Arduino pour la domotique

Marc-Olivier Schwartz
Blogueur et entrepreneur dans le domaine du matériel libre

Traduit de l'américain
par Vincent Briois

DUNOD

Cet ouvrage en français est la traduction augmentée et remaniée de l'ouvrage *Home Automation with Arduino*, publié par Marc-Olivier Schwartz.

Maquette de couverture : Misteratomic

© Dunod, 2015
5 rue Laromiguière, 75005 Paris
www.dunod.com

ISBN 978-2-10-073049-0

Table des matières

Avant-propos

J'ai découvert l'univers fascinant de la domotique à l'occasion d'une visite chez l'un de mes amis. Tout semblait étonnement simple à mes yeux : éclairage qui s'active automatiquement à la tombée du jour, transmission du relevé des températures de chaque pièce de la maison vers un serveur central, configuration du statut de l'ensemble des détecteurs d'alarme à partir d'un téléphone portable...

La contrainte principale à cette époque était de taille : le système était spécialement conçu pour cette maison par des entreprises privées. Son acquisition était donc réservée à une clientèle aisée. D'ailleurs, nombreux sont ceux qui continuent à croire que la domotique reste financièrement inaccessible.

Ces systèmes « propriétaires » présentent selon moi un autre point faible : l'impossibilité de les contrôler soi-même. Vous devez suivre un processus bien défini par le fabricant à propos de l'unité centrale, des capteurs et de l'interface. Si par exemple un des capteurs de votre système est déficient, il devra être remplacé par un autre de la même marque. Je me souviens avoir essayé de manipuler un de ces systèmes vendus dans le commerce. Lorsque je tentais de rendre l'utilisation d'un capteur plus simple ou de corriger un bogue dans l'interface, c'était tout simplement impossible.

Bien sûr, créer sa propre installation domotique n'est pas un concept nouveau. Je me rappelle de mes premières manipulations sur un microcontrôleur en 2003. Cela me semblait assez simple pour quelqu'un du monde de l'ingénierie. Ces systèmes n'offraient que des applications limitées et chacun d'eux nécessitait l'apprentissage de connaissances spécifiques sur sa propre plateforme. De plus, les kits d'évaluation de ces microcontrôleurs étaient également assez onéreux.

Ces dernières années ont vu la montée en puissance d'un nouveau mouvement, celui du matériel libre (*open hardware*). Ce mouvement est semblable à celui du monde du logiciel et consiste à donner la possibilité à quiconque d'accéder aux plans de produits physiques et ainsi de les personnaliser. De ce mouvement est née une plateforme qui révolutionne le monde de l'électronique : la plateforme Arduino. Pourvue d'un environnement simple et agréable à utiliser, Arduino facilite la programmation de microcontrôleurs.

En ce qui me concerne, cela a changé complètement ma vision de la domotique. La réalisation d'installations domotiques est désormais accessible à la plupart de ceux qui

ont des notions en électronique et en programmation. Cet ouvrage est là pour vous montrer la démarche à suivre.

Depuis la parution initiale de ce livre au format numérique, des milliers d'utilisateurs ont mis en application les principes décrits afin de créer leurs propres installations domotiques. J'ai également reçu un certain nombre de commentaires pertinents qui m'ont aidé à améliorer l'ouvrage.

Cependant, beaucoup de ces retours regrettaient la difficulté relative des projets présentés. Ces commentaires évoquaient plus souvent le nombre de langages de programmation abordés que le contenu même des projets. Pour les projets impliquant par exemple la communication entre une carte Arduino et un ordinateur, j'avais utilisé à la fois les langages Python, PHP, HTML et JavaScript. Il était également nécessaire d'installer et de lancer un serveur web sur son ordinateur, ce qui rendait la tâche encore plus complexe.

C'est la raison pour laquelle j'ai effectué des modifications dans la nouvelle édition de ce livre en privilégiant la simplicité et la clarté.

En ce qui concerne la mise en place de l'interface sur un ordinateur, je n'ai conservé qu'un seul langage : **JavaScript**. J'ai supprimé l'ensemble des références aux autres langages tels que Python et PHP. Côté serveur, j'ai opté pour l'utilitaire de programmation **Node.js** dans JavaScript. Ainsi, il n'est plus nécessaire d'installer un serveur web sur votre ordinateur.

J'espère que ces changements vous paraîtront pertinents et que vous mettrez en application l'ensemble des connaissances acquises dans ce livre pour des projets encore plus élaborés avec Arduino.

À propos de l'auteur

Je suis ingénieur en électronique, entrepreneur et auteur, titulaire d'un diplôme d'ingénieur Supélec, la plus prestigieuse école de France dans ce domaine, et d'un master en microtechnique obtenu à l'École polytechnique fédérale de Lausanne (EPFL).

J'ai plus de cinq années d'expérience professionnelle en ingénierie électronique et je m'intéresse à tout ce qui touche l'électronique, la domotique, la plateforme Arduino, les projets à base de matériel libre et l'impression 3D.

Je travaille en tant qu'entrepreneur à plein-temps depuis 2011 et gère des sites Internet fournissant des informations sur le matériel libre. Je crée également en parallèle mon propre matériel libre.

Remerciements

J'adresse mes remerciements à tous mes amis qui m'ont apporté leur soutien tout au long de l'écriture de ce livre et de tous les projets sur lesquels j'ai travaillé.

Je remercie également mes parents pour leur soutien durant cette période mais aussi dans les différents projets que j'ai pu mener en général, lors des périodes difficiles et moins difficiles.

Enfin, je tiens à remercier Sylwia, ma compagne, pour son soutien permanent. C'est elle qui me donne l'inspiration nécessaire pour aborder chaque journée avec sérénité et la force de persévérer dans mes projets.

Merci à tous.

RESSOURCES NUMÉRIQUES ET SITE WEB COMPLÉMENTAIRE

Cet ouvrage dispose d'un site web complémentaire en anglais, Open Home Automation, accessible à cette adresse :

http://www.openhomeautomation.net

Vous y trouverez davantage de projets et de ressources en matière de domotique et de matériel libre.

Vous pouvez retrouver l'ensemble des codes présents dans cet ouvrage à cette adresse :

https://github.com/openhomeautomation/home-automation-arduino

Ce dépôt **GitHub** créé spécialement pour le livre contient les codes les plus récents des projets abordés.

Partie I

Introduction à Arduino

1 Premiers pas

Avant de vous entraîner dans la réalisation de votre premier projet de domotique ouvert, je souhaiterais vous parler de la plateforme Arduino que nous allons utiliser tout au long de l'ouvrage.

UNE PLATEFORME RÉVOLUTIONNAIRE

Les débuts de Arduino remontent à 2005, lorsque ses créateurs, Massimo Banzi[1] et David Cuartielles, travaillaient sur la mise au point d'un appareil simple à programmer par un public de non-spécialistes afin que leurs étudiants en design puissent construire des projets impliquant des microcontrôleurs.

La plateforme Arduino n'a pas été conçue dans le seul but de fonctionner avec des cartes ou des microcontrôleurs, elle constituait également une solution complète matérielle et logicielle plus simple à utiliser par rapport aux autres microcontrôleurs.

Côté matériel, Arduino est un microcontrôleur monocarte, équipé en général d'un microcontrôleur 8 bits Atmel AVR, même si de nouveaux modèles tels que la carte Arduino Due disposent d'un processeur ARM 32 bits. Nous n'aurons pas besoin d'une carte aussi puissante pour nos projets ; de ce fait, nous utiliserons la carte Arduino la plus courante : le modèle **Arduino Uno**.

Les broches des cartes Arduino ont la particularité d'être disposées toujours de la même façon, ce qui rend ainsi les connexions aux cartes d'extension (*shield*) très aisées. Ces dernières fournissent des fonctionnalités complémentaires à la carte, comme la possibilité de commander un moteur à courant continu dans le domaine de la robotique, ou de se connecter à un téléphone portable en Bluetooth.

Selon moi, c'est bien la partie logicielle qui rend la plateforme Arduino si puissante.

Pour programmer une carte Arduino, vous pouvez vous servir du logiciel officiel (en téléchargement libre) et utiliser ensuite un langage proche de C++ pour écrire le code que vous désirez charger sur la carte.

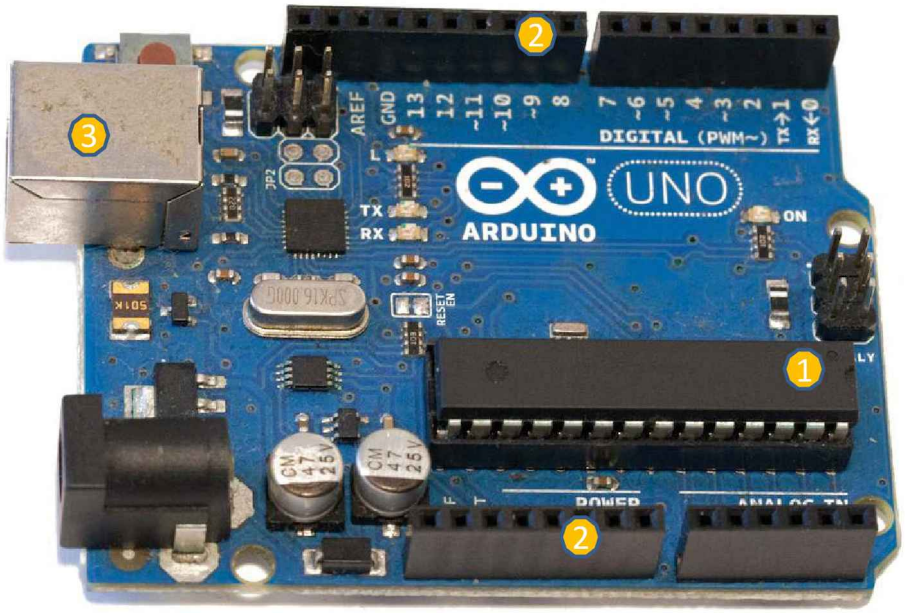
En comparaison avec d'autres microcontrôleurs, la programmation de la carte est très abordable ; vous pourrez très aisément lui faire réaliser ce que vous souhaitez. L'éclairage d'une LED ne nécessitera avec Arduino qu'une seule ligne de code. Avec d'autres microcontrôleurs, il vous faudrait en écrire plusieurs.

La plateforme Arduino dispose d'une très vaste communauté, ce qui est aussi un atout majeur. Il existe, grâce à cela, beaucoup de documentation sur le site officiel Arduino

1. Voir *Démarrez avec Arduino,* par M. Banzi et M. Shiloh (3e édition, Tous Makers !, Dunod, 2015).

(www.arduino.cc) pour chaque fonction utilisée. Des tutoriels relatifs aux fonctions les plus courantes sont également à votre disposition.

Voici maintenant des informations complémentaires au sujet de la carte **Arduino Uno** avec un aperçu de la carte dont je me suis servi pour réaliser l'ensemble des projets décrits dans ce livre :



La carte en elle-même est de très petite taille.

Ce que vous voyez dans le coin inférieur droit de l'illustration est un **microcontrôleur Atmel** (❶), le « cerveau » de la carte. C'est lui qui reçoit le logiciel que nous allons développer dans le cadre de nos projets de domotique.

En haut et en bas de la carte, vous pouvez voir une série de **connecteurs** (❷). Ils serviront à connecter les signaux d'entrée et de sortie comme les entrées analogiques, les entrées et sorties numériques ainsi qu'à connecter des composants extérieurs aux tensions de référence (masse et 5 V).

Enfin, vous pouvez voir le **port USB** (❸) dans le coin supérieur gauche qui nous permettra de relier la carte à votre ordinateur.

CE QU'IL FAUT SAVOIR EN MATIÈRE D'ÉLECTRONIQUE

Ce livre n'aborde pas les principes généraux de l'électronique ; vous trouverez d'autres ouvrages plus adaptés à cet effet.

Il est question dans cet ouvrage de vous montrer comment réaliser une installation domotique. Vous y apprendrez à connecter différents composants, capteurs et autres appareils à la plateforme Arduino.

Cependant, afin de comprendre le mode de fonctionnement de ces composants, il vous faut connaître quelques principes fondamentaux. Vous trouverez dans cette partie un bref aperçu des principes évoqués dans les projets du livre.

Les principales variables utilisées en électronique

Il existe un bon nombre de variables pour définir un circuit. Nous allons uniquement aborder les plus importantes.

Un circuit électrique est comparable à un circuit hydraulique. Afin que l'eau puisse effectuer naturellement son circuit d'un point A à un point B, il est indispensable d'avoir un écart de hauteur entre ces deux points.

En électronique, cet écart est appelé **tension** (U), et cette grandeur est mesurée en volts.

Le débit d'eau entre un point A et un point B est comparable à un flux d'électrons au sein d'un circuit électrique. On appelle **courant électrique** ce flux d'électrons. Il est représenté par la lettre I.

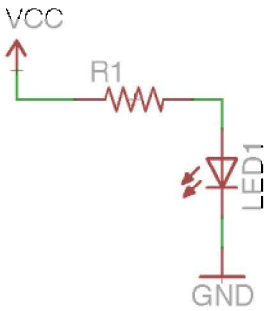
La **puissance** P dissipée par un équipement donné est obtenue en multipliant la tension par l'intensité. Elle est mesurée en watts (W) :

$$P = U * I$$

Représentation d'un circuit de base

Un circuit électrique peut être représenté à l'aide de symboles normalisés.

Voici par exemple un circuit simple comprenant une source de tension VCC (alimentation tension continue), une résistance R1, une LED appelée LED1 et une broche de masse (GND).



Par la suite, nous verrons plus en détail quelques-uns de ces composants mais nous allons pour l'instant nous contenter d'identifier les plus courants.

La première étape dans la lecture d'un schéma électrique consiste à repérer la source d'alimentation et les broches de masse.

Dans le cas présent, la source est représentée par la broche VCC. Dans la plupart des projets de ce livre, cette source sera d'une tension de 5 V.

La broche de masse est représentée ici par la broche GND.

Après avoir localisé ces deux broches, vous pourrez vous intéresser aux composants. Dans ce cas, il s'agit d'une résistance et d'une LED.

Sources d'alimentation

Dans le premier circuit de cette partie, la source d'alimentation était une broche nommée « VCC ». Théoriquement, VCC peut être n'importe quelle source de tension, mais elle est par convention une source d'alimentation positive à basse tension (de 3,3, 5 ou 12 V en général).

On utilisera en général le port USB de la carte Arduino pour alimenter les projets présentés dans cet ouvrage. Toutefois, il vous est possible d'alimenter votre carte Arduino à partir d'autres sources d'alimentation telles que des alimentations régulées directement branchées à une prise murale (veillez cependant à ne pas dépasser la tension maximale tolérée par votre carte Arduino) ou à des piles.

Résistance

La résistance est un composant clé au sein d'un circuit électrique. En reprenant l'exemple du circuit hydraulique, la résistance joue le rôle de limiteur de débit d'eau (ici d'électrons) à un endroit donné du circuit.

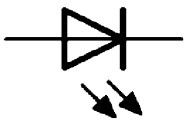
Afin de quantifier cette limitation, nous introduisons une nouvelle variable, *R*, qui s'exprime en ohms (Ω). On appelle **loi d'Ohm** la formule reliant tension (*U*), intensité du courant (*I*) et résistance (*R*) selon la relation :

$$U = R * I$$

LED

La LED (diode électroluminescente) est le composant le plus utilisé en tant qu'élément de signalisation et de test dans un circuit. Lorsque du courant électrique (d'une intensité de 20 mA en moyenne) traverse une LED, celle-ci produit une lumière rouge, bleue, verte ou encore blanche selon son type.

Sur une carte Arduino, elle sert par exemple à vérifier que la carte est bien alimentée ou à indiquer le bon fonctionnement de la communication série. Les LED peuvent également être utilisées pour effectuer des tests du logiciel (sur la broche n° 13).

