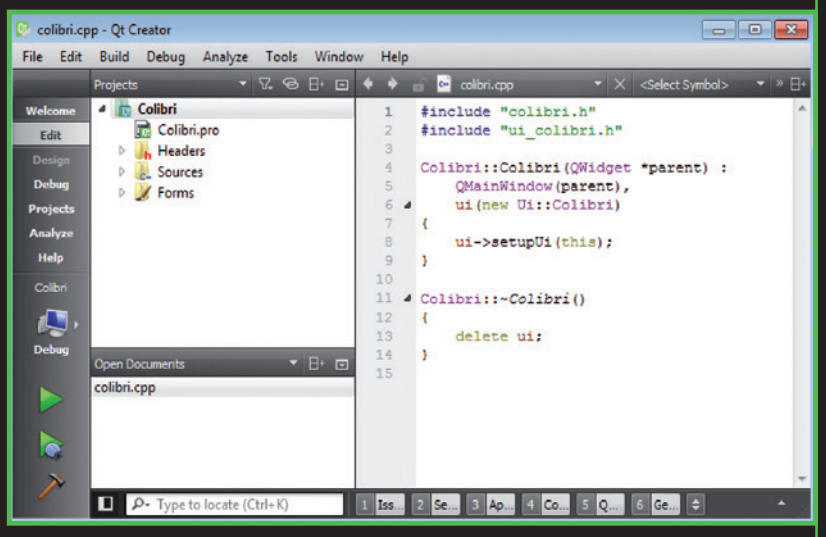


Fig. 5 - Struttura base dell'applicazione.

Sul lato sinistro ci sono i Widget (controlli) disponibili, al centro c'è la finestra dell'applicazione (sfondo grigio con i puntini della griglia), mentre a destra le proprietà dei Widget stessi. Siccome vogliamo creare un'app che si adatti allo schermo del nostro device, iniziamo ad inserire un elemento di layout (FormLayout) che è il più semplice da utilizzare; una volta che ci saremo impratichiti

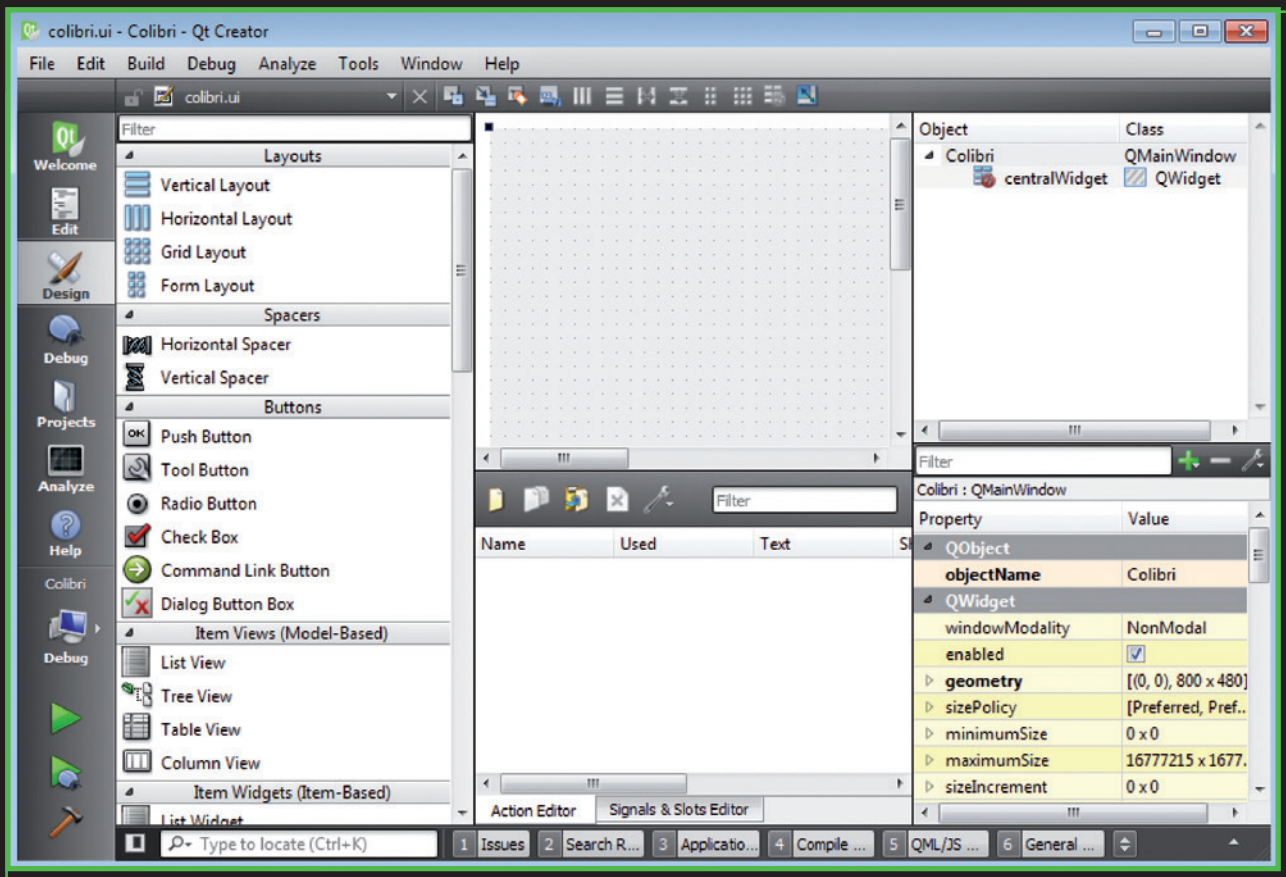


Fig. 6 - Finestra di dialogo Form Editor.

con il sistema potremo utilizzare altre tipologie che permettono posizionamenti più flessibili. Il Form Layout consente di creare un semplice dialogo costituito da varie righe, ognuna delle quali contiene un'etichetta (Label) ed un elemento grafico.
Trasciniamo quindi con il mouse il Vertical Layout dentro la nostra finestra dell'applicazione, ridimensioniamolo in modo da occupare quasi tutta la finestra e trasciniamo dentro di esso questi elementi in sequenza: Label - Horizontal Slider - Label - Horizontal Slider - Label - Horizontal Slider - Label - Horizontal Slider - Label - Combo Box, avendo l'accortezza di piazzarli all'interno del Form Layout come mostra la **Fig. 7**.
Fatto questo, è opportuno dare dei nomi significativi ai vari elementi ed inserire nei testi nelle **Label**; selezionandoli uno ad uno nel riquadro in basso a destra appariranno le proprietà degli oggetti che sono liberamente modificabili. Per le **Label**, la proprietà da cambiare è la **Text**, mentre per il nome dell'ogggetto si può lasciare quello predefinito.