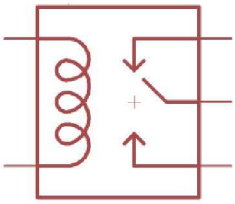


Comme nous l'avons évoqué dans le cadre du premier circuit, on associe souvent les LED à des résistances afin de limiter le courant qui y circule. Gardez bien à l'esprit que les pattes des LED ne sont pas toutes identiques. Une alimentation électrique positive (VCC, par exemple) doit être reliée à la patte gauche (sur le schéma) de la LED, l'anode et l'autre patte, la cathode, doit quant à elle être connectée à la masse. La cathode est facilement repérable puisqu'elle dispose de la patte la plus courte.

Relais

En domotique, notre objectif est de pouvoir allumer ou éteindre des composants comme une lampe de la même manière qu'un interrupteur mural. Cela est rendu possible grâce à des relais qui sont en fait des commutateurs électromécaniques.

Un relais est composé principalement de deux parties : la partie « commande » du relais, située à gauche, est ce que l'on appelle la **bobine**. Lorsque la bobine est soumise à une tension (de 5 V pour les relais utilisés dans ce livre), l'état de l'autre partie du relais change (de fermé à ouvert par ex.).



Cette seconde partie du relais présente l'avantage de tolérer une tension bien supérieure (jusqu'à 300 V pour ceux utilisés dans le livre) à celle supportée par la carte Arduino. C'est ce qui permet à la carte Arduino de piloter des appareils raccordés au réseau électrique, comme les lampes.

POUR ALLER PLUS LOIN

Cette partie sert à vous familiariser avec les principes de base en électronique ainsi qu'avec la majorité des composants maniés dans cet ouvrage.

Pour aller plus loin et développer vos compétences en électronique, je vous conseille dans un premier temps de faire des recherches sur Internet. Il sera aussi intéressant de vous reporter au chapitre *Ressources* du livre pour des suggestions de lecture sur Arduino et l'électronique en général.

2 Lire les données d'un capteur

Nous débuterons le premier chapitre de ce livre par un projet essentiel en domotique : la lecture de données mesurées par un capteur connecté à une carte Arduino. Nous allons connecter un capteur d'humidité et de température à la carte, écrire un sketch afin d'en lire les données et afficher le résultat dans le moniteur série.

MATÉRIEL ET LOGICIEL REQUIS

Dressons tout d'abord une liste des composants requis pour ce projet.

Premièrement, munissez-vous d'une carte Arduino. J'ai opté pour la carte Arduino la plus simple sur le marché, la carte **Arduino Uno**. Nous utiliserons cette carte tout au long de cet ouvrage.

Concernant le capteur, mon choix s'est porté sur le **DHT11**. Il s'agit d'un capteur numérique utilisé pour mesurer la température et l'humidité. Nous verrons qu'il est très simple d'utiliser ce capteur puisqu'il dispose de sa propre bibliothèque Arduino.

Vous aurez également besoin d'une **plaque d'essai** (*breadboard*) et de **fils de raccordement** pour établir les connexions entre la carte Arduino et le capteur.

LISTE DES COMPOSANTS

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Capteur DHT11 avec résistance de 4,7 kΩ (http://snootlab.com/adafruit/255-capteur-de-temperateur-et-d-humidite-dht11-extras-.html)
- Plaque d'essai (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

Côté **logiciel**, vous devrez vous procurer la bibliothèque Arduino et l'installer sur le capteur DHT. Les bibliothèques Arduino fournissent des fonctionnalités complémentaires à Arduino. Ainsi, il ne vous est pas nécessaire d'écrire du code vous-même à chaque fois que vous souhaitez ajouter un nouveau composant. Ici par exemple, la bibliothèque se chargera de la communication avec le capteur.

Vous pouvez télécharger la bibliothèque à l'adresse suivante :
https://github.com/adafruit/DHT-sensor-library

Pour installer la bibliothèque, il vous suffit d'extraire son dossier (nommé DHT) vers le dossier **/libraries** de votre dossier racine Arduino. Si ce dossier n'existe pas, il vous suffit de le créer, puis d'y placer le dossier DHT.

CONFIGURER LE MATÉRIEL

Nous allons maintenant configurer la partie matérielle. Comme vous pourrez le voir, ce projet est très simple à réaliser.

Voici le schéma complet du projet qui vous facilitera sans doute la tâche :

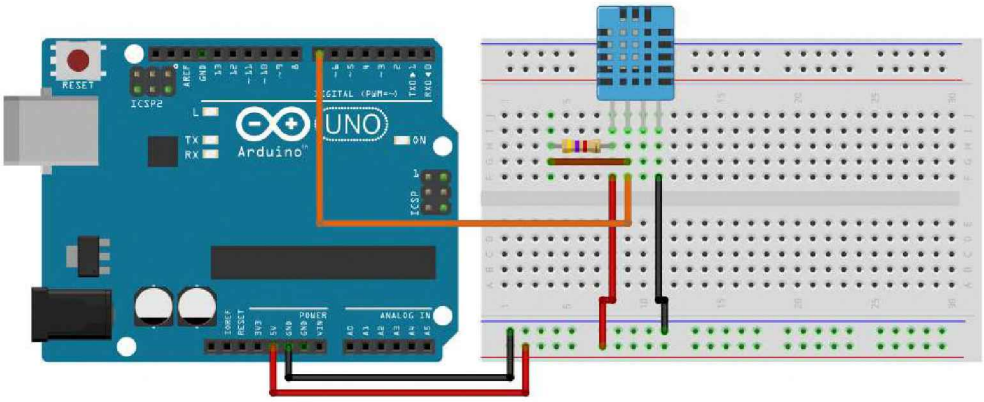


Figure 2.1 Image créée avec le logiciel Fritzing (http://fritzing.org/)

Il vous faut tout d'abord placer le capteur DHT11 sur la plaque d'essai. Voici les broches (*pins*) du capteur :

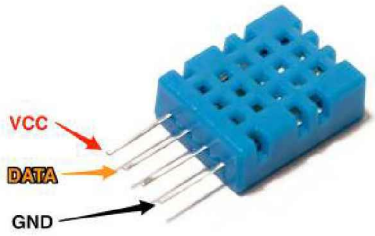
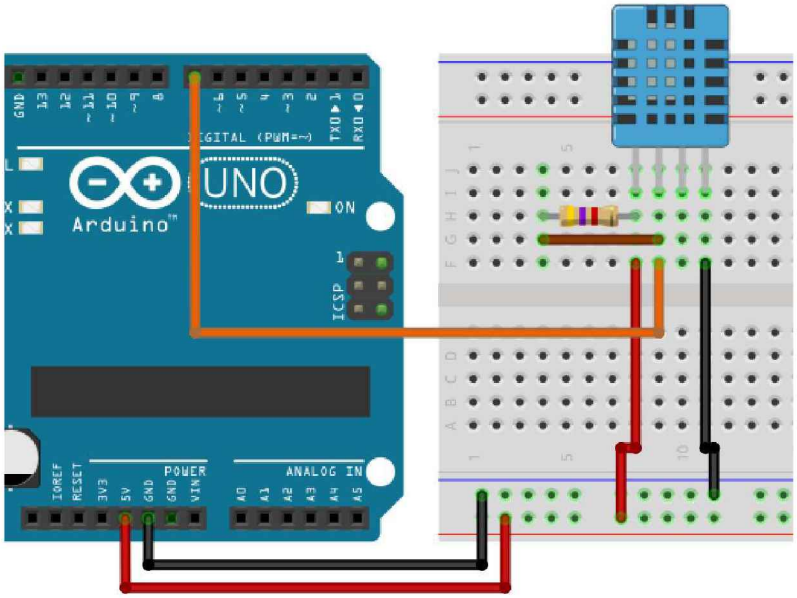


Figure 2.2 DHT

Une fois le capteur DHT en place sur la plaque d'essai, vous devez connecter la broche **n° 1** du capteur (VCC) à la broche 5 V de la carte Arduino.

Ensuite, connectez la dernière broche du capteur (GND) à la **broche GND** de la carte Arduino.

Après cette étape, connectez la broche DATA du capteur à la broche n° **7** de la carte Arduino. Pour finir, insérez la résistance de 4,7 kΩ (kilo-ohms) entre les broches VCC et DATA du capteur.



LIRE LES DONNÉES DU CAPTEUR

Nous allons maintenant écrire un sketch simple permettant de lire des données du capteur et de les visualiser au sein du moniteur série Arduino.

Vous trouverez ci-dessous le sketch Arduino complet pour ce chapitre :

```
// Code to measure data from the DHT11 sensor

// Libraries
#include "DHT.h"

// DHT sensor
#define DHTPIN 7
#define DHTTYPE DHT11

// DHT instance
DHT dht(DHTPIN, DHTTYPE);

void setup()
{
  // Initialize the Serial port
  Serial.begin(9600);
```

```
  // Init DHT
  dht.begin();
}


void loop()
{
  // Measure from DHT
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  // Display temperature
  Serial.print("Temperature: ");
  Serial.print((int)temperature);
  Serial.println(" C");

   // Display humidity
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println("%");

  // Wait 500 ms
  delay(500);
}
```

Voyons maintenant ce sketch en détail. Premièrement, on inclut la bibliothèque pour le capteur DHT :

```
#include "DHT.h"
```

Ensuite, il nous faut préciser sur quelle broche le capteur est connecté (7) et définir le type de capteur (DHT11) :

```
#define DHTPIN 7
#define DHTTYPE DHT11
```

Une fois cette étape réalisée, nous pouvons créer une instance de la bibliothèque DHT :

```
DHT dht(DHTPIN, DHTTYPE);
```

À présent, nous lançons le moniteur série dans la fonction `setup()` du sketch Arduino :

```
Serial.begin(9600);
```

Il nous faut aussi initialiser le capteur DHT11 :

```
dht.begin();
```

Dans la fonction `loop()` du sketch, nous pouvons désormais lire des données à partir du capteur :

```
float temperature = dht.readTemperature();
```

```
float humidity = dht.readHumidity();
```

Nous écrivons également ces données dans le moniteur série après chaque lecture. Voici par exemple le code permettant d'inscrire la température :

```
Serial.print("Temperature: ");
Serial.print((int)temperature);
Serial.println(" C");
```

Enfin, il est nécessaire de répéter ces étapes toutes les 500 ms :

```
delay(500);
```

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/intro-book

Vous pouvez charger le sketch sur la carte Arduino et ouvrir le moniteur série en vous assurant que la vitesse du port série est réglée à **9600**.

Vous devriez constater que les mesures du capteur sont affichées en continu :

```
Temperature: 24 C
Humidity: 35%
```

Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Assurez-vous en premier lieu que vous avez téléchargé et installé la bibliothèque du capteur DHT et que tous ses fichiers sont contenus dans un dossier nommé DHT au sein du dossier racine Arduino.
- Veillez aussi à ce que le capteur DHT soit correctement connecté à la carte Arduino.
- Enfin, assurez-vous d'utiliser la version la plus récente du code à partir du dépôt GitHub du livre.

POUR ALLER PLUS LOIN

Résumons ce que nous avons appris dans ce chapitre. Nous avons vu comment réaliser un projet très simple avec une carte Arduino, en lisant les données d'un capteur numérique d'humidité et de température.

À partir de ce que nous avons étudié dans ce chapitre, il est possible d'aller plus loin. Vous pouvez vous procurer un autre capteur, par exemple de lumière ou de pression barométrique. Vous trouverez de nombreux capteurs en faisant des recherches sur Internet ou en utilisant les liens du chapitre *Ressources* du livre.

3 Piloter un relais

Dans ce deuxième chapitre, nous allons voir comment envoyer une commande vers un relais *via* la carte Arduino.

Le **relais** est un élément clé en domotique étant donné qu'il permet de commander des appareils consommant beaucoup d'énergie (lampes) à partir d'une carte Arduino.

MATÉRIEL ET LOGICIEL REQUIS

Dressons tout d'abord la liste des composants requis pour ce projet.

Premièrement, munissez-vous d'une **carte Arduino**, la même que celle déjà choisie précédemment.

En ce qui concerne le **relais**, mon choix s'est porté sur un modèle 5 V de la marque Polulu :

Figure 3.1 Module relais

La marque du relais n'est pas un facteur très important. Il est cependant essentiel de choisir un relais 5 V pour qu'il puisse être piloté directement par la carte Arduino. Veuillez aussi vous assurer que la puissance maximale, le courant nominal et la tension nominale de votre relais correspondent à ce que vous voulez en faire.

Ensuite, vous aurez besoin de fils de raccordement mâle/femelle pour établir les différentes connexions entre la carte Arduino et le capteur.

LISTE DES COMPOSANTS

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Module relais (http://www.pololu.com/product/2480)
- Fils de raccordement mâle/femelle (http://snootlab.com/cables/23-kit-10-cordons-6-m-f.html)

CONFIGURER LE MATÉRIEL

Assemblons maintenant notre projet. Les connexions de ce projet sont très simples puisqu'aucune plaque d'essai n'est nécessaire.

Pour vous aider, voici le schéma du projet (la plaque d'essai apparaît dans l'unique but de visualiser les différentes connexions) :

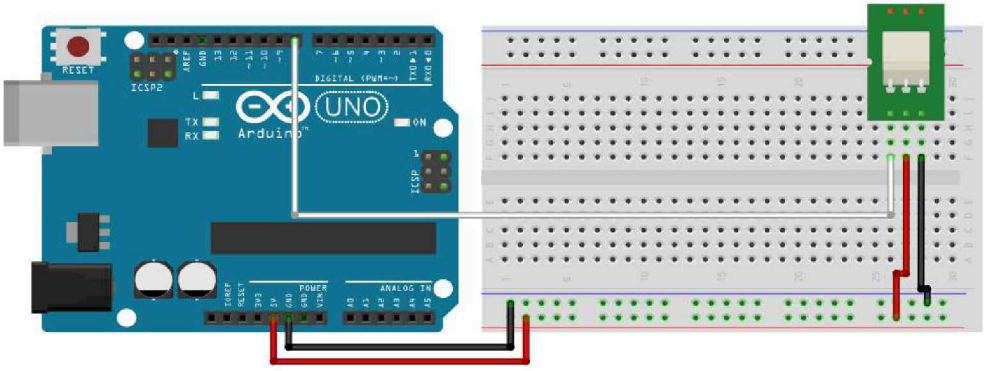
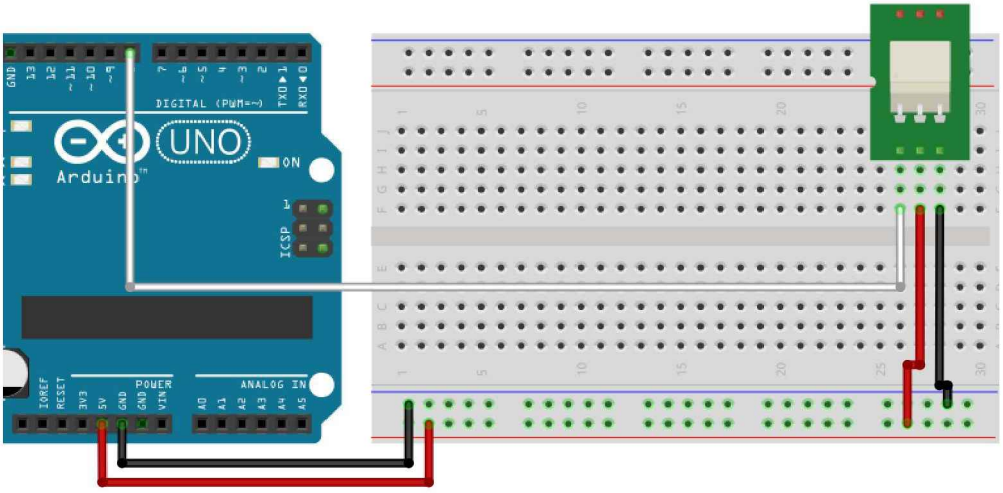


Figure 3.2 Image créée avec le logiciel Fritzing (http://fritzing.org/)

Un module relais dispose en principe de trois broches : deux broches pour l'alimentation électrique et la dernière pour le signal de commande permettant de l'activer ou de le désactiver.

Connectez tout d'abord la broche VCC du relais à la broche 5 V de la carte Arduino. Puis, connectez la broche GND du relais à la broche GND de la carte. Enfin, connectez la broche signal du relais (appelée SIG en général) à la broche n° 8 de la carte.

À ce stade, vous pouvez également connecter aux sorties du module relais l'appareil que vous désirez commander (ex. : une lampe). Cette opération n'est cependant pas recommandée ici car nous voulons vérifier en premier lieu que le relais fonctionne.



PILOTER LE RELAIS

Nous allons maintenant écrire un sketch simple pour commander le relais. En guise de test, nous allons simplement activer et désactiver le relais toutes les 5 secondes.

Vous trouverez ci-dessous le sketch complet de cette opération :

```
// Simple sketch to control the relay

// Relay pin
const int relay_pin = 8;

void setup() {
  pinMode(relay_pin,OUTPUT);
}

void loop() {

  // Activate relay
  digitalWrite(relay_pin, HIGH);

  // Wait for 5 seconds
  delay(5000);

   // Deactivate relay
  digitalWrite(relay_pin, LOW);

  // Wait for 5 seconds
  delay(5000);
}
```

Étudions maintenant ce code. Premièrement, on indique sur quelle broche le relais est connecté :

```
const int relay_pin = 8;
```

Ensuite, dans la fonction `setup()` du sketch, on configure cette broche en tant que sortie afin qu'elle puisse recevoir nos commandes :

```
pinMode(relay_pin,OUTPUT);
```

Après ça, dans la fonction `loop()` du sketch, nous activons dans un premier temps le relais grâce à la commande `digitalWrite()` :

```
digitalWrite(relay_pin, HIGH);
```

Il nous faut maintenant instaurer un délai de 5 s :

```
delay(5000);
```

Passé ce délai, on désactive le relais :

```
digitalWrite(relay_pin, LOW);
```

Il nous faut de nouveau insérer un délai de 5 s.

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/intro-book

Vous pouvez désormais charger le sketch sur la carte Arduino. Le relais émettra alors un clic caractéristique et vous devriez voir le voyant LED du module relais s'allumer. Après 5 s, vous devriez entendre ce même clic marquant la désactivation du relais.

Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Assurez-vous que le module relais est correctement raccordé à la carte Arduino en vous référant aux indications données dans ce chapitre.
- Enfin, assurez-vous d'utiliser la version la plus récente du code à partir du dépôt GitHub du livre.

POUR ALLER PLUS LOIN

Dans ce chapitre, nous avons vu comment connecter un module relais à une carte Arduino et le piloter en utilisant des commandes numériques.

Il vous est bien évidemment possible de mettre en pratique ce que vous avez appris dans ce chapitre afin de contrôler d'autres appareils tels qu'un éclairage à LED en suivant le même principe.

4 Utiliser la bibliothèque aREST

Dans ce chapitre, nous allons utiliser tout ce que nous avons vu jusqu'à présent et aller au-delà avec l'installation d'une nouvelle bibliothèque Arduino. Cette bibliothèque nous permettra d'avoir, à distance, une complète maîtrise de la carte Arduino sans avoir à écrire beaucoup de code. Ceci constitue la première étape vers la réalisation d'une installation domotique sans fil puisqu'il s'agit exactement du même procédé.

MATÉRIEL ET LOGICIEL REQUIS

Dressons tout d'abord une liste des composants requis pour ce projet. Les composants nécessaires sont essentiellement les mêmes que ceux utilisés pour les deux premiers chapitres.

Premièrement, munissez-vous de la carte **Arduino Uno**.

Concernant le capteur, mon choix s'est une nouvelle fois porté sur le **DHT11**. Il s'agit d'un capteur numérique utilisé pour mesurer la température et l'humidité. En matière de **relais**, j'ai opté, comme lors du chapitre précédent, pour un modèle 5 V de la marque Polulu.

Vous aurez également besoin d'une **plaque d'essai** et de **fils de raccordement** pour établir les connexions entre la carte Arduino et le capteur.

LISTE DES COMPOSANTS

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Capteur DHT11 avec résistance de 4,7 kΩ (http://snootlab.com/adafruit/255-capteur-de-temperateur-et-d-humidite-dht11-extras-.html)
- Module relais (http://www.pololu.com/product/2480)
- Plaque d'essai (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

Côté **logiciel**, vous devrez vous procurer la bibliothèque pour le capteur DHT et vous aurez également besoin de la bibliothèque aREST.

Vous pouvez télécharger les bibliothèques aux adresses suivantes :
https://github.com/adafruit/DHT-sensor-library
https://github.com/marcoschwartz/aREST

La bibliothèque **aREST** nous permettra principalement d'avoir une maîtrise complète de la carte Arduino en envoyant des commandes inspirées de l'architecture REST (*Representational State Transfer*) *via* une connexion USB entre la carte et votre ordinateur.

Le principe de l'architecture REST est de pouvoir mettre en place des interfaces de programmation (API) permettant un accès facile à l'ensemble des ressources du système, telles qu'une carte Arduino dans notre cas.

En somme, cela signifie que nous aurons la possibilité d'écrire une commande aussi simple que `/digital/8/1` afin de configurer l'état de la broche n° 8 de la carte Arduino sur HIGH.

> Pour plus d'informations sur l'architecture REST, veuillez consulter la page suivante :
> http://fr.wikipedia.org/wiki/Representational_State_Transfer

Pour installer une bibliothèque, il vous suffit d'extraire son dossier vers **/libraries**, situé dans le dossier racine Arduino.

CONFIGURER LE MATÉRIEL

Voyons comment monter ce projet. Nous devons d'abord connecter le capteur DHT, puis le relais à la carte Arduino.

Afin de connecter les différents composants à la carte Arduino, il vous suffit de vous référer au chapitre 6pour le capteur DHTet au chapitre 7 pour le module relais.

Voici le schéma complet du projet qui vous facilitera sans doute la tâche :

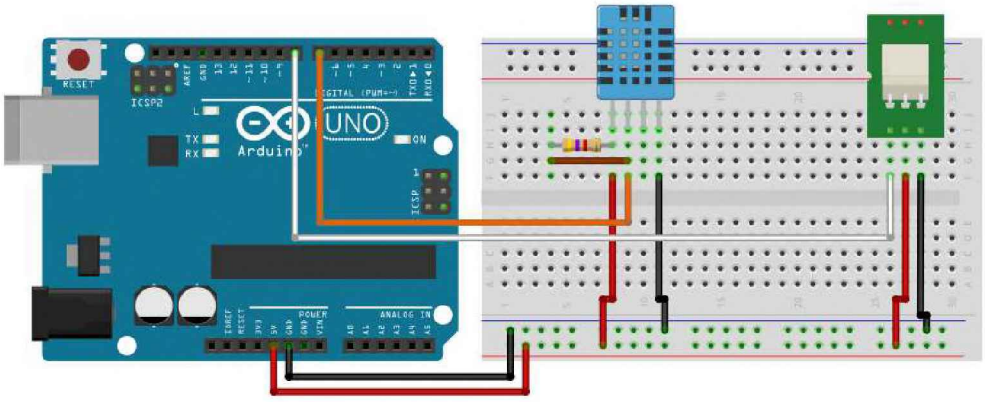
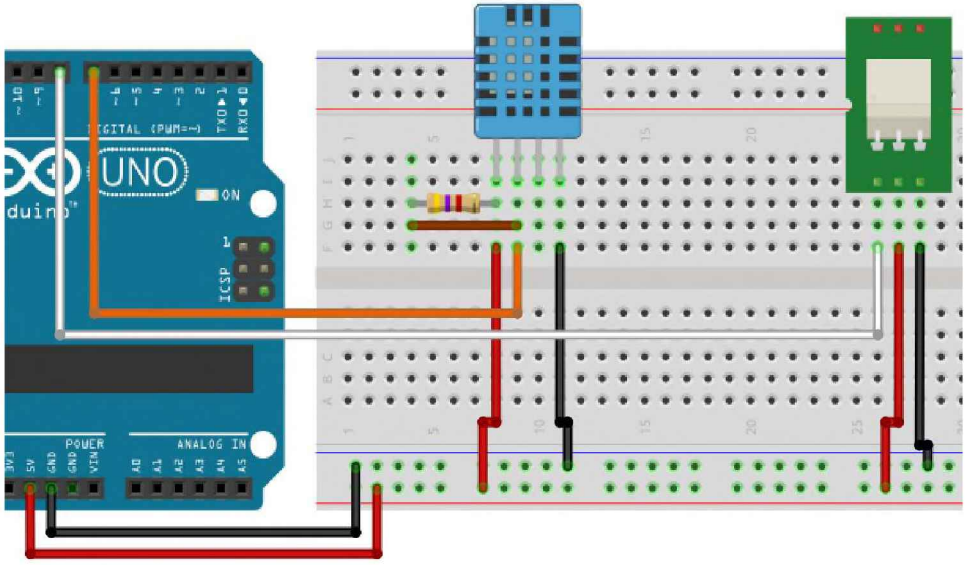
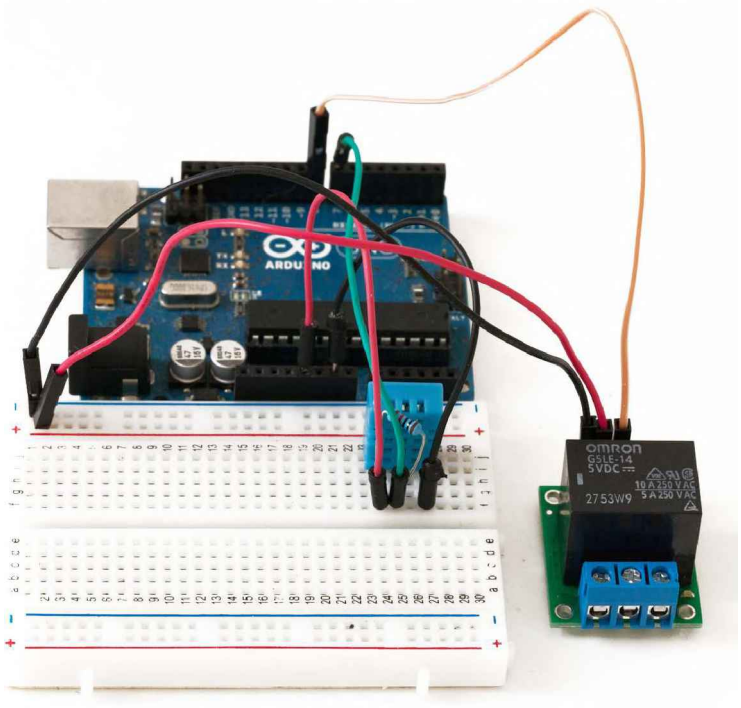


Figure 4.1 Image créée avec le logiciel Fritzing (http://fritzing.org/)



L'illustration suivante vous donne un aperçu visuel du projet.



CONTRÔLER LE PROJET

Nous allons à présent créer le sketch permettant de contrôler notre projet en utilisant la bibliothèque aREST *via* le moniteur série.

Vous trouverez ci-après le sketch complet de cette opération :

```
// A demo of the aREST library

// Libraries
#include <aREST.h>
#include "DHT.h"

// DHT sensor
#define DHTPIN 7
#define DHTTYPE DHT11

// Create aREST instance
aREST rest = aREST();

// DHT instance
DHT dht(DHTPIN, DHTTYPE);

// Variables to be exposed to the API
int temperature;
int humidity;

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Expose variables to REST API
  rest.variable("temperature",&temperature);
  rest.variable("humidity",&humidity);

  // Give name and ID to device
  rest.set_id("001");
  rest.set_name("arduino_project");

  // Start temperature sensor
  dht.begin();

}

void loop() {

  // Measure from DHT
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  temperature = (int)t;
  humidity = (int)h;
```

```
    // Handle REST calls
    rest.handle(Serial);

}
```

Voyons maintenant ce sketch en détail. Premièrement, on inclut la bibliothèque du capteur DHT et la bibliothèque aREST :

```
#include <aREST.h>
#include "DHT.h"
```

Il nous faut également définir la broche sur laquelle le capteur DHT11 est connecté :

```
#define DHTPIN 7
#define DHTTYPE DHT11
```

Pour pouvoir commander la carte à distance *via* le moniteur série, nous devons créer une instance de la bibliothèque aREST :

```
aREST rest = aREST();
```

Il est également nécessaire de créer une instance de la bibliothèque DHT :

```
DHT dht(DHTPIN, DHTTYPE);
```

Enfin, nous devons déclarer deux variables qui serviront à stocker des données du capteur :

```
int temperature;
int humidity;
```

À présent, nous devons lancer les communications série dans la fonction `setup()` :

```
Serial.begin(9600);
```

Nous devons aussi « exposer » les variables de mesure vers la bibliothèque aREST afin que celles-ci puissent être appelées en tapant des commandes dans le moniteur série. Ce qui se traduit en code par :

```
rest.variable("temperature",&temperature);
rest.variable("humidity",&humidity);
```

Toujours dans la fonction `setup()`, on nomme la carte et on lui attribue un ID (numéro d'identification) :

```
rest.set_id("001");
rest.set_name("arduino_project");
```

Il nous faut également initialiser le capteur DHT :

```
dht.begin();
```

Dans la fonction `loop()` du sketch, il faut tout d'abord récupérer les valeurs du capteur DHT, pour les convertir ensuite en nombres entiers :

```
float h = dht.readHumidity();
float t = dht.readTemperature();
temperature = (int)t;
humidity = (int)h;
```

Enfin, il nous faut gérer les requêtes provenant de l'extérieur avec :

```
rest.handle(Serial);
```

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/intro-book

Vous pouvez désormais charger le sketch sur la carte Arduino et ouvrir le moniteur série en vous assurant que la vitesse du port série est réglée à 9600. Examinons tout d'abord les valeurs du capteur DHT.

Pour afficher les valeurs de température, il vous suffit de taper :

```
/temperature
```

Vous obtiendrez alors le message suivant :

```
{"temperature": 28, "id": "001", "name": "arduino_project",
"connected": true}
```

Procédez de la même façon pour l'humidité :

```
/humidity
```

Vous recevrez alors ce message :

```
{"humidity": 35, "id": "001", "name": "arduino_project",
"connected": true}
```

Voyons maintenant comment commander le module relais. Rappelez-vous tout d'abord qu'il est nécessaire de configurer la broche n° 8 en tant que sortie, ce que nous n'avons pas fait ici dans le sketch. Rassurez-vous, c'est très facile à réaliser avec la bibliothèque aREST. Saisissez simplement :

```
/mode/8/o
```

Vous devrez alors recevoir la réponse suivante dans le moniteur série :

```
{"message": "Pin D8 set to output", "id": "001",
"name": "arduino_project", "connected": true}
```

Afin d'activer le relais, vous devez configurer l'état de la broche n° 8 sur HIGH. Pour cela, écrivez la commande suivante :

```
/digital/8/1
```

Vous devriez recevoir instantanément un message de confirmation et entendre le clic du relais.

Pour désactiver le relais, entrez :

```
/digital/8/0
```

Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Assurez-vous que vous avez téléchargé et installé correctement les bibliothèques citées dans ce chapitre.
- Assurez-vous que le module relais et le capteur DHT sont bien raccordés à la carte Arduino en vous référant aux indications données dans ce chapitre.
- Enfin, assurez-vous d'utiliser la version la plus récente du code à partir du dépôt GitHub du livre.

POUR ALLER PLUS LOIN

Vous avez appris à réaliser un projet Arduino légèrement plus complexe, en utilisant à la fois un capteur et un module relais. En outre, nous avons vu comment utiliser la bibliothèque aREST. Celle-ci fera l'objet de multiples utilisations dans le cadre de projets de domotique, étant donné que les mêmes commandes peuvent être utilisées en WiFi ou en Bluetooth.

Les chapitres suivants vous permettront notamment de commencer à monter des projets de domotique à partir des connaissances acquises jusqu'ici.