Per i vari controlli (**Sliders e Combo Box**) occorre invece cambiare i nomi degli oggetti; noi scegliamo **redSlider**, **greenSlider**, **blueSlider**, **whiteSlider** e **deviceCombo** rispettivamente (**Fig. 8**). Adesso, facendo clic con il tasto destro del mouse sullo sfondo grigio del form bisogna selezionare **Lay out** e **Lay out in a form layout** in modo da adattare le dimensioni alla schermata.
Fatto questo, avete completato la costruzione dell'interfaccia grafica; occorre ora scrivere il codice per farla funzionare.

Meccanismo signal-slot

Le **Qt** utilizzano un interessante meccanismo, non standard nel **C++** (che richiede un preprocessing del codice, un po' come quanto succede nell'**IDE** di **Arduino**), cosa fatta automaticamente da **Qt Creator** tramite l'uso dell'applicazione **qmake**: il sistema **signal-slot**. In pratica, ogni **widget** (elemento grafico) è in grado di generare diversi tipi di eventi (**signals**) a seconda delle interazioni dell'utente; questi segnali possono essere **connessi** a pezzi

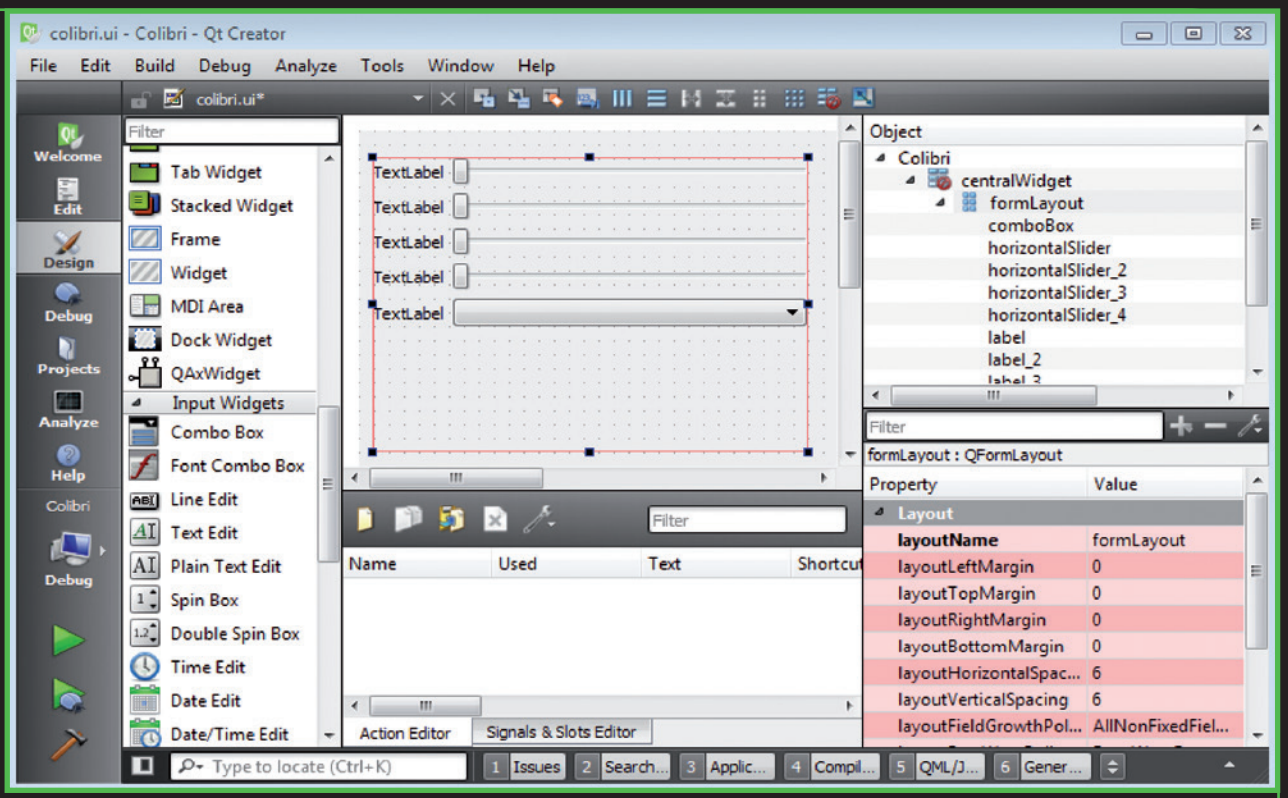


Fig. 7 - Elementi nel Form Layout.

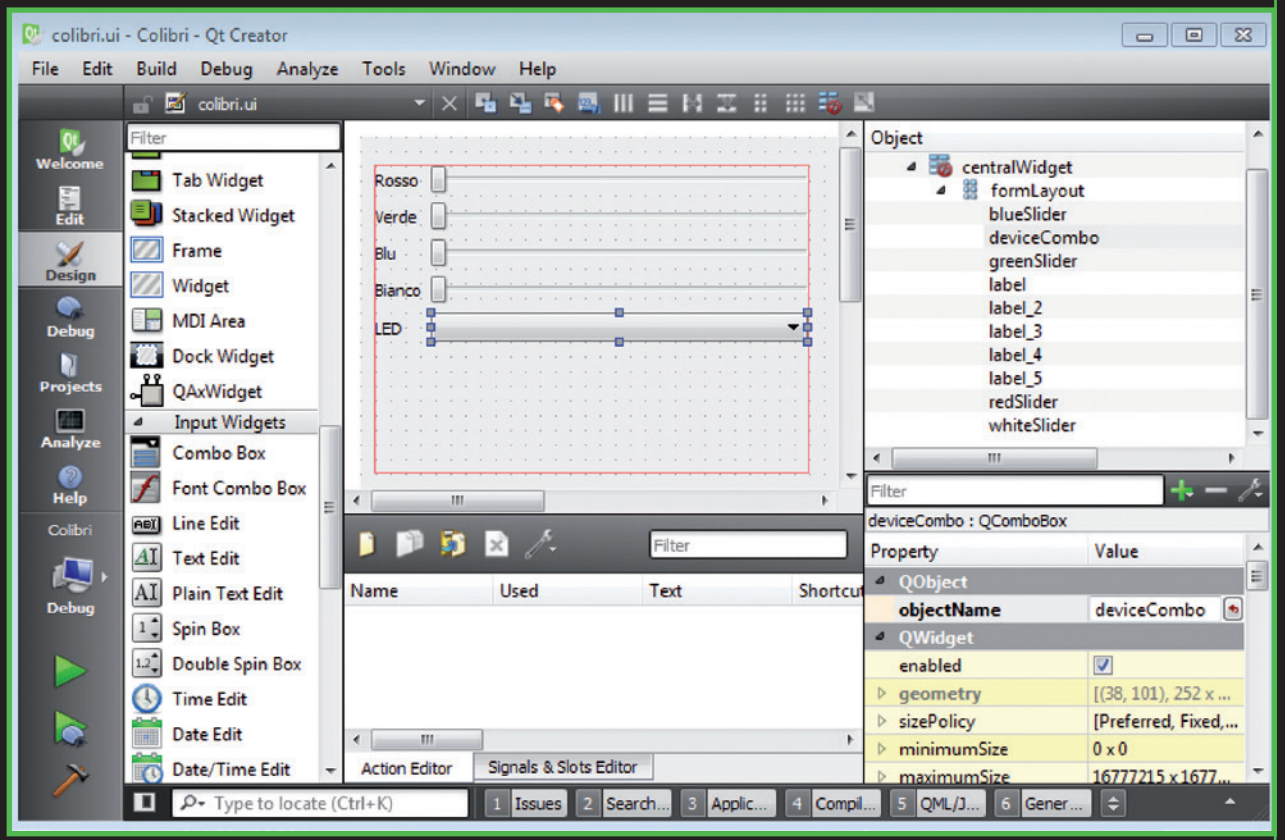


Fig. 8 – Nomi dei controlli dell'app.

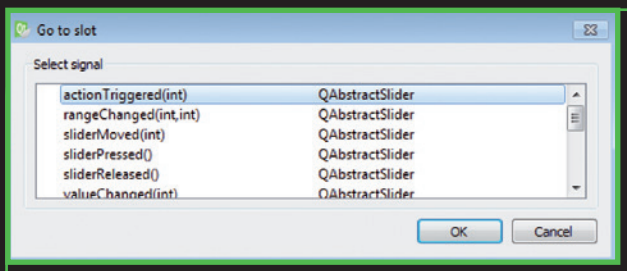


Fig. 9 - Finestra Go to slot.

```
15
16  void Colibri::on_redSlider_sliderMoved(int position)
17  {
18
19  }
20
```

Fig. 10 - Funzione di gestione del segnale.

```
10  class Colibri : public QMainWindow
11  {
12      Q_OBJECT
13
14  public:
15      explicit Colibri(QWidget *parent = 0);
16      ~Colibri();
17
18  private slots:
19      void on_redSlider_sliderMoved(int position);
20
21  private:
22      Ui::Colibri *ui;
23  };
```

Fig. 11 - Dichiarazione della funzione.

```
16  void Colibri::on_redSlider_sliderMoved(int position)
17  {
18
19  }
20
21  void Colibri::on_greenSlider_sliderMoved(int position)
22  {
23
24  }
25
26  void Colibri::on_blueSlider_sliderMoved(int position)
27  {
28
29  }
30
31  void Colibri::on_whiteSlider_sliderMoved(int position)
32  {
33
34  }
35
36  void Colibri::on_deviceCombo_currentIndexChanged(int index)
37  {
38
39  }
```

Fig. 12 - File Colibri.cpp dopo la creazione degli slot.

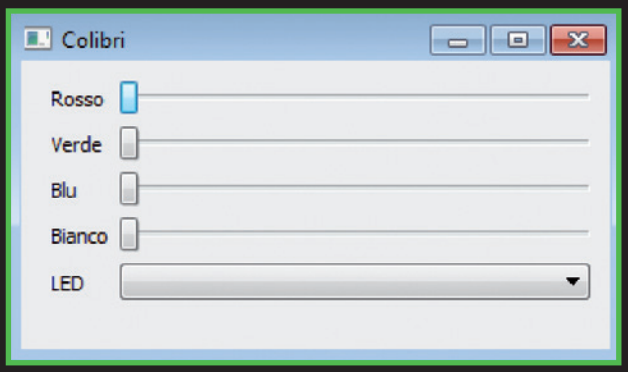
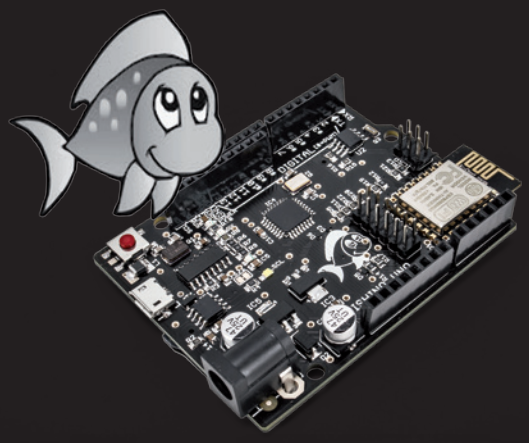


Fig. 13 - Gli slider dell'app ancora inattivi.