Partie II

Concevoir des installations autonomes

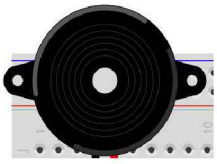
5 Installer un système d'alarme simple

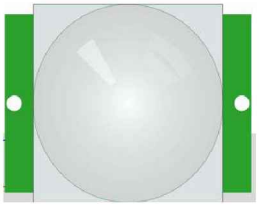
MATÉRIEL REQUIS

Nous allons maintenant créer notre tout premier projet de domotique qui est un système d'alarme simple.

Nous allons connecter un détecteur de mouvement PIR à la carte Arduino.

Nous ferons en sorte qu'une LED clignote et qu'un son soit émis par un buzzer piézoélectrique à chaque fois qu'un mouvement est détecté.





Ce projet simple à réaliser vous donnera les bases de la domotique avec Arduino.

LISTE DES COMPOSANTS

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Capteur de mouvement PIR (http://snootlab.com/adafruit/285-capteur-de-presence-pir.html)
- LED (http://pixelohm.fr/interface/90-led-rouge-verte-ou-jaune-5mm.html)
- Résistance de 330 Ω (http://composants.e44.com/resistances/resistances-a-couche-carbone/resistances-1-w/resistance-carbone-1w-330-ohms-tolerance-5-RF1330.html)
- Buzzer piézoélectrique (http://shop.mchobby.be/senseur-divers/57-piezo.html)
- Plaque d'essai (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

ASSEMBLER LE MATÉRIEL

Le schéma suivant synthétise les connexions matérielles :

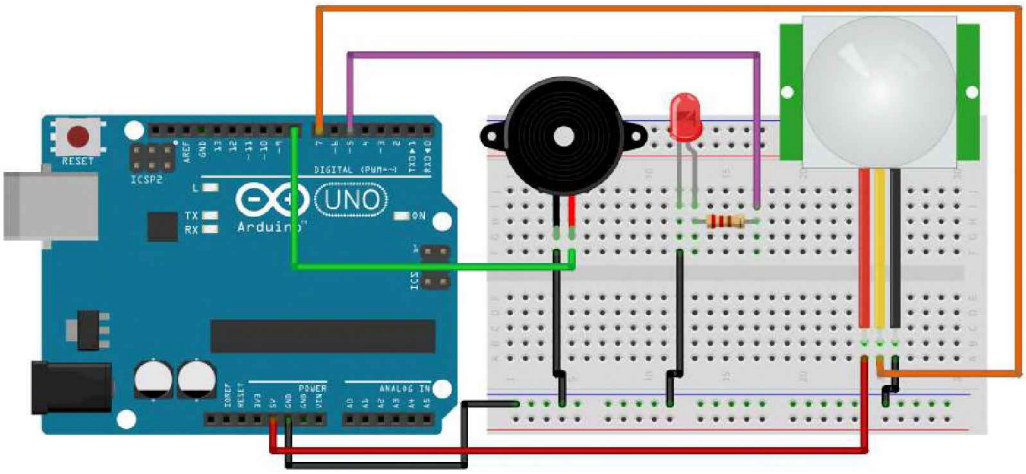


Figure 5.1 Image créée avec le logiciel Fritzing (http://fritzing.org/)

Placez tout d'abord l'ensemble des composants sur la plaque d'essai.

Disposez ensuite la plaque d'essai à proximité de la carte Arduino.

Après cela, connectez le détecteur de mouvement PIR à la plaque d'essai.

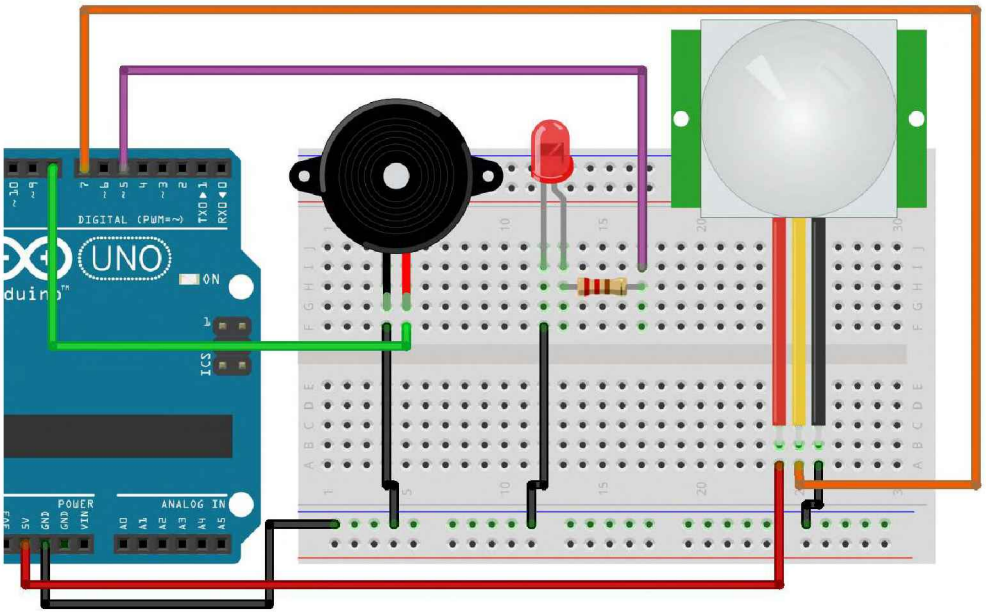
Connectez la broche GND de la carte Arduino à la rangée bleue de la plaque d'essai car il nous faudra connecter tous les appareils à la même masse.

Connectez la résistance en série à l'anode de la LED sur la plaque d'essai (l'anode est la patte la plus longue de la LED).

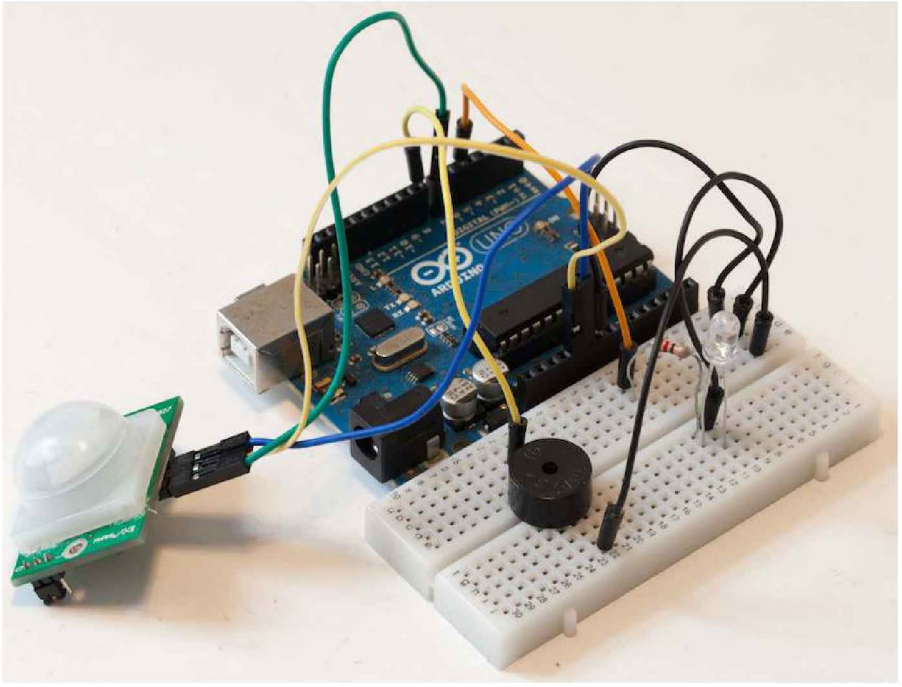
Connectez ensuite l'autre patte de la résistance à la broche n° 5 de la carte Arduino et la cathode de la LED à la masse de la carte Arduino.

En ce qui concerne le détecteur de mouvement PIR, connectez la broche GND à la masse Arduino, la broche VCC à la broche Arduino 5 V et la broche SIG à la broche Arduino n° 7.

Nous nous intéressons maintenant au buzzer piézoélectrique. Connectez le pôle positif (marqué d'un +) à la broche n° 8 et l'autre pôle à la masse Arduino.



Voici un aperçu du projet entièrement assemblé :



CONFIGURER LE MATÉRIEL

Le matériel étant désormais assemblé, nous pouvons commencer l'écriture du sketch Arduino de notre système d'alarme.

Vous trouverez ci-dessous la version complète du code de cette partie :

```
// Pins
constint alarm_pin = 8;
constint led_pin = 5;
constint motion_pin = 7;

// Alarm
boolean alarm_mode = false;

// Variables for the flashing LDED
int ledState = LOW;
long previousMillis = 0;
long interval = 100;
// Interval at which to blink (milliseconds)

void setup()
{
// Set pins to output
  pinMode(led_pin,OUTPUT);
  pinMode(alarm_pin,OUTPUT);

// Wait before starting the alarm
  delay(5000);
}

void loop()
{
// Motion detected ?
if (digitalRead(motion_pin)) {
    alarm_mode = true;
  }

// If alarm mode is on, flash the LED and make the alarm ring
if (alarm_mode){
unsignedlong currentMillis = millis();
if(currentMillis - previousMillis > interval) {
      previousMillis = currentMillis;
if (ledState == LOW)
        ledState = HIGH;
else
        ledState = LOW;
// Switch the LED
    digitalWrite(led_pin, ledState);
    }
    tone(alarm_pin,1000);
  }
}
```

Étudions de plus près ce code. La première étape consiste à déclarer les broches auxquelles les différents composants sont connectés :

```
constint alarm_pin = 8;
constint led_pin = 5;
constint motion_pin = 7;
```

Il nous faut maintenant mémoriser l'état de l'alarme (activée ou non) au sein d'une variable :

```
boolean alarm_mode = false;
```

Il faut également créer une variable afin de faire clignoter la LED lorsque l'alarme est activée :

```
int ledState = LOW;
long previousMillis = 0;
long interval = 100;  // Interval at which to blink (milliseconds)
```

À présent, nous allons définir les broches et le buzzer piézoélectrique en tant que sorties au sein de la fonction `setup()` du sketch :

```
pinMode(led_pin,OUTPUT);
pinMode(alarm_pin,OUTPUT);
```

Instaurons un délai de 5 secondes dans le code afin d'éviter que l'alarme ne s'active dans la foulée :

```
delay(5000);
```

Dans la fonction `loop()` du sketch, il nous faut vérifier en permanence l'état du détecteur de mouvement PIR. La variable de l'alarme doit être définie sur « true » à chaque fois qu'un mouvement est détecté.

```
if (digitalRead(motion_pin)) {
  alarm_mode = true;
}
```

Lorsque l'alarme est activée, deux opérations sont nécessaires : faire clignoter la LED en continu et déclencher une sonnerie à partir du buzzer piézoélectrique. Pour cela, écrivez le code suivant :

```
if (alarm_mode){
unsignedlong currentMillis = millis();
if(currentMillis - previousMillis > interval) {
    previousMillis = currentMillis;
if (ledState == LOW)
      ledState = HIGH;
else
      ledState = LOW;

// Switch the LED
    digitalWrite(led_pin, ledState);
    }
```

```
        tone(alarm_pin,1000);
    }
```

Vous pouvez retrouver l'ensemble des codes de ce premier projet dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino/

TESTER LE SYSTÈME D'ALARME

Vous pouvez désormais tester ce premier projet.

Chargez le code sur la carte Arduino *via* le logiciel Arduino IDE.

Une fois le délai initial de 5 secondes passé, effectuez un mouvement devant le capteur.

Vous devriez entendre l'alarme s'activer et observer le clignotement permanent de la LED.

Pour la désactiver, appuyez simplement sur le bouton rouge « Reset » de la carte Arduino.

Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Premièrement, assurez-vous que toutes les connexions matérielles ont été correctement effectuées en vous reportant à la partie consacrée à la configuration.
- Vérifiez également que vous avez utilisé la version du code la plus récente à partir du dépôt GitHub du livre.

POUR ALLER PLUS LOIN

J'espère que ce projet assez abordable vous a d'ores et déjà donné une idée de ce que vous pouvez faire en domotique avec Arduino.

Dans le chapitre suivant, nous allons nous servir de la plateforme Arduino pour réaliser des projets de domotique encore plus intéressants.

6 Mesurer température, humidité et lumière

Dans ce projet, nous allons voir comment obtenir un suivi de la température, de l'humidité et du niveau d'éclairage d'une pièce à l'aide d'Arduino, d'un capteur d'humidité et de température, d'une photorésistance et d'un écran LCD. L'ensemble des données apparaîtra sur l'écran LCD.

Ce projet peut servir d'élément de base à une installation plus complexe pour contrôler à distance les données liées à votre domicile.

MATÉRIEL ET LOGICIEL REQUIS

Pour ce projet, il vous faudra bien entendu une carte **Arduino Uno**. Sa réalisation est également possible avec une carte **Arduino Mega** ou **Leonardo**.

Afin de prendre des mesures de température et d'humidité, vous aurez besoin d'un **capteur DHT11** accompagné d'une **résistance de 4,7 kΩ**. L'utilisation d'un capteur DHT22 est possible, vous devrez cependant modifier une ligne de code.

Pour mesurer le niveau d'éclairage, j'ai choisi une photorésistance dotée d'une **résistance de 10 kΩ**. On obtiendra en retour un signal proportionnel au niveau de lumière entrante.

Il est nécessaire de s'équiper d'un **écran LCD** afin d'afficher les mesures. J'ai ainsi opté pour un écran LCD 4 lignes-20 caractères qui permet d'afficher simultanément quatre mesures. Vous pouvez utiliser un écran d'une plus petite taille mais vous ne serez pas en mesure de voir en même temps les données de température et d'humidité par exemple.

L'écran choisi dispose d'une **interface I2C** permettant d'interagir avec la carte Arduino. Je recommande fortement l'utilisation d'un écran équipé de cette interface car seules deux broches de communication seront raccordées à la carte Arduino.

Pour finir, j'ai utilisé une **plaque d'essai** ainsi que des **fils de raccordement** mâle/mâle pour effectuer les différents branchements.

LISTE DES COMPOSANTS

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Capteur DHT11 avec résistance de 4,7 kΩ (http://snootlab.com/adafruit/255-capteur-de-temperateur-et-d-humidite-dht11-extras-.html)
- Photorésistance (http://snootlab.com/composants/97-photoresistance.html)
- Résistance de 10 kΩ (http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html)

☞

☞

- Afficheur LCD (http://www.es-france.com/produit2332/product_info.html)
- Plaque d'essai (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

Côté **logiciel**, vous devrez vous procurer la bibliothèque pour le capteur DHT et vous aurez également besoin de la bibliothèque LiquidCrystal pour l'afficheur LCD.

Vous pouvez télécharger les bibliothèques aux adresses suivantes :
https://github.com/adafruit/DHT-sensor-library
https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads

Pour installer une bibliothèque, il vous suffit de placer son dossier dans **/libraries**, situé dans le dossier racine Arduino.