di codice C++, chiamati **slot**. È un po' come connettere un azione al codice che la deve gestire. Il meccanismo signal-slot è gestibile sia manualmente, creando gli slot nel codice e collegandoli con una funzione connect, sia tramite l'interfaccia grafica, il che è più semplice ma produce come effetto collaterale un codice più lungo, creando uno slot per ogni segnale connesso. Sceglieremo qui, per semplicità, questa strada.
Nel layout editor clicchiamo col tasto destro sul primo slider (il redSlider) e nel menu che si apre selezioniamo Go to slot; si aprirà la finestra di dialogo mostrata in **Fig. 9**.
Selezioniamo quindi la voce sliderMoved(int) e facciamo clic su OK: si aprirà automaticamente il file colibri.cpp con il cursore posizionato dentro alla funzione che gestirà il nostro segnale (**Fig. 10**). Aprendo il file Colibri.h si potrà notare che è stata correttamente aggiunta la dichiarazione della nostra funzione (**Fig. 11**). Notate che Qt Creator fornisce un metodo veloce per creare lo scheletro del codice.

Proseguiamo creando altre connessioni, come fatto per il primo segnale; dovremo gestire tutti i segnali di tipo sliderMoved() di tutti gli slider dell'interfaccia grafica e il segnale currentIndexChanged(int) per il combo box. Una volta completata la creazione degli slot, ci troveremo nel file Colibri.cpp la situazione mostrata in **Fig. 12**.
Abbiamo qui completato la parte "semiautomatica" della nostra applicazione; resta ora da scrivere il codice per implementare le funzioni. Volendo è già possibile testare il funzionamento dello scheletro dell'applicazione; allo scopo è sufficiente cliccare sull'icona in basso a sinistra con il triangolino verde e, dopo qualche istante, apparirà la finestra della nostra App. Ovviamente, non avendo ancora scritto nemmeno una riga di codice, lo spostamento degli slider non sortirà alcun effetto (**Fig. 13**). Quindi dobbiamo scrivere il codice che animerà l'app, ma di questo ci occuperemo nella terza ed ultima puntata di questo corso.



Scriviamo finalmente il firmware della nostra applicazione e lo sketch per Fishino. Ultima puntata.

Qt Code less. Create more. Deploy everywhere.

3

di MASSIMO DEL FEDELE

Abbiamo conosciuto le librerie Qt e i tool di sviluppo che ci servono per realizzare la nostra app e con essi abbiamo costruito quello che abbiamo definito lo "scheletro" della nostra applicazione per il controllo da smartphone Android (o da PC con installato Windows), tramite link WiFi e la scheda Fishino di un modulo RGBW Colibrì. Nella seconda puntata di questo tutorial abbiamo anche spiegato come creare l'interfaccia grafica, che abbiamo costruito insieme e che per il momento è inattiva e naturalmente potrà funzionare solo dopo che avremo scritto il codice per "animarla"; abbiamo anche introdotto il sistema **signal-slot**, dove ogni **widget** (elemento grafico) è in grado di generare diversi tipi di eventi (**signals**) a seconda delle interazioni dell'utente (tali segnali possono essere connessi a pezzi di codice C++, chiamati **slot**, come un'azione viene correlata al codice che la deve gestire).

In questa puntata conclusiva del nostro corso completiamo il nostro lavoro, mettendo mano al codice e scrivendo le parti mancanti nei file **Colibri.h** e **Colibri.cpp** accennati nella puntata precedente; ma prima di farlo aggiungiamo il modulo network al file del progetto, visto che avremo bisogno dei componenti per accedere ai socket UDP; il **Listato 1** mostra il contenuto del file **Colibri.pro** al quale abbiamo aggiunto la sola parola network dopo **QT +=**.

Guardate adesso il **Listato 2**, contenente il file **Colibri.h** al quale abbiamo aggiunto innanzitutto le costanti in **UDP_PACKET_CODES** contenenti i codici dei comandi da inviare alla scheda Fishino nei pacchetti **UDP**; notate che abbiamo previsto tre codici (escluso il **UDP_EMPTY** che è un comando nullo): il primo serve per impostare i valori di

Listato 1

```
#-------------------------------------------------
# Project created by QtCreator 2016-03-09T09:24:39
#-------------------------------------------------

QT       += network core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = Colibri
TEMPLATE = app


SOURCES += main.cpp\
        colibri.cpp

HEADERS  += colibri.h

FORMS    += colibri.ui

CONFIG += mobility
MOBILITY =
```

luminosità del LED RGBW, il secondo per leggerli ed il terzo è il comando di **discovery** che permette di localizzare tutte le board Fishino presenti in rete che eseguono lo sketch **Colibrì**. Successivamente abbiamo inserito il numero di porta UDP utilizzata per le comunicazioni (è 47777) ed alcune funzioni a basso livello impiegate per inviare e ricevere pacchetti UDP, oltre ad alcune ad alto livello per impostare o leggere i valori di luminosità e per rilevare, ancora una volta, le schede Fishino collegate in rete wireless.

Veniamo adesso al codice sorgente vero e proprio, che è quello visibile nel **Listato 3**: stiamo parlando del file **Colibri.cpp**.

Qui possiamo vedere l'implementazione delle varie parti dell'App; in particolare, notiamo che nelle funzioni degli **slot**, che in precedenza erano vuote, abbiamo inserito le chiamate a **setDeviceValues** per gli slider dell'interfaccia grafica e **getDeviceValues** per il **combo box**, in modo da poter modificare le luci scorrendo gli slider e leggere i valori impostati quando si seleziona una nuova luce tramite il combo box.