CONFIGURER LE MATÉRIEL

L'illustration suivante synthétise les connexions matérielles :

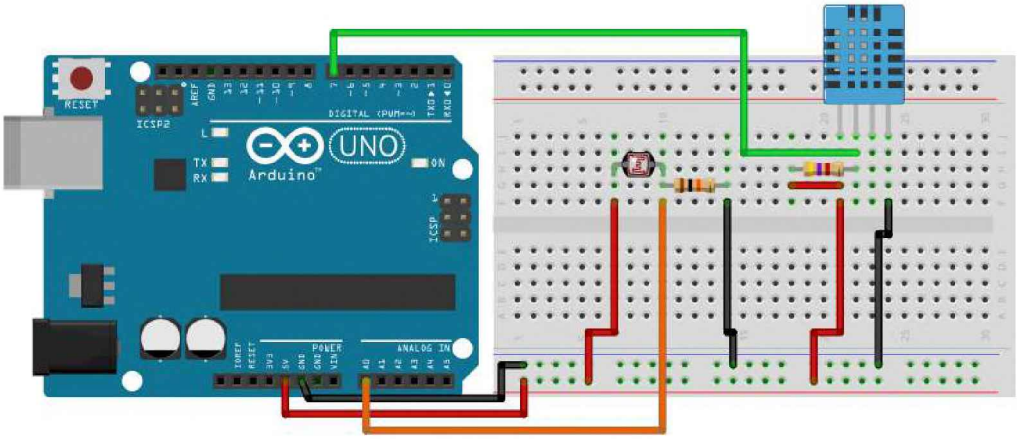
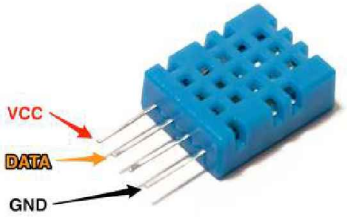


Figure 6.1 Image créée avec le logiciel Fritzing (http://fritzing.org/)

Les connexions matérielles du projet sont relativement simples : raccord du capteur DHT11, de l'afficheur LCD et de la partie mesure du niveau d'éclairage avec la photorésistance.

Connectons tout d'abord la broche 5 V de la carte Arduino Uno à la rangée rouge de la plaque d'essai et la broche de masse à la rangée bleue.

Reportez-vous à l'image suivante pour savoir quelle broche du capteur DHT11 raccorder.



Après cela, connectez la broche n° 1 du capteur DHT11 (VCC) à la rangée rouge de la plaque d'essai et la broche n° 4 (GND) à la rangée bleue.

Ensuite, connectez la broche n° 2 du capteur à la broche n° 7 de la carte Arduino.

Pour en finir avec le branchement du capteur, insérez la résistance de 4,7 kΩ entre les broches n° 1 et n° 2 du capteur.

Intéressons-nous maintenant à la photorésistance.

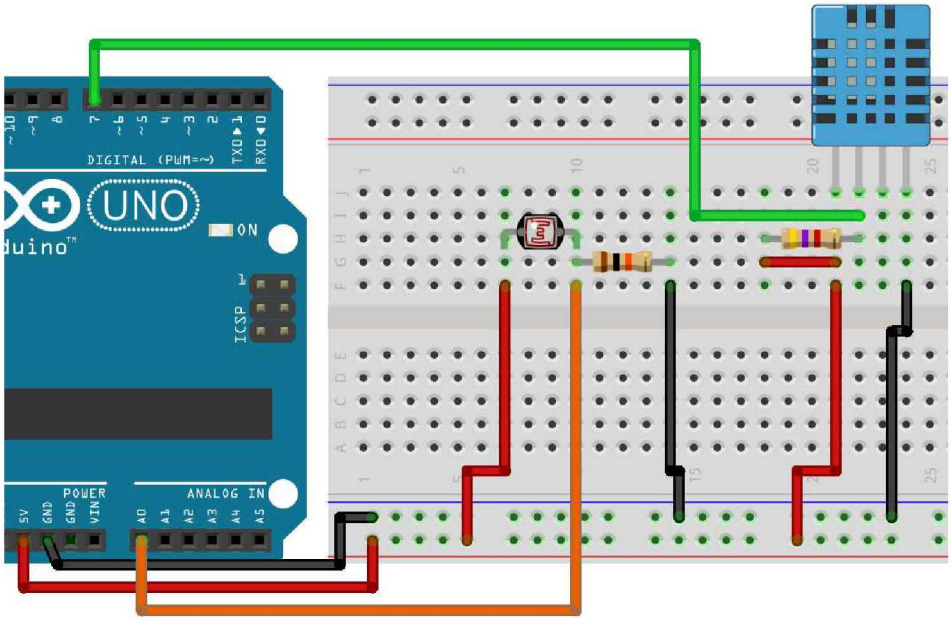
Placez-la d'abord en série avec la résistance de 10 kΩ sur la plaque d'essai.

Ensuite, connectez l'autre patte de la photorésistance à la rangée rouge de la plaque d'essai et la seconde patte de la résistance de 10 kΩ à la rangée bleue (masse).

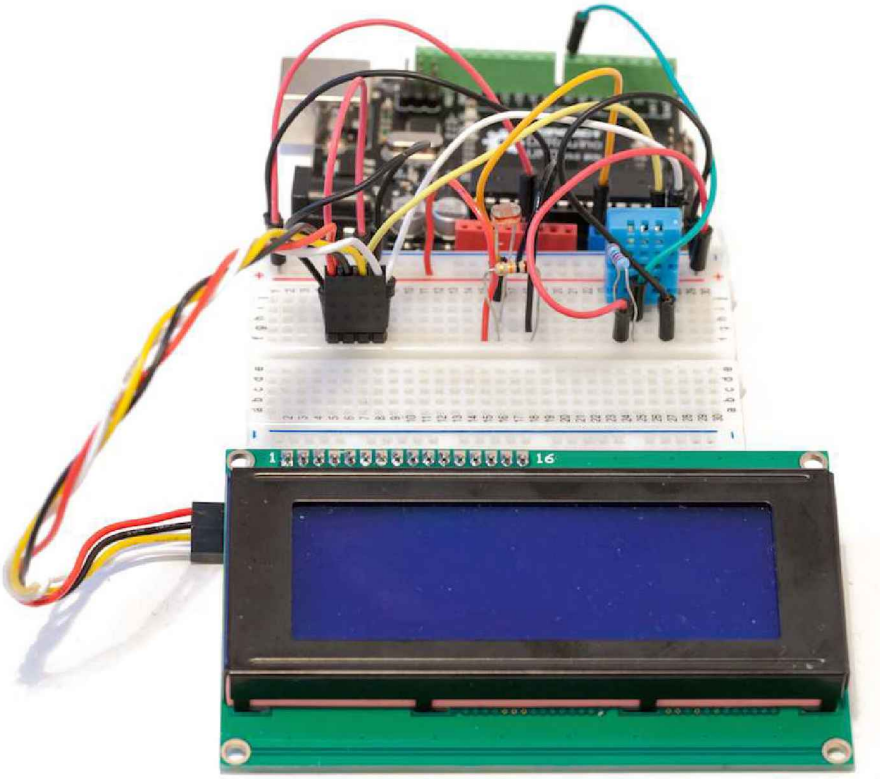
Enfin, connectez la broche commune entre la photorésistance et la résistance à l'entrée analogique A0 de la carte Arduino Uno.

Nous allons maintenant raccorder l'afficheur LCD.

Étant donné que notre afficheur dispose d'une interface I2C, seuls deux câbles devront être raccordés pour le signal et deux autres pour l'alimentation. Connectons tout d'abord la broche VCC du LCD à la rangée rouge de la plaque d'essai et la broche GND à la rangée bleue ; puis la broche SDA de l'afficheur à l'entrée analogique A4 de la carte Arduino et enfin la broche SCL à l'entrée A5.



L'illustration suivante vous donne un aperçu visuel du projet entièrement assemblé :



TESTER LES CAPTEURS

Une fois le matériel du projet complètement assemblé, il nous faut procéder au test des différents capteurs de la carte.

Pour ce faire, nous allons simplement écrire un sketch Arduino. Il s'agira de lire les données des capteurs et de les faire apparaître sur le port série.

Vous trouverez ci-dessous la version complète du code de cette partie :

```
// Libraries
#include "DHT.h"

// DHT sensor
#define DHTPIN 7
#define DHTTYPE DHT11

// DHT instance
DHT dht(DHTPIN, DHTTYPE);

void setup()
{
```

```
// Initialize the Serial port
  Serial.begin(9600);

// Init DHT
  dht.begin();
}

void loop()
{
// Measure from DHT
float temperature = dht.readTemperature();
float humidity = dht.readHumidity();

// Measure light level
float sensor_reading = analogRead(A0);
float light = sensor_reading/1024*100;

// Display temperature
  Serial.print("Temperature: ");
  Serial.print((int)temperature);
  Serial.println(" C");

// Display humidity
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println("%");

// Display light level
  Serial.print("Light: ");
  Serial.print(light);
  Serial.println("%");
  Serial.println("");

// Wait 500 ms
  delay(500);

}
```

Premièrement, importons la bibliothèque pour le capteur DHT :

```
#include "DHT.h"
```

Créons une instance pour le capteur DHT :

```
DHT dht(DHTPIN, DHTTYPE);
```

Nous allons initialiser le capteur dans la fonction `setup()` du sketch :

```
dht.begin();
```

Faisons de même avec le port série :

```
Serial.begin(9600);
```

Dans la fonction `loop()`, nous allons lire les données des capteurs en continu et les faire apparaître dans le port série.

Commençons par les données de température et d'humidité :

```
float temperature = dht.readTemperature();
float humidity = dht.readHumidity();
```

En ce qui concerne la photorésistance, nous allons lire en premier lieu les données provenant de l'entrée analogique A0, soit une valeur comprise entre 0 et 1023 étant donné que le convertisseur analogique-numérique de la carte Arduino Uno possède une résolution de 10 bits, c'est-à-dire 1 024 niveaux.

Nous diviserons ensuite cette valeur par 1 024 et multiplierons le résultat par 100 pour obtenir un niveau d'éclairage en pourcentage :

```
float sensor_reading = analogRead(A0);
float light = sensor_reading/1024*100;
```

Une fois cette étape réalisée, il nous faut inscrire ces données sur le port série. Commençons par la température :

```
Serial.print("Temperature: ");
Serial.print((int)temperature);
Serial.println("C");
```

Pour inscrire les valeurs relatives à l'humidité, nous procédons de la même manière que pour le niveau d'éclairage :

```
Serial.print("Light: ");
Serial.print(light);
Serial.println("%");
```

Il ne nous reste plus qu'à ajouter un délai de 500 ms entre chaque série de mesures :

```
delay(500);
```

Vous pouvez retrouver le code complet du chapitre dans le dossier correspondant du dépôt GitHub du livre :
https://github.com/openhomeautomation/home-automation-arduino

Il est temps à présent de tester ce premier sketch Arduino.

Chargez le code sur la carte Arduino et lancez le moniteur série du logiciel IDE (en vous assurant que la vitesse du port série est réglée à 9600).

Vous devriez voir apparaître ceci :

```
Temperature: 25 C
Humidity: 36.00%
Light: 83,79%
```

Si vous obtenez ce résultat, c'est que vos capteurs fonctionnement correctement, bravo ! Tentez par exemple de passer votre main devant la photorésistance. Vous devriez observer une variation soudaine du niveau d'éclairage.

Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Assurez-vous d'abord que les capteurs et l'afficheur LCD sont correctement raccordés à la carte Arduino.
- De même, vérifiez que les bibliothèques du capteur DHT et de l'afficheur LCD aient bien été installées.

AFFICHER LES DONNÉES SUR L'ÉCRAN LCD

Nous allons maintenant procéder à l'assemblage du projet en nous servant de ce que nous avons déjà fait jusqu'ici. Par conséquent, nous allons conserver la partie du sketch consacrée aux mesures et afficher les résultats sur l'écran LCD.

Comme le code est dans son ensemble semblable au sketch précédemment écrit, je détaillerai uniquement les parties qui concernent l'affichage sur le LCD. Vous pouvez bien entendu retrouver le code complet dans le dépôt **GitHub** du livre.

Voici la version complète du code de cette partie :

```
// Libraries
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "DHT.h"

// DHT sensor
#define DHTPIN 7
#define DHTTYPE DHT11

// LCD display instance
LiquidCrystal_I2C lcd(0x27,20,4);

// DHT instance
DHT dht(DHTPIN, DHTTYPE);

void setup()
{
// Initialize the lcd
  lcd.init();

// Print a message to the LCD
lcd.backlight();
  lcd.setCursor(1,0);
  lcd.print("Hello !");
  lcd.setCursor(1,1);
  lcd.print("Initializing ...");

// Init DHT
  dht.begin();

// Clear LCD
  delay(2000);
  lcd.clear();
}
```

```
void loop()
{
// Measure from DHT
float temperature = dht.readTemperature();
float humidity = dht.readHumidity();

// Measure light level
float sensor_reading = analogRead(A0);
float light = sensor_reading/1024*100;

// Display temperature
  lcd.setCursor(1,0);
  lcd.print("Temperature: ");
  lcd.print((int)temperature);
  lcd.print((char)223);
  lcd.print("C");

// Display humidity
  lcd.setCursor(1,1);
  lcd.print("Humidity: ");
  lcd.print(humidity);
  lcd.print("%");

// Display light level
  lcd.setCursor(1,2);
  lcd.print("Light: ");
  lcd.print(light);
  lcd.print("%");

// Wait 100 ms
  delay(100);

}
```

Il nous faut premièrement ajouter les bibliothèques de l'afficheur LCD et du capteur DHT :

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "DHT.h"
```

Nous pouvons ensuite créer une instance de l'afficheur LCD.

Dans le cas où vous utiliseriez une taille d'écran différente de celle préconisée (affichage à deux lignes par exemple), c'est maintenant qu'il faut le préciser :

```
LiquidCrystal_I2C lcd(0x27,20,4);
```

Nous devons initialiser l'afficheur LCD dans la fonction `setup()` du sketch :

```
lcd.init();
```

Toujours dans cette fonction, activez le rétroéclairage de l'écran LCD et saisissez un message de bienvenue :

```
lcd.backlight();
lcd.setCursor(1,0);
lcd.print("Hello !");
lcd.setCursor(1,1);
lcd.print("Initializing ...");
```

Après un délai de deux secondes, nous allons simplement effacer ce qui est affiché à l'écran avant de prendre des mesures :

```
delay(2000);
lcd.clear();
```

Après avoir relevé ces mesures, nous faisons apparaître la température sur la première ligne de l'écran dans la fonction `loop()` :

```
lcd.setCursor(1,0);
lcd.print("Temperature: ");
lcd.print((int)temperature);
lcd.print((char)223);
lcd.print("C");
```

Ensuite, nous faisons apparaître les valeurs relatives à l'humidité sur la seconde ligne :

```
lcd.setCursor(1,1);
lcd.print("Humidity: ");
lcd.print(humidity);
lcd.print("%");
```

Si vous disposez, comme moi, de quatre lignes sur votre écran, effectuez la même opération sur la troisième ligne pour le niveau d'éclairage :

```
lcd.setCursor(1,2);
lcd.print("Light: ");
lcd.print(light);
lcd.print("%");
```

Si vous utilisez un écran LCD à deux lignes, plusieurs options s'offrent à vous.

Ajoutez par exemple un délai, effacez ce qui apparaît à l'écran et inscrivez le niveau d'éclairage sur la première ligne.

Ajoutez aussi un délai de 100 ms entre chaque série de mesures et réactualisation de l'écran LCD :

```
delay(100);
```

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

Il est temps de tester le projet.
Chargez une nouvelle fois le code sur la carte Arduino et patientez un court instant.
Votre message de bienvenue devrait apparaître à l'écran, suivi des mesures.
Voici un aperçu du projet en action :

Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Vérifiez que le code visant à tester les différents capteurs fonctionne correctement. Pour cela, n'hésitez pas à revenir en arrière si nécessaire.
- Assurez-vous également que votre écran LCD est bien branché.
- Vérifiez enfin que la bibliothèque LCD que vous utilisez est compatible avec votre écran.

POUR ALLER PLUS LOIN

Dans ce chapitre, nous avons connecté différents capteurs à Arduino, en l'occurrence un capteur d'humidité et un capteur de température. Nous avons par la suite fait apparaître les valeurs fournies sur l'écran LCD, également contrôlé par la carte Arduino.

Le contenu de ce chapitre peut servir à la réalisation de projets plus intéressants encore. Il vous est possible d'inclure un nombre plus important de capteurs et de faire apparaître leurs valeurs à l'écran. Vous pouvez par exemple introduire un capteur de pression barométrique dans votre projet. Vous pouvez aussi conserver les mêmes capteurs et vous munir d'un écran OLED qui affichera les mesures sous forme de graphiques.

7 Construire une lampe intelligente

Dans le cadre de ce projet, nous allons réaliser une installation domotique assez classique autour de la création d'une lampe dite « intelligente », c'est-à-dire capable de s'allumer automatiquement en cas de faible éclairage et inversement de s'éteindre si le niveau d'éclairage est suffisant.

Pour ce faire, nous utiliserons un **module relais** pour commander la lampe ainsi qu'une **photorésistance** chargée de mesurer le niveau d'éclairage ambiant. L'utilisation de la plateforme Arduino nous permettra d'apporter d'autres fonctionnalités au projet.

Nous commençons par intégrer un **capteur de courant** au projet. Il sera ainsi possible de déterminer la quantité de courant qui traverse la lampe et sa consommation à un moment précis. Pour ce projet, nous utiliserons aussi un afficheur LCD pour visualiser l'état du relais, la consommation énergétique de la lampe et la valeur de l'éclairage ambiant. En ce qui concerne la lampe en elle-même, nous prendrons un modèle de bureau de base sachant que n'importe quelle autre lampe pourrait faire l'affaire.

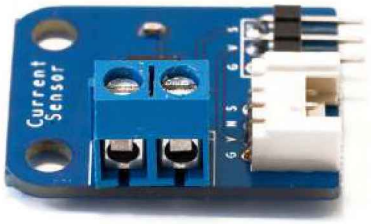
MATÉRIEL ET LOGICIEL REQUIS

Pour ce projet, il vous faudra bien entendu une carte Arduino Uno ou une version similaire.

J'ai utilisé un **module relais** 5 V de chez Polulu (voir ci-dessous) pourvu d'un relais placé sur une carte et de tous les composants nécessaires à la connexion Arduino.

Pour mesurer le courant qui circule à travers la lampe, j'ai utilisé une carte comportant un **capteur AC712** du fabricant ITead Studio. Ce capteur s'utilise très facilement avec Arduino puisqu'il renvoie une tension proportionnelle au courant mesuré. Grâce à une formule bien précise, nous pourrons estimer l'intensité du courant qui traverse la lampe en se basant sur la tension mesurée par la carte Arduino. Vous pouvez bien entendu vous servir d'une carte différente à partir du moment où elle possède le même capteur.

Voici un aperçu de la carte dont je me suis servi pour mesurer le courant :

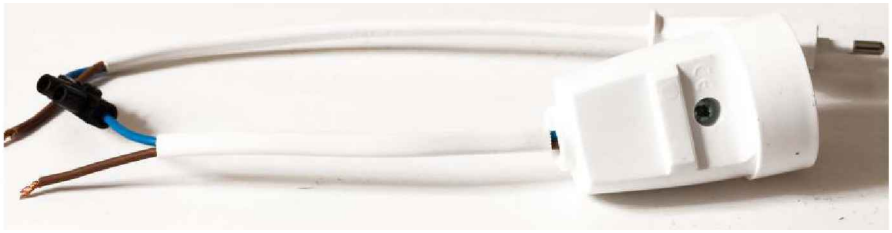


Pour mesurer le niveau d'éclairage, j'ai choisi une **photorésistance** dotée d'une résistance de 10 kΩ qui renverra un signal proportionnel à la quantité de lumière entrante.

Un **afficheur LCD** nous est indispensable pour afficher l'état du relais, la consommation énergétique de l'appareil ainsi que le niveau d'éclairage. J'ai opté pour un écran LCD 4 lignes-20 caractères ce qui me permet d'afficher simultanément quatre mesures. Il vous est possible d'utiliser un écran d'une plus petite taille mais vous ne pourrez visualiser que deux mesures en même temps, l'état du relais et la consommation énergétique par exemple.

L'écran LCD que j'ai choisi dispose d'une **interface I2C** permettant d'interagir avec la carte Arduino. Je recommande fortement d'utiliser un écran équipé de cette interface car seules deux broches de communication seront raccordées à la carte Arduino.

Pour connecter la lampe au projet, je me suis servi de **deux fiches secteur** munies de simples câbles et comprenant une prise femelle (où la lampe viendra se brancher) et une prise mâle (à brancher sur la prise murale). Voilà les câbles dont je me suis servi :



La manipulation de ces câbles nécessite de prendre certaines précautions puisqu'il s'agit là d'un projet soumis à de hautes tensions.

Pour finir, j'ai utilisé une **plaque d'essai** ainsi que des fils de raccordement pour effectuer les différents branchements.

LISTE DES COMPOSANTS

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Module relais (http://snootlab.com/tinker-it/229-mod.html)
- Capteur de courant (http://www.robotshop.com/eu/fr/capteur-courant-5a-acs712-electronic-brick.html)
- Photorésistance (http://snootlab.com/composants/97-photoresistance.html)

☞

☞

- Résistance de 10 kΩ (http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html)
- Afficheur LCD (http://www.es-france.com/produit2332/product_info.html)
- Plaque d'essai (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

La lampe que j'ai utilisée est un modèle standard (30 W). Néanmoins, le module relais choisi peut supporter une puissance allant jusqu'à 1 200 W. Vous pouvez donc y brancher une lampe ou un appareil plus puissant.

Côté logiciel, Arduino IDE et la bibliothèque LiquidCrystal pour l'afficheur LCD vous suffiront pour réaliser ce projet.

Vous pouvez télécharger bibliothèque LiquidCrystal à cette adresse :
https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads

Pour installer une bibliothèque, il vous suffit de placer son dossier dans **/libraries**, situé dans le dossier racine Arduino.

CONFIGURER LE MATÉRIEL

Assemblons maintenant le matériel de notre projet. Nous allons le faire en deux fois. Il nous faut d'abord connecter les différents composants comme le module relais à la carte Arduino, avant d'incorporer la lampe à notre montage.

Les **connexions matérielles** sont assez simples : connectez le module relais, le capteur de courant et la photorésistance.

Connectons tout d'abord la broche 5 V de la carte Arduino Uno à la rangée rouge de la plaque d'essai et la broche de masse à la rangée bleue. Connectez ensuite la photorésistance en série avec la résistance de 10 kΩ sur la plaque d'essai. L'autre patte de la photorésistance sera connectée à la rangée rouge de la plaque d'essai et la seconde patte de la résistance à la rangée bleue (masse). Enfin, connectez la broche commune entre la photorésistance et la résistance à l'entrée analogique A0 de la carte Arduino.

Pour le module relais, vous aurez trois broches à connecter : VCC, GND et SIG, la broche désignée habituellement comme **broche de signal**. Raccordez la broche VCC à la rangée rouge de la plaque d'essai, puisqu'elle doit être connectée à la broche 5 V de la carte Arduino. Raccordez la broche GND à la rangée bleue de la plaque d'essai, puisqu'elle doit être connectée à la broche de masse de la carte Arduino.

Pour finir, connectez la broche SIG à la broche n° 8 de la carte Arduino.

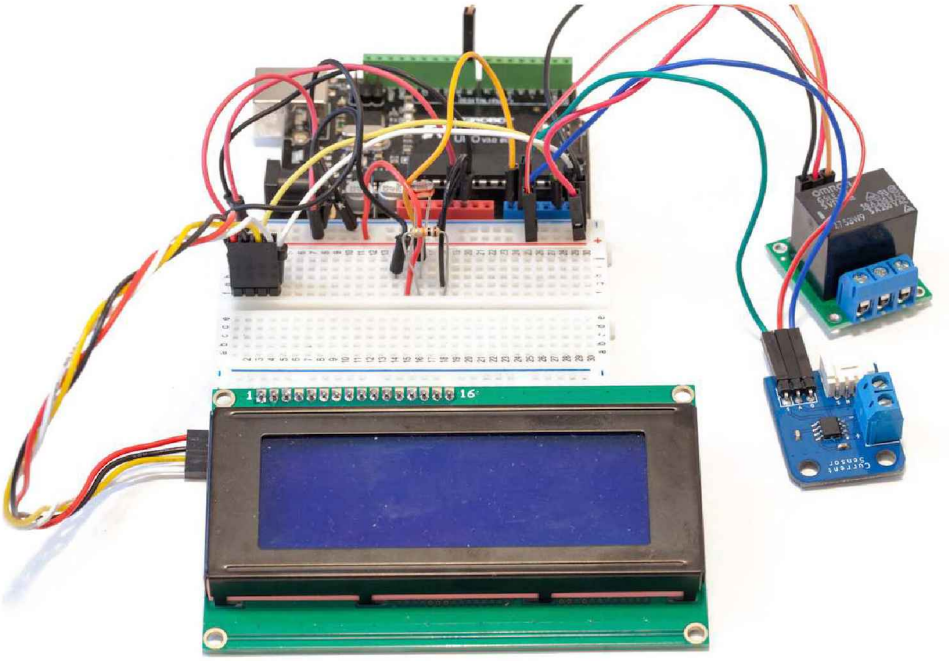
Nous devons procéder de la même façon pour connecter le capteur de courant. Celui-ci dispose de trois broches : VCC, GND et OUT.

Comme pour le relais, la broche VCC doit être connectée à la broche 5 V de la carte Arduino. Il vous faut donc la relier à la rangée rouge de la plaque d'essai. Raccordez la broche GND à la rangée bleue de la plaque d'essai, car elle doit être connectée à la broche de masse de la carte Arduino. Pour finir, connectez la broche OUT à l'entrée analogique A1 de la carte Arduino.

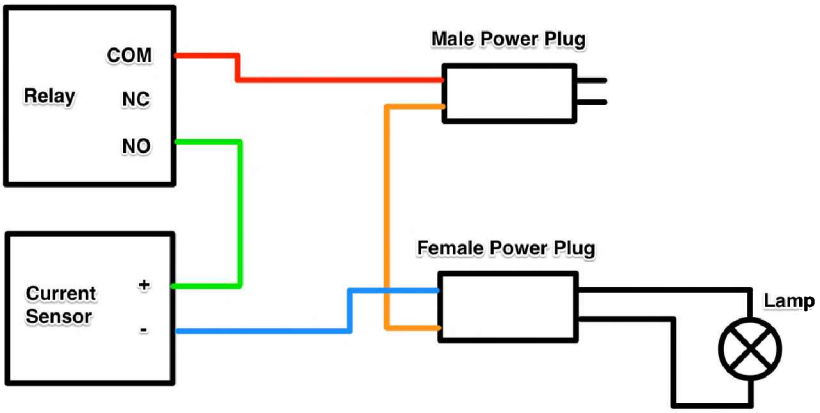
Nous allons maintenant raccorder l'afficheur LCD.

Étant donné que notre afficheur dispose d'une interface I2C, seuls deux câbles devront être raccordés pour le signal et deux autres pour l'alimentation. Raccordons tout d'abord la broche VCC du LCD à la rangée rouge de la plaque d'essai et la broche GND à la rangée bleue. Ensuite, connectons la broche SDA de l'afficheur LCD à l'entrée analogique A4 de la carte Arduino et la broche SCL à l'entrée A5.

Voici un aperçu du projet entièrement assemblé, mais sans la lampe :



Ajoutons-la maintenant au projet. L'idée est de faire passer l'alimentation électrique principale (de la prise murale) par le relais, puis par le capteur de courant pour atteindre enfin la lampe. Le schéma suivant vous aidera à effectuer les branchements :



Puisque des tensions dangereuses sont impliquées (110 V ou 230 V suivant le continent où vous résidez), certaines précautions sont à prendre. Pour les connaître, veuillez vous reporter au chapitre 1.

Notez qu'il est possible de tester ce projet sans brancher le moindre appareil au relais et au capteur de courant.

Vérifions maintenant si le projet fonctionne.

TESTER LE RELAIS

Nous allons tester l'élément principal du montage : le relais qui commande la lampe.

Pour ce faire, activons et désactivons le relais par intervalles de 5 secondes. Nous verrons ainsi si le relais fonctionne et si les branchements avec la lampe ont été correctement réalisés.

Voici la version complète du code de cette partie :

```
// Relay pin
constint relay_pin = 8;

void setup() {
  pinMode(relay_pin,OUTPUT);
}

void loop() {

// Activate relay
  digitalWrite(relay_pin, HIGH);

// Wait for 5 seconds
  delay(5000);

// Deactivate relay
  digitalWrite(relay_pin, LOW);

// Wait for 5 seconds
  delay(5000);
}
```

En premier lieu, on indique à quelle broche le relais est connecté :

```
constint relay_pin = 8;
```

Nous devons définir cette broche en tant que sortie dans la fonction `setup()` du sketch :

```
pinMode(relay_pin,OUTPUT);
```

Ensuite, on définit l'état de cette broche sur HIGH dans la fonction `loop()`, ce qui activera le relais :

```
digitalWrite(relay_pin, HIGH);
```

Insérons maintenant un délai de 5 secondes :

```
delay(5000);
```

On désactive une nouvelle fois le relais :

```
digitalWrite(relay_pin, LOW);
```

On insère une nouvelle fois un délai de 5 secondes avant de répéter la fonction `loop()` :

```
delay(5000);
```

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

Il est temps à présent de tester le sketch.

Assurez-vous que la lampe est correctement connectée au projet et que la fiche mâle est branchée à la prise murale.

Après cela, chargez le sketch Arduino sur la carte. Vous devriez voir le relais s'activer et se désactiver à 5 secondes d'intervalle, allumant et éteignant la lampe.

Veillez à fixer le relais de façon à ce qu'il ne puisse être touché par accident.

MESURER LA PUISSANCE ET LA COMMANDE AUTOMATIQUE DE L'ÉCLAIRAGE

Abordons à présent la partie majeure du projet : l'écriture du sketch Arduino pour notre lampe intelligente. Il nous faut relever en continu le niveau d'éclairage et la consommation énergétique de la lampe, faire apparaître ces données sur l'afficheur LCD et modifier l'état du relais en conséquence.

Voici la version complète du code de cette partie :

```
// Libraries
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Relay state
constint relay_pin = 8;
boolean relay_state = false;

// LCD display instance
LiquidCrystal_I2C lcd(0x27,20,4);

// Define measurement variables
float amplitude_current;
float effective_value;
float effective_voltage = 230; // Set voltage to 230V (Europe)
                               // or 110V (US)
float effective_power;
```

```
float zero_sensor;

void setup()
{
// Initialize the lcd
  lcd.init();

// Print a message to the LCD.
lcd.backlight();
  lcd.setCursor(1,0);
  lcd.print("Hello !");
  lcd.setCursor(1,1);
  lcd.print("Initializing ...");

// Set relay pin to output
  pinMode(relay_pin,OUTPUT);

// Calibrate sensor with null current
  zero_sensor = getSensorValue(A1);

// Clear LCD
  delay(2000);
  lcd.clear();
}

void loop()
{
// Measure light level
float sensor_reading = analogRead(A0);
float light = (sensor_reading/1024*100);

// Perform power measurement
float sensor_value = getSensorValue(A1);

// Convert to current
  amplitude_current =
            (float)(sensor_value-zero_sensor)/1024*5/185*1000000;
  effective_value = amplitude_current/1.414;
  effective_power = abs(effective_value*effective_voltage/1000);

// Switch relay accordingly
// If the light level is more than 75 %, switch the lights off
if (light >75) {
    digitalWrite(relay_pin, LOW);
    relay_state = false;
  }
// If the light level is less than 50 %, switch the lights off
if (light <50) {
    digitalWrite(relay_pin, HIGH);
    relay_state = true;
  }
```

```
// Update LCD screen

// Display relay state
  lcd.setCursor(1,0);
  lcd.print("Relay: ");
if (relay_state) {lcd.print("On ");}
else {lcd.print("Off");}

// Display energy consumption
  lcd.setCursor(1,1);
  lcd.print("Power: ");
  lcd.print(effective_power);
  lcd.print("W");

// Display light level
  lcd.setCursor(1,2);
  lcd.print("Light: ");
  lcd.print(light);
  lcd.print("%");

// Wait 500 ms
  delay(500);