Le funzioni ad alto livello creano un pacchetto **UDP** costituito da una serie di byte, dei quali il primo è il **codice di comando**, il secondo è il **numero di device** (va definito perché ogni scheda Fishino connessa in rete è in grado di comandare più lampade, ovvero più controller Colibrì) mentre i rimanenti dipendono dal comando stesso e possono essere vuoti. Ad esempio, le linee di codice:

```
// build the needed query UDP packet
QByteArray packet;
packet.append(UDP_GETLIGHT).append(device);

// send the packet
sendUdpPacket(packet, addr);
```

creano un array di byte dinamico (**QByteArray**) e vi inseriscono (**append**) il comando per leggere i valori di luminosità (**UDP_GETLIGHT**) nel device (**device**) richiesto. Il pacchetto viene quindi spedito (**sendUdpPacket**) alla scheda Fishino caratterizzata dall'indirizzo IP richiesto (addr). Sempre degno di nota è il codice nel costruttore **Colibri::Colibri()** che inizializza un **socket Udp** e si mette in ascolto di pacchetti sulla porta specificata, realizzando quindi un semplice ma efficace **server UDP**:

```
// initialize UDP socket
udpSocket = new QUdpSocket(this);
udpSocket->bind(UDP_PORT, QUdpSocket::ShareAddress);
```

Ultima, ma non in ordine di importanza, è la funzione **discoverDevices()** che si occupa di rilevare tutte le schede **Fishino** connesse al sistema. Questa sfrutta una particolarità del protocollo **UDP**, vale a dire la possibilità di inviare un pacchetto di Broadcast, ovvero non diretto ad un particolare IP ma a tutti gli IP della rete locale.

La funzione in questione invia un pacchetto di discovery in broadcast e si mette in attesa delle risposte da parte delle schede Fishino, ciascuna delle quali deve comunicare, in breve tempo,

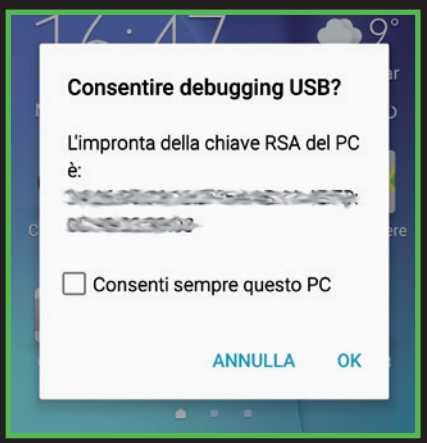


Fig. 1 Richiesta di autorizzazione.

la propria presenza nella rete ed il numero di lampade RGBW (ovvero di controller Colibrì) ad essa connesse.
Fatto questo, le informazioni ricevute vengono utilizzate per riempire il combo box, in modo da poter selezionare la lampada RGBW (il controller Colibrì) su cui intervenire con l'App.
Per ragioni di spazio non approfondiamo ulteriormente il codice: le Qt contengono un insieme estesissimo di funzionalità per descrivere le quali non basterebbero 10 numeri della rivista. Contengono comunque un'ottima documentazione nella quale si potranno cercare le classi utilizzate e tutti gli approfondimenti del caso.
Il programma è stato volutamente mantenuto semplice, evitando alcune funzionalità che sarebbero indispensabili in un'applicazione completa, quali l'interrogazione delle lampade RGBW entro un certo intervallo di tempo (ciò, allo scopo di controllare che i valori non siano stati cambiati tramite un'altra applicazione eseguita in parallelo), ed un sistema per modificare i nomi delle lampade che adesso appaiono nel formato IP:nn, ovvero vengono visualizzate tramite il numero di IP e di device all'interno della stessa scheda Fishino.
Bene, a questo punto possiamo affermare che la nostra applicazione è terminata!
Ma ora viene spontanea una domanda: "non volevamo fare un'App per il nostro smartphone?" Ebbene....presto fatto! Il bello delle Qt sta proprio qui: basta cambiare il kit di compilazione (vedete come fare nella prima puntata) ed il nostro programma per PC desktop Windows viene istantaneamente

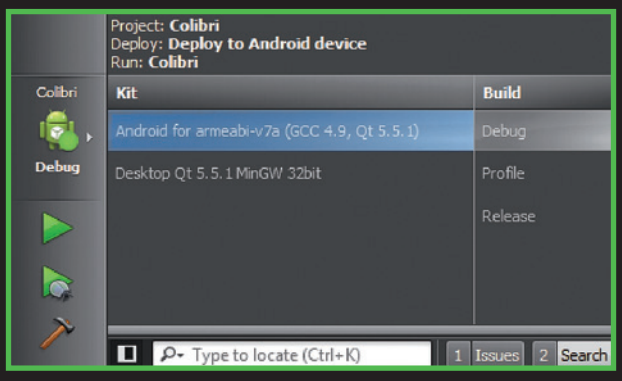


Fig. 2 - Esecuzione dello scheletro del programma.

Listato 2

```
#ifndef COLIBRI_H
#define COLIBRI_H

#include <QMainWindow>

#include <QUdpSocket>

// udp packet codes
// codici nei pacchetti UDP
typedef enum
{
    UDP_EMPTY             = 0,
    UDP_SETLIGHT = 1,
    UDP_GETLIGHT = 2,
    UDP_FIND              = 0x55

} UDP_PACKETS_CODES;

#define UDP_PORT  47777

namespace Ui {
class Colibri;
}

class Colibri : public QMainWindow
{
    Q_OBJECT

public:
    explicit Colibri(QWidget *parent = 0);
    ~Colibri();

private slots:
    void on_redSlider_sliderMoved(int position);

    void on_greenSlider_sliderMoved(int position);

    void on_blueSlider_sliderMoved(int position);

    void on_whiteSlider_sliderMoved(int position);

    void on_deviceCombo_currentIndexChanged(int index);

private:
    Ui::Colibri *ui;

    // the server socket
    QUdpSocket *udpSocket;

    // send an UDP packet to an host
    void sendUdpPacket(QByteArray const &packet, _
    QHostAddress addr);

    // wait for and get an UDP packet from host
    QByteArray receiveUdpPacket(quint16 timeout, _
    QHostAddress &sender, quint16 &port);
    QByteArray receiveUdpPacket(quint16 timeout);

    // discover all connected colibrì devices
    void discoverDevices(void);

    // get device values
    void getDeviceValues(void);

    // set device values
    void setDeviceValues(void);
};

#endif // COLIBRI_H
```