}

// Get the reading from the current sensor
float getSensorValue(int pin)
{
int sensorValue;
float avgSensor = 0;
int nb_measurements = 100;
for (int i = 0; i < nb_measurements; i++) {
    sensorValue = analogRead(pin);
    avgSensor = avgSensor + float(sensorValue);
  }
  avgSensor = avgSensor/float(nb_measurements);
return avgSensor;
}
```

Premièrement, on inclut les bibliothèques nécessaires au capteur DHT :

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

Nous pouvons maintenant créer une instance de l'afficheur LCD. Suivant l'écran que vous utilisez, vous devrez peut-être changer le nombre de lignes (quatre lignes pour le mien) :

```
LiquidCrystal_I2C lcd(0x27,20,4);
```

On déclare la broche à laquelle le relais est relié ainsi qu'une variable pour mémoriser l'état du relais :

```
constint relay_pin = 8;
boolean relay_state = false;
```

Ensuite, nous devons créer une variable pour calculer la valeur du courant et de l'électricité consommée par la lampe. Si la tension est de 110 V au lieu de 230 V, vous devrez inscrire la bonne valeur de variable « effective voltage » (tension efficace) :

```
float amplitude_current;
float effective_value;
float effective_voltage = 230; // Set voltage to 230V (Europe)
                               // or 110V (US)
float effective_power;
float zero_sensor;
```

À présent, initialisons l'écran dans la fonction `setup()` du sketch :

```
lcd.init();
```

Définissons à présent la broche du relais en tant que sortie :

```
pinMode(relay_pin,OUTPUT);
```

Il nous faut maintenant procéder à la **calibration du capteur de courant**.

En termes de capteur de courant, j'ai porté mon choix sur un capteur analogique qui renvoie une tension proportionnelle au courant mesuré. Toutefois, il nous faut connaître la tension pour laquelle le capteur enregistre une mesure de courant nulle. Comme le relais est éteint pour le moment, nous sommes certains qu'aucun courant électrique ne traverse la lampe.

On obtient la valeur nulle de courant au niveau du capteur en appliquant la fonction appelée `getSensorValue()` sur la broche A1 :

```
zero_sensor = getSensorValue(A1);
```

Nous ne verrons pas cette fonction en détail mais il faut retenir qu'elle stocke une moyenne de mesures effectuées sur la broche analogique afin d'obtenir une valeur stable en sortie.

On mémorise ensuite le résultat dans une variable appelée `zero_sensor`.

Pour conclure la fonction `setup()`, il nous faut instaurer un léger délai avant d'effacer ce qui apparaît à l'écran :

```
delay(2000);
lcd.clear();
```

Après cette étape, dans la fonction `loop()` du sketch, nous allons d'abord lire les données de l'entrée analogique A0 qui nous renverra une valeur comprise entre 0 et 1023.

La résolution du convertisseur analogique-numérique de la carte Arduino Uno est de 10 bits, soit 1 024 valeurs.

Nous divisons ensuite cette valeur par 1024 et multiplions le résultat par 100 pour obtenir un niveau d'éclairage en pourcentage :

```
float sensor_reading = analogRead(A0);
float light = sensor_reading/1024*100;
```

On obtient ensuite la valeur du capteur de courant de la même façon (moyenne des valeurs sur plusieurs échantillons) :

```
float sensor_value = getSensorValue(A1);
```

Puis on calcule l'intensité de courant et la puissance à partir de cette valeur.

Pour le calcul du courant, nous prendrons les données de calibration obtenues précédemment ainsi que la formule figurant sur la fiche technique du capteur. On obtient ensuite la valeur efficace du courant en divisant le résultat par la racine carrée de 2.

Pour finir, vous obtiendrez la puissance efficace en multipliant la valeur efficace du courant par la valeur efficace de la tension (que l'on divisera aussi par 1 000 pour un résultat en watts) :

```
amplitude_current =
(float)(sensor_value-zero_sensor)/1024*5/185*1000000;
effective_value = amplitude_current/1.414;
effective_power = abs(effective_value*effective_voltage/1000);
```

Il nous faut maintenant décider d'**activer ou non le relais**.

Un niveau d'éclairage supérieur à 75 % signifie qu'il fait clair, la lampe sera donc éteinte :

```
if (light >75) {
  digitalWrite(relay_pin, LOW);
  relay_state = false;
}
```

À l'inverse, nous activerons la lampe pour un niveau d'éclairage inférieur à 50 % :

```
if (light <50) {
  digitalWrite(relay_pin, HIGH);
  relay_state = true;
}
```

Pour les deux valeurs limites, vous devez bien entendu utiliser vos propres valeurs.

Vous pouvez mesurer par exemple le niveau d'éclairage pendant la nuit, lorsque la pièce se trouve dans l'obscurité quasi totale et que l'éclairage provient uniquement de la lampe.

Pour déterminer l'autre valeur limite, on mesure la valeur de l'éclairage durant la journée, avec seulement la lumière du jour.

Vous allez donc modifier les deux valeurs limites en fonction de ces paramètres.

Il est important d'insister sur le fait que deux valeurs limites sont nécessaires ici. Gardez à l'esprit que la valeur donnée par la photorésistance peut varier légèrement au fil du temps. Il vous faut donc deux valeurs limites pour ne pas voir votre lampe s'allumer et s'éteindre en permanence (si une seule valeur limite est établie).

La dernière étape consiste à faire apparaître les données sur l'écran LCD.

On inscrit en premier lieu l'état du relais :

```
lcd.setCursor(1,0);
lcd.print("Relay: ");
if (relay_state) {lcd.print("On ");}
else {lcd.print("Off");}
```

Ensuite, on ajoute la puissance efficace calculée à partir des données du capteur de courant :

```
lcd.setCursor(1,1);
lcd.print("Power: ");
lcd.print(effective_power);
lcd.print("W");
```

On inscrit maintenant la valeur du niveau d'éclairage ambiant :

```
lcd.setCursor(1,2);
lcd.print("Light: ");
lcd.print(light);
lcd.print("%");
```

On ajoute un intervalle de 500 ms entre chaque prise de mesure :

```
delay(500);
```

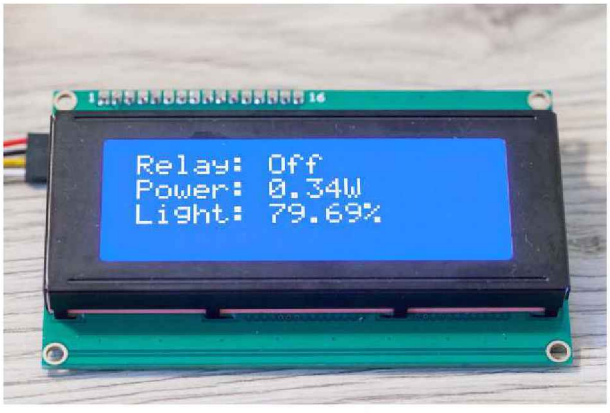
Vous pouvez retrouver le code complet du chapitre dans le dossier correspondant du dépôt GitHub du livre :
https://github.com/openhomeautomation/home-automation-arduino

Il nous faut voir si notre **lampe intelligente fonctionne**.

Assurez-vous que tous les branchements ont été correctement réalisés et que les codes requis sont chargés sur la carte Arduino.

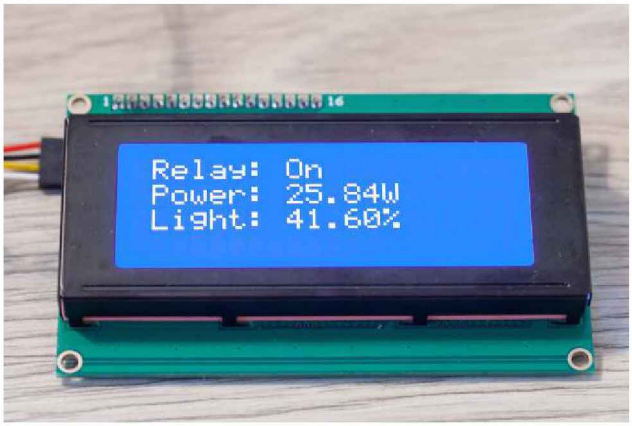
Supposons qu'il fasse clair dans votre pièce : le relais devrait donc être désactivé.

Voici ce que votre afficheur LCD est censé afficher :



Vous pouvez voir que même si la lampe est éteinte, elle provoque malgré tout une faible consommation d'énergie d'après la carte Arduino. Cela vient du fait que la sortie du capteur de courant ACS712 capte du bruit attribuable à plusieurs facteurs, comme les champs magnétiques.

Pour simuler les conditions d'une pièce sombre, je masque la photorésistance avec un morceau de tissu. Le niveau d'éclairage chute alors et la lampe s'allume.



Vous remarquerez aussi que la puissance mesurée avoisine les 25 watts, ce qui est cohérent puisque la puissance de la lampe est de 30 watts.

Si votre projet ne fonctionne pas à ce stade, il vous faut vérifier plusieurs choses :

- Assurez-vous d'abord que le relais, l'afficheur LCD et les capteurs sont correctement raccordés à votre carte Arduino.
- Veillez à modifier le code si vous utilisez un écran LCD de plus petite taille que le mien.

POUR ALLER PLUS LOIN

Résumons ce que nous avons appris. Nous avons créé une lampe intelligente à l'aide d'Arduino et de quelques composants de base. La lampe s'allume ou s'éteint automatiquement en fonction du niveau d'éclairage ambiant. Nous avons ajouté un nouvel élément au projet, un dispositif de mesure de courant, qui fournit des informations en termes de puissance consommée par la lampe lorsqu'elle est en marche. Nous nous sommes également servi d'un afficheur LCD pour afficher l'état de la lampe, sa consommation énergétique ainsi que le niveau d'éclairage ambiant.

Vous pouvez compléter ce projet en ajoutant d'autres capteurs. Il vous est possible de le fusionner avec celui du chapitre précédent. Vous disposez alors d'un plus grand nombre de données à l'écran. Vous pouvez acquérir un contrôle plus étendu de la lampe pour la rendre plus « intelligente ». Ajoutez par exemple au montage un détecteur de mouvement afin que la lampe s'allume si le niveau d'éclairage est bas et si des mouvements sont détectés.

8 Installer des détecteurs de mouvement avec XBee

Lors des chapitres précédents, nous avons construit des installations domotiques capables de fonctionner en parfaite autonomie, sans interagir avec l'extérieur. Les systèmes vendus dans le commerce ne fonctionnent cependant pas selon ce principe. Leurs composants communiquent ensemble sans fil. C'est d'ailleurs exactement de cette manière que nous allons procéder dans les chapitres restants.

Pour des installations domotiques reposant sur la plateforme Arduino, nous allons faire appel à une technologie largement répandue : XBee. XBee est une technologie basée sur le protocole ZigBee, permettant de communiquer par ondes radio de faible puissance en s'appuyant sur la norme IEEE 802.15. Elle a été conçue pour les systèmes à petits transferts de données (comme les capteurs) fonctionnant sur piles et nécessitant d'être protégés. Cette technologie est donc parfaitement adaptée aux installations domotiques.

Nous allons ici nous appuyer sur un des projets du livre consacré à la création d'un système d'alarme basé sur Arduino et un détecteur de mouvement PIR, que nous allons compléter par l'ajout d'un module XBee. Nous verrons comment créer une interface sur votre ordinateur indiquant l'état de plusieurs de ces détecteurs de mouvement sans fil XBee à partir d'un navigateur web.

Après cette opération, nous verrons les fonctionnalités qu'il est possible d'ajouter à nos détecteurs sans fil. Il est possible de détecter du monoxyde de carbone, émanant d'un réchaud défectueux par exemple.

MATÉRIEL ET LOGICIEL REQUIS

Dressons tout d'abord une liste des composants requis pour ce projet. Celui-ci s'articule autour de deux éléments clés. Les **détecteurs de mouvement** formeront le premier ensemble. De l'autre côté, nous aurons un **module XBee** connecté en USB à votre ordinateur.

Il vous faudra une carte Arduino pour chaque détecteur. Mon choix s'est porté une fois de plus sur le modèle **Arduino Uno**.

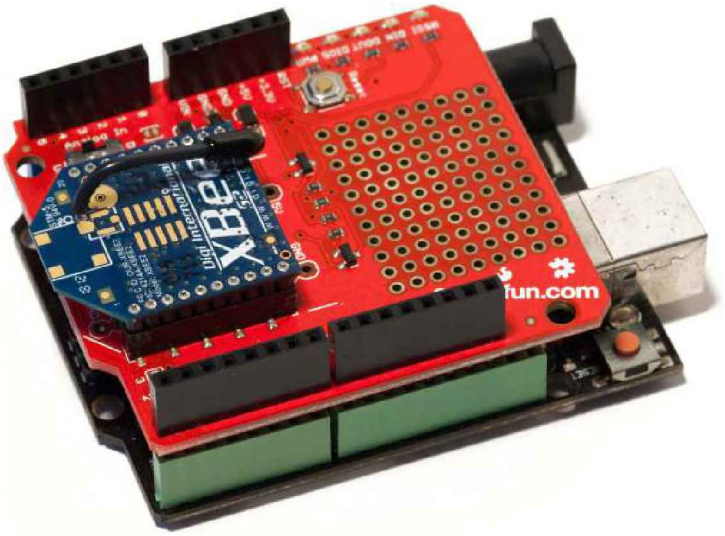
Procurez-vous également un **détecteur de mouvement**. J'ai opté pour le même détecteur de mouvement PIR utilisé plus tôt dans l'ouvrage.

Créons à présent une interaction entre le module XBee et la carte Arduino. Afin de raccorder les deux éléments, je me suis servi d'une carte d'extension XBee pour Arduino de la marque Sparkfun. Cette carte dispose d'une fiche permettant le raccord à tout

type de module XBee et d'un interrupteur pour connecter et déconnecter le module du port série du microcontrôleur Arduino. Nous pourrons constater sa très grande utilité un peu plus loin dans le chapitre.

En ce qui concerne les modules, j'ai choisi des modèles **XBee Série 2** équipés d'une antenne filaire. Les modèles Série 1 sont plus simples d'utilisation mais ceux de la gamme Série 2 permettent de créer des réseaux maillés et de cibler un module en particulier, ce qui nous sera particulièrement utile dans ce chapitre.

Voici un aperçu de la carte d'extension XBee imbriquée sur la carte Arduino, avec en outre un module XBee :



LISTE DES COMPOSANTS

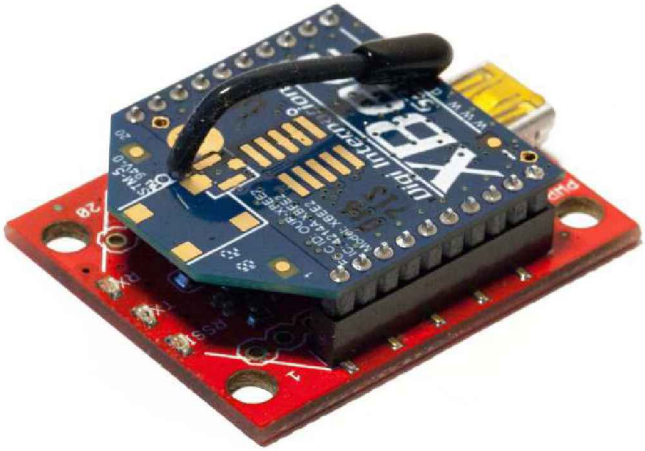
- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Détecteur de mouvement PIR (http://snootlab.com/adafruit/285-capteur-de-presence-pir.html)
- Carte d'extension XBee pour Arduino (http://ultra-lab.net/fr/tienda/shield-xbee)
- Module XBee Série 2 avec antenne (http://www.robotshop.com/eu/fr/module-xbee-2mw-antenne-serie-2-zibbee-mesh.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

Il vous faut à présent faire en sorte que votre ordinateur puisse se connecter à des modules XBee. En effet, contrairement aux protocoles Bluetooth et WiFi, les ordinateurs ne disposent pas du matériel pour communiquer avec un module XBee.

Nous utiliserons une carte XBee Explorer USB de chez Sparkfun pour connecter le module XBee à l'ordinateur. Vous avez la possibilité d'y brancher un module et de le raccorder en USB à votre ordinateur. Elle s'utilise comme un port série, ce qui signifie que vous pouvez lui envoyer des messages *via* le moniteur série du logiciel Arduino IDE.

Pour ce projet, j'ai choisi le même module XBee que pour les détecteurs de mouvement XBee.

Voici l'aperçu d'un module XBee branché sur la carte XBee Explorer USB :



LISTE DES COMPOSANTS (SUITE)

Afin d'utiliser XBee sur votre ordinateur, il vous faudra ces composants :

- Carte XBee Explorer USB (http://www.gotronic.fr/art-module-xbee-explorer-usb-20253.htm)
- Module XBee Série 2 avec antenne (http://www.robotshop.com/eu/fr/module-xbee-2mw-antenne-serie-2-zibbee-mesh.html)

Côté logiciel, installez sur votre ordinateur, si ce n'est pas déjà fait, la dernière version du programme Arduino IDE ainsi que la bibliothèque aREST pour Arduino.

Vous trouverez la bibliothèque aREST à cette adresse :
https://github.com/marcoschwartz/aREST

Afin d'installer une bibliothèque, il vous suffit d'extraire son dossier vers **/libraries**, situé dans le dossier racine Arduino (créez ce dossier s'il n'existe pas).

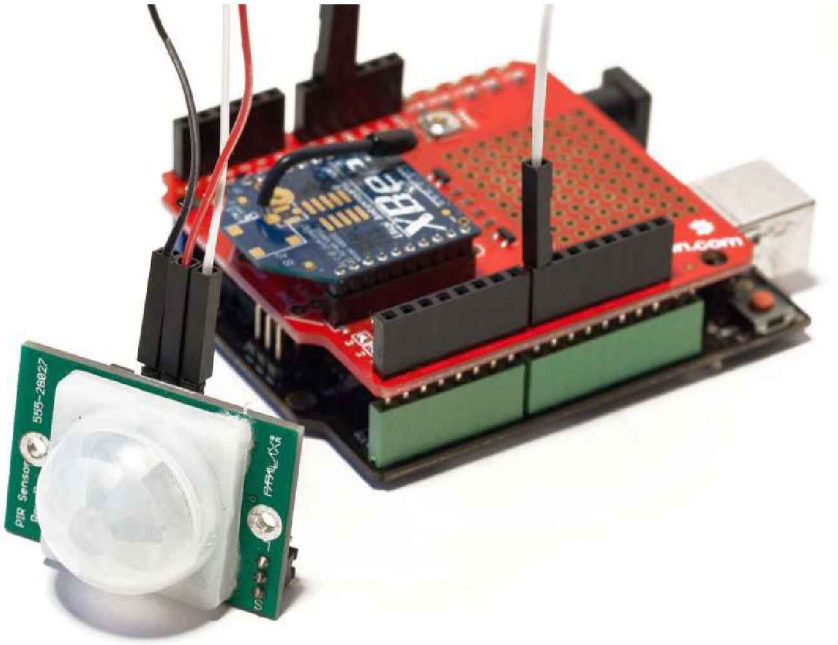
RÉALISER UN DÉTECTEUR DE MOUVEMENT SANS FIL XBEE

Nous allons voir comment réaliser un détecteur de mouvement XBee. Si vous souhaitez utiliser plusieurs détecteurs pour ce projet, répétez les étapes suivantes pour chaque module.

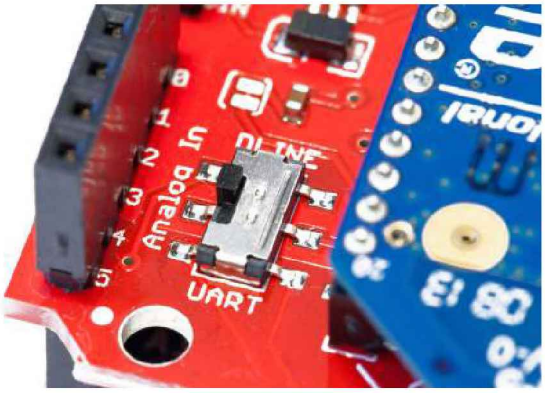
La partie configuration est très simple. La première étape consiste à raccorder la carte d'extension XBee à la carte Arduino et à brancher un module XBee sur la carte d'extension.

En ce qui concerne le détecteur de mouvement PIR, connectez la broche GND à la masse Arduino, la broche VCC à la broche Arduino 5 V et la broche SIG à la broche Arduino n° 8.

Vous devriez aboutir à un résultat semblable à celui-ci :



Vérifiez la bonne position de l'interrupteur sur la carte d'extension XBee. En effet, nous devons à ce stade veiller à ce que le module XBee ne soit pas connecté au port série du microcontrôleur Arduino. Pour ce faire, réglez la position de l'interrupteur sur « DLINE ».



Pour finir, connectez à votre ordinateur la carte XBee Explorer USB assemblée au module XBee.

TESTER LE DÉTECTEUR DE MOUVEMENT

Testons à présent le détecteur de mouvement. Nous allons écrire un simple sketch consacré à l'affichage permanent du statut du détecteur de mouvement sur le port

série de l'Arduino. Pour l'instant, laissez l'interrupteur de la carte d'extension XBee sur la position « DLINE ».

Voici la version complète du sketch de cette partie :