Listato 3

```
#include "colibri.h"
#include "ui_colibri.h"

#include <QTime>

Colibri::Colibri(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::Colibri)
{
    ui->setupUi(this);

    ui->redSlider->setMinimum(0);
    ui->redSlider->setMaximum(255);
    ui->greenSlider->setMinimum(0);
    ui->greenSlider->setMaximum(255);
    ui->blueSlider->setMinimum(0);
    ui->blueSlider->setMaximum(255);
    ui->whiteSlider->setMinimum(0);
    ui->whiteSlider->setMaximum(255);

    // initialize UDP socket
    udpSocket = new QUdpSocket(this);
    udpSocket->bind(UDP_PORT, QUdpSocket::ShareAddress);

    // discover available devices
    discoverDevices();
}

Colibri::~Colibri()
{
    delete ui;
}

void Colibri::on_redSlider_sliderMoved(int /* position */)
{
    setDeviceValues();
}

void Colibri::on_greenSlider_sliderMoved(int /* position */)
{
    setDeviceValues();
}

void Colibri::on_blueSlider_sliderMoved(int /* position */)
{
    setDeviceValues();
}

void Colibri::on_whiteSlider_sliderMoved(int /* position */)
{
    setDeviceValues();
}

void Colibri::on_deviceCombo_currentIndexChanged _
int /* index */)
{
    getDeviceValues();
}

// send an UDP packet to an host
void Colibri::sendUdpPacket(QByteArray const &packet, _
QHostAddress addr)
{
    QUdpSocket *sock = new QUdpSocket;
    sock->writeDatagram(packet, addr, UDP_PORT);
    delete(sock);
}

// wait for and get an UDP packet from host
QByteArray Colibri::receiveUdpPacket(quint16 timeout, _
QHostAddress &sender, quint16 &port)
{
    QByteArray res;
    res.clear();
```

(Continua)

trasformato in un'app per Android!
Prima di tutto, però, è d'obbligo una premessa: affinché tutto ciò sia possibile occorre attivare la "modalità sviluppatore" nel nostro cellulare, poi abilitare l'esecuzione di applicazioni non fidate, altrimenti non potremo caricare la nostra app. Poichè ogni modello di cellulare è differente, consigliamo una ricerca in Internet per scoprire qual è la procedura corretta per attivare la modalità sviluppatore sul vostro modello di smartphone.
Fatto questo, colleghiamo il telefono al

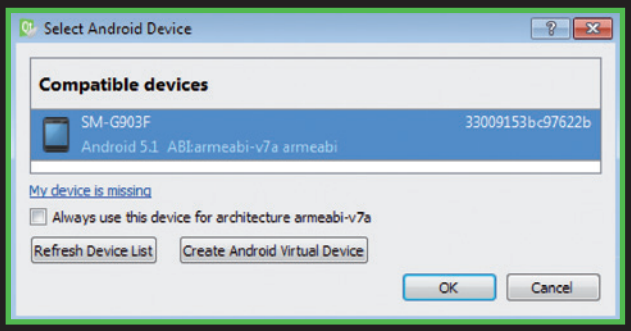


Fig. 3 - Dispositivi connessi.

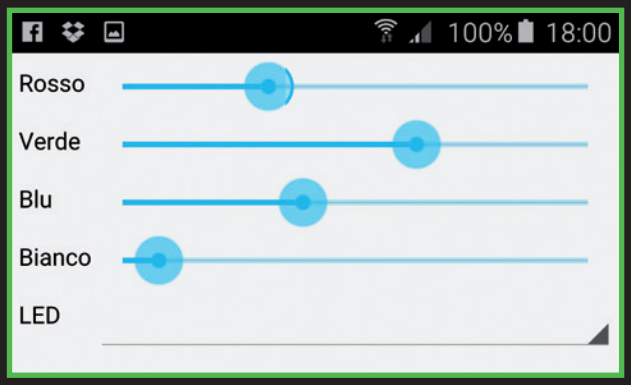


Fig. 4 - Programma in esecuzione.

computer utilizzato per lo sviluppo, tramite il cavo USB: dovrebbe apparire sullo schermo una richiesta di autorizzazione simile a quella proposta dalla **Fig. 1**. È indispensabile accettare tale richiesta; consigliamo inoltre di selezionare la casella "Consenti sempre da questo PC" per evitare ulteriori richieste future.
Nel caso la schermata non appaia, scollegare e ricollegare lo smartphone dopo qualche secondo.

Nella schermata che appare (riferitevi alla **Fig. 2**) premete sul simbolo del PC che appare in basso a sinistra, sopra il triangolino verde che avete usato in precedenza per eseguire lo scheletro del programma; poi selezionate il Kit Android e l'icona del PC si trasformerà nell'usuale icona Android. A questo punto è sufficiente premere il solito triangolino verde (RUN) per caricare l'applicazione sullo smartphone; la prima volta che verrà fatto ciò, nello schermo del computer apparirà una finestra di dialogo contenente la lista dei dispositivi Android connessi (**Fig. 3**). Nel nostro caso abbiamo solo un device, quindi la scelta è obbligata, ma se non fosse così dovrete scorrere fino a selezionare il device con cui state operando. Mettendo il segno di spunta sulla casella "Always use this device...." potremo evitare ulteriori richieste di scelta in futuro. Il programma verrà dunque ricompilato in formato Android, caricato sul dispositivo e lanciato (**Fig. 4**).
Come potete notare, a fronte di un'installazione piuttosto laboriosa le Qt offrono un vantaggio enorme nello sviluppo delle applicazioni portabili tra vari device; dati i limiti di spazio a disposizione ne abbiamo ovviamente solo saggiato le potenzialità, ma in rete è disponibile una documentazione abbondante, ed in seguito svlilupperemo altre applicazioni con questo toolkit. Terminato lo sviluppo dell'app Android, passiamo ora allo sketch che dovrà essere caricato sulla scheda Fishino per far funzionare il controllo ed alle connessioni da effettuare con il driver Colibrì (**Listato 4**).

Lo sketch per Fishino

Il firmware da caricare è veramente corposo e il relativo listato occuperebbe diverse pagine: per uno spazio che in questo numero non abbiamo e che esigenze grafiche non si potrebbe trovare. Quindi per conciliare questa esigenza inseriremo qui solo i punti salienti dello sketch, che è comunque scaricabile per intero dal nostro sito web *www.elettronicain.it*.
Ad inizio sketch, dopo i soliti include (#include "Fishino.h", #include "SPI.h", #include "Flash.h" e #include "EEPROM.h") troviamo la seguente porzione di codice:

```
// ogni colibrì richiede 4 uscite PWM
// questo array è utilizzato per connettere le
uscite con i 4 colori
```

Listato 3 (segue)

```
    // wait for incoming answers
    QTime timer;
    timer.start();
    while(timer.elapsed() < timeout)
    {
        // if there are pending packets....
        if(udpSocket->hasPendingDatagrams())
        {
            // build res array

            // read it
            res.resize(udpSocket->pendingDatagramSize());
            udpSocket->readDatagram(res.data(), _
            res.size(), &sender, &port);

            return res;
        }
    }
    return res;
}

QByteArray Colibri::receiveUdpPacket(quint16 timeout)
{
    QHostAddress host;
    quint16 port;
    return receiveUdpPacket(timeout, host, port);
}

void Colibri::discoverDevices()
{
    // build UDP discovery packet
    QByteArray arr(1, UDP_FIND);
    sendUdpPacket(arr, QHostAddress::Broadcast);

    // clear droplist
    ui->deviceCombo->clear();

    // wait for incoming answers
    QByteArray res;
    do
    {
        // try to read the packet
        QHostAddress host;
        quint16 port;
        res = receiveUdpPacket(500, host, port);
        qDebug() << "Packet size:" << res.size() << "\n";

        // if not done, leave
        if(!res.size())
            break;

        // if packet has a single byte, it's our _
        // own query packet
        // just discard it
        if(res.size() <= 1)
            continue;

        QString deb;
        for(int k = 0; k < res.size(); k++)
            deb += QString("0x%1").arg((int)res[k], _
            0, 16) + ":";
        qDebug() << deb;

        quint8 const *resPtr = (quint8 const *)res.data();
        if(*resPtr++ == UDP_FIND)
        {
            quint8 nDevices = *resPtr++;
            for(quint8 iDev = 0; iDev < nDevices; _
            iDev++)
            {
                // device number
                quint8 devNum = *resPtr++;

                // device name
QString name = (const char *)resPtr;
              resPtr += strlen((const char *)resPtr) + 1;
                QList<QVariant> list;
```

(Continua)

Listato 3 (segue)

```
                list << devNum << host.toIPv4Address();
                  ui->deviceCombo->addItem(name, list);
              }
          }

    }
    while(true);
}

// get device values
void Colibri::getDeviceValues(void)
{
    // get current device index
    int comboIdx = ui->deviceCombo->currentIndex();
    if(comboIdx == -1)
        return;

    // get device IP and number from combobox data
    QList<QVariant> list =
    ui->deviceCombo->currentData().value<QList<QVariant> >();
    qint8 device = list[0].value<qint8>();
    QHostAddress addr(list[1].value<quint32>());

    // build the needed query UDP packet
    QByteArray packet;
    packet.append(UDP_GETLIGHT).append(device);

    // send the packet
    sendUdpPacket(packet, addr);

    // read response back
    packet = receiveUdpPacket(200);

    qDebug() << "Packet size:" << packet.size() << "\n";

    // response packet must be 6 bytes long
    if(packet.size() == 6 && (quint8)packet[0] == UDP_GETLIGHT)
    {
       quint8 const *packP = (quint8 const *)packet.data() + 2;
       ui->redSlider->setValue(*packP++);
       ui->greenSlider->setValue(*packP++);
       ui->blueSlider->setValue(*packP++);
       ui->whiteSlider->setValue(*packP++);
    }

}

// set device values
void Colibri::setDeviceValues(void)
{
    qint8 r, g, b, w;

    // get current device index
    int comboIdx = ui->deviceCombo->currentIndex();
    if(comboIdx == -1)
        return;

    // get slider values
    r = ui->redSlider->value();
    g = ui->greenSlider->value();
    b = ui->blueSlider->value();
    w = ui->whiteSlider->value();

    // get device IP and number from combobox data
    QList<QVariant> list =
    ui->deviceCombo->currentData().value<QList<QVariant> >();
    qint8 device = list[0].value<qint8>();
    QHostAddress addr(list[1].value<quint32>());

    // build the needed query UDP packet
    QByteArray packet;
    packet.append(UDP_SETLIGHT).append(device);
    packet.append(r).append(g).append(b).append(w);

    // send the packet
    sendUdpPacket(packet, addr);
}
```

```
typedef uint8_t DEVICE_CONNECTIONS[4];
```

che definisce il tipo DEVICE_CONNECTIONS come un array di 4 numeri interi, che corrispondono agli altrettanti I/O digitali ai quali si vuole connettere il modulo controller RGBW Colibrì.
Successivamente troviamo le usuali variabili riguardanti la connessione wireless (SSID, PASSWORD, IP, eccetera) e dopo, la porzione di codice mostrata nel **Listato 4**, nella quale viene definita la porta UDP di comunicazione ed è rappresentata la mappatura tra i controller Colibrì connessi alla scheda Fishino e gli I/O cui fisicamente si collegano; in questo caso, come specificato nei commenti, abbiamo duplicato lo stesso device per motivi didattici, mancando sufficienti PWM per connettere due device.
Più avanti troviamo la variabile che realizza il server UDP e le costanti che identificano i vari pacchetti scambiati :

```
// il client/server UDP
FishinoUDP udp;

// codici nei pacchetti UDP
typedef enum
{
        UDP_EMPTY         = 0,
        UDP_SETLIGHT      = 1,
        UDP_GETLIGHT      = 2,
        UDP_FIND = 0x55

} UDP_PACKETS_CODES;
```

Seguono alcune routine (**Listato 5**) che processano i singoli pacchetti, che omettiamo per brevità, descrivendo qui solo la più interessante, ovvero quella che risponde ad una richiesta di broadcast trasmettendo i dati del device connesso al server, permettendone l'identificazione automatica.
Descriviamo infine il "cuore" dell'applicazione, ovvero la funzione che si occupa di identificare i pacchetti ricevuti e ad inviarli alle rispettive funzioni di gestione; la vedete nel **Listato 6**.
Nel **Setup()**, a parte le consuete inizializzazioni avviamo il server UDP:

```
// inizia l'ascolto dei pacchetti UDP alla porta specificata
Serial << F("Starting connection to server...\n");
udp.begin(UDP_PORT);
```

Invece nel loop controlliamo l'eventuale arrivo di pacchetti UDP e, nel caso, li leggiamo

Listato 4

```
// inserire qui la porta UDP in attesa dei pacchetti
#define UDP_PORT           47777

// inserire qui il nome dell'applicazione che sarà inviato al client dopo la richiesta di discovery
#define APP_NAME           "fishnlights"

// connessioni tra gli output ed i colibrì
// ogni colibrì richiede 4 uscite PWM (R, G, B, W)
// siccome Fishino ha solo 6 PWM disponibili, li abbiamo semplicemente
// duplicati per mostrare nell'App 2 luci connesse
DEVICE_CONNECTIONS DEVICES[] =
{
        {3, 5, 6, 9},
        {3, 5, 6, 9},
};
```

Listato 5

```
// processa i pacchetti con codice FIND
bool processFind(uint8_t const *_packet)
{
        Serial << F("GOT FIND PACKET\n");
        char buf[DEVICE_NAME_MAX + 1];

        uint16_t packetLen = 2;

        // calcola la dimensione del pacchetto di risposta
        for(uint8_t i = 0; i < numDevices; i++)
        {
                Serial << F("Reading name of #") << (int)i << F(" device\n");
                if(!getDeviceName(i, buf, DEVICE_NAME_MAX))
                        return false;
                Serial << F("GOT :") << buf << "\n";
                packetLen += strlen(buf) + 2;
        }
        Serial << F("PACKET SIZE : ") << packetLen << "\n";
        uint8_t *packet = (uint8_t *)malloc(packetLen);
        uint8_t *packP = packet;
        *packP++ = UDP_FIND;
        *packP++ = numDevices;
        for(uint8_t i = 0; i < numDevices; i++)
        {
                // first the device number
                *packP++ = i;

                // then the device name
                getDeviceName(i, buf, DEVICE_NAME_MAX);
                strcpy((char *)packP, buf);
                packP += strlen(buf) + 1;
        }

        // reinvia il pacchetto al mittente
        Serial << F("Sending packet to ") << udp.remoteIP() << " at port " << UDP_PORT << "\n";
        udp.beginPacket(udp.remoteIP(), UDP_PORT);
        udp.write(packet, packetLen);
        return udp.endPacket();
}
```

e li inviamo alla **processPacket()** vista in precedenza; la rispettiva porzione di codice è visibile nel **Listato 7**.
La personalizzazione dello sketch è semplicissima: nella parte iniziale di configurazione, a parte i soliti dati della rete WiFi, occorre inserire questa struttura:

```
DEVICE_CONNECTIONS DEVICES[] =
{
        {3, 5, 6, 9},
        {3, 5, 6, 9},
};
```

che contiene la mappatura dei canali RGBW con i corrispondenti pin di Fishino, che devono essere dotati di uscita PWM.
Nell'esempio abbiamo semplicemente duplicato una lampada sugli stessi pins, in mancanza

Listato 6

```
// processa i pacchetti UDP
bool processPacket(uint8_t const *packet)
{
        Serial << F("GOT PACKET\n");
        uint8_t type = packet[0];
        packet++;
        switch(type)
        {
                case UDP_SETLIGHT:
                        return processSetLight(packet);

                case UDP_GETLIGHT:
                        return processGetLight(packet);

                case UDP_FIND:
                        return processFind(packet);

                default:
                        Serial << F("UNKNOWN PACKET ") << (int)type << "\n";
                        return false;
        }
}
```

Listato 7

```
// ciclo infinito
void loop(void)
{
        // check incoming packets
        int packetSize = udp.parsePacket();
        if (packetSize)
        {
                uint8_t *buf = (uint8_t *)malloc(packetSize);
                if(udp.read(buf, packetSize))
                        processPacket(buf);
                else
                        Serial << F("Error reading packet, size is ") << packetSize << F(" bytes\n");
                free(buf);
        }
}
```

degli 8 necessari per due lampade; in un futuro articolo proporremo una versione estesa dello sketch pensata per consentire, tramite l'uso della scheda di ampliamento OCTOPUS (la quale dispone di ben 16 uscite PWM), di controllare quattro Led RGBW per ciascuna scheda Fishino, o fino a 32 LED RGBW sovrapponendo un massimo di otto interfacce.
Il funzionamento dello sketch è piuttosto semplice: nel setup, dopo la connessione alla rete WiFi (o la creazione di un Access Point, se si preferisce) e dopo aver definito tutte le uscite PWM della scheda Fishino ed averle azzerate, viene messo in ascolto un client UDP sulla porta 47777:

```
udp.begin(UDP_PORT);
```

Nel loop si esegue ciclicamente un controllo su eventuali pacchetti ricevuti e, nel caso, viene chiamata la funzione **processPacket()** che esamina i dati nel pacchetto e li smista alla funzione di gestione corrispondente al primo byte in esso contenuto: **processSetLight**, **processGetLight** e **processFind**, rispettivamente, per impostare o leggere il valore dei PWM o per segnalare la presenza della scheda nella rete.
Si può notare che sono previste l'impostazione e la lettura di un "nome luce" da EEPROM; questa è una funzionalità in via di sviluppo, sia dal lato app che dal lato Fishino, che permetterà di dare un nome alle varie utenze, le quali attualmente vengono visualizzate solo tramite numero di indirizzo IP.
Bene, con questo abbiamo terminato la lunga, ma speriamo utile, esposizione di questo progetto, che, siamo certi, sarà stata l'occasione per conoscere le librerie QT e il mondo di tool di sviluppo che ne permette l'utilizzo ottimale. ■