```
int sensor_pin = 8;

void setup() {
  Serial.begin(9600);
}

void loop() {

// Read sensor data
int sensor_state = digitalRead(sensor_pin);

// Print data
  Serial.print("Motion sensor state: ");
  Serial.println(sensor_state);
  delay(100);
}
```

On commence le sketch en déclarant la broche à laquelle le détecteur de mouvement PIR est connecté :

```
int sensor_pin = 8;
```

À présent, on démarre le port série dans la fonction `setup()` du sketch :

```
Serial.begin(9600);
```

Dans la fonction `loop()` du sketch, il nous faut lire les données de la broche du détecteur de mouvement PIR :

```
int sensor_state = digitalRead(sensor_pin);
```

Enfin, on fait apparaître l'état de cette broche dans le port série toutes les 100 ms :

```
Serial.print("Motion sensor state: ");
Serial.println(sensor_state);
delay(100);
```

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

Il est temps à présent de tester le premier sketch de ce chapitre.

Chargez-le sur la carte Arduino et ouvrez le moniteur série (en vous assurant que la vitesse du port série est réglée à 9600). Tentez de passer votre main devant le détecteur. Vous devriez observer un changement de l'état du détecteur.

```
Motion sensor state: 0
Motion sensor state: 0
Motion sensor state: 0
```

```
Motion sensor state: 0
Motion sensor state: 1
Motion sensor state: 1
Motion sensor state: 1
Motion sensor state: 1
Motion sensor state: 0
Motion sensor state: 0
Motion sensor state: 0
```

Si vous n'obtenez pas ce résultat à ce stade, il vous faut vérifier plusieurs choses :

- Assurez-vous que le détecteur de mouvement est correctement raccordé à la carte Arduino en vous référant aux indications données plus tôt dans le chapitre.
- Vérifiez ensuite que vous avez chargé la version du code la plus récente.

UTILISER LE MODULE XBEE

Cette partie sera consacrée au maniement du module XBee installé sur la carte d'extension pour accéder sans fil au détecteur de mouvement.

Nous allons écrire un simple sketch Arduino dans lequel on utilisera la bibliothèque aREST pour recevoir et traiter les requêtes provenant de l'extérieur.

Mais avant cela, il est nécessaire de configurer nos radios XBee. Par défaut, les modules XBee sont paramétrés pour utiliser le même canal de communication appelé « PAN ID ».

Vous pouvez tester ces modules immédiatement, sans opérer de modification supplémentaire. En revanche, évitez cela à tout prix si vous comptez les utiliser dans votre installation domotique. En effet, étant définis sur le PAN ID par défaut, ils diffuseront *via* ce canal des messages vers tous les appareils aux alentours, y compris ceux de vos voisins. Pire encore, cela rendrait vos détecteurs XBee vulnérables au piratage.

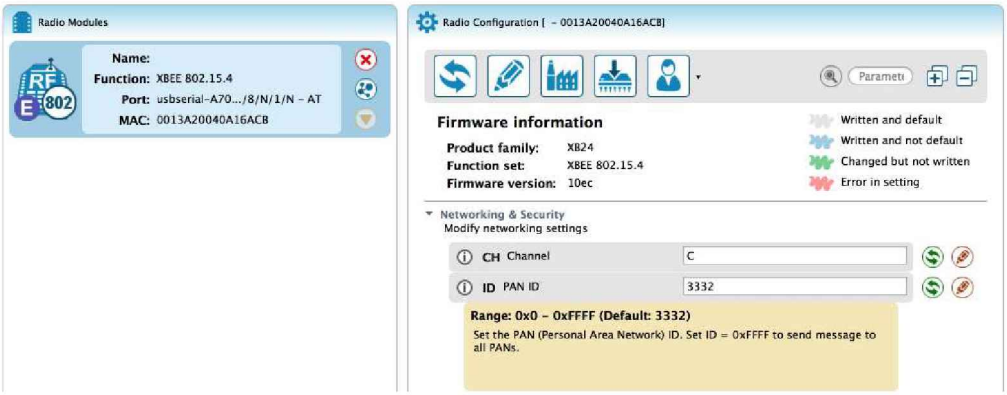
Il est également nécessaire de définir correctement nos radios en leur attribuant des rôles. Au sein d'un réseau XBee Series 2, l'une des radios doit être définie en tant que « *coordinator* ». C'est indispensable au fonctionnement du réseau. Les autres radios peuvent être définies comme « *routers* », relayant des messages, ou en tant que « end devices », dont la seule fonction est de réceptionner des messages du réseau. Étant donné que nous avons ici un réseau simple, une configuration basique suffit : définissez la radio XBee connectée à votre ordinateur comme « coordinator » et les autres en tant que « end devices ».

De ce fait, nous devons modifier ces paramètres. Ces modules XBee sont très simples à configurer *via* le logiciel officiel, appelé X-CTU.

> Vous pouvez le télécharger à l'adresse suivante :
> http://www.digi.com/support/productdetail?pid=3352&type=utilities

Une fois installé, ouvrez le logiciel et connectez le module XBee que vous souhaitez configurer à la carte XBee Explorer branchée en USB à votre ordinateur. En ouvrant le port série de la carte XBee Explorer, vous aurez accès aux paramètres du module XBee.

Premièrement, dans la fenêtre « Radio Configuration », recherchez « PAN ID » :

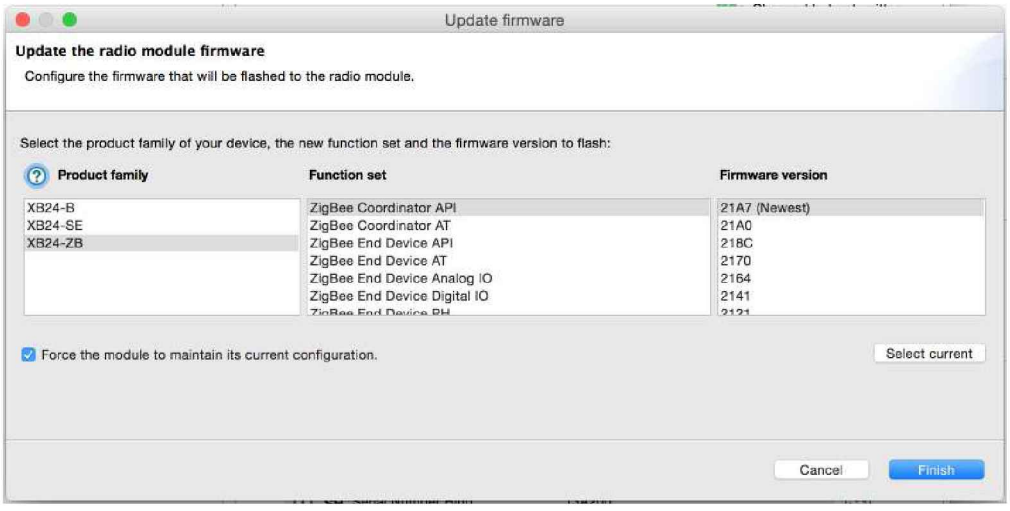


Depuis ce menu, vous pouvez définir le « PAN ID » du module XBee branché à la carte XBee Explorer. Il vous suffit de répéter l'opération pour chacun des modules XBee que vous souhaitez configurer.

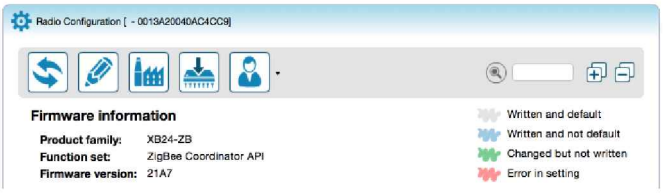
Donnons des rôles à nos modules. Raccordez maintenant le module qui servira de « coordinator » à la carte XBee Explorer. Ajoutez le module *via* le menu de gauche et cliquez sur le bouton de mise à jour du micrologiciel (illustré par une flèche qui pointe vers la carte) de la barre de menu supérieure.

Grâce à ce bouton, vous pourrez sélectionner le micrologiciel nécessaire au « coordinator » XBee. Sélectionnez sur la gauche « XB24-ZB », puis « ZigBee Coordinator API » à partir de la liste.

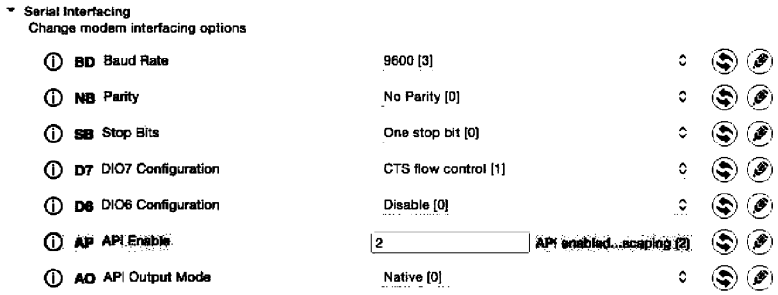
Choisissez la dernière version du micrologiciel et confirmez :



Quelques instants après, vous constaterez que votre module est paramétré en tant que « coordinator XBee » en mode « API » :



Toujours pour ce module XBee, il nous faut descendre dans la liste des paramètres et sélectionner « 2 » pour le paramètre « API enable parameter » et le sauvegarder dans la radio XBee à l'aide du bouton de droite :



Enfin, il nous faut configurer les autres radios XBee en tant que « End Devices ». Connectez chacune d'elles à la carte XBee Explorer, cliquez sur le bouton de mise à jour du micrologiciel, sélectionnez sur la gauche « XB24-ZB » puis « ZigBee End Device AT » dans la liste. Confirmez et voici ce que vous devriez obtenir :

Créons maintenant ce sketch Arduino basé sur le sketch test. Je définirai ici uniquement le code qui a été rajouté. Voici la version complète du sketch de cette partie :

```
#include <SPI.h>
#include <aREST.h>

// Motion sensor ID
char * xbee_id = "1";

// Create ArduREST instance
aREST rest = aREST();

void setup() {

// Start Serial
  Serial.begin(9600);

// Give name and ID to device
  rest.set_id(xbee_id);
  rest.set_name("motion1");
}

void loop() {

// Handle REST calls
  rest.handle(Serial);

}
```

On débute le sketch en incluant les bibliothèques requises :

```
c #include <aREST.h>
```

On définit également le numéro d'identification du détecteur. Cette étape est primordiale si vous possédez plusieurs détecteurs chez vous. Assurez-vous de leur attribuer des numéros d'identification différents :

```
String xbee_id = "1";
```

Il nous faut aussi créer une instance de la bibliothèque aREST :

```
aREST rest = aREST();
```

À présent, initialisons le port série dans la fonction `setup()` du sketch. Il est essentiel de régler sa vitesse à 9600, étant donné que les modules XBee sont par défaut configurés à cette vitesse :

```
Serial.begin(9600);
```

Définissons également le numéro d'identification du montage défini :

```
rest.set_id(xbee_id);
```

Pour finir, dans la fonction `loop()` du sketch, nous allons simplement traiter les requêtes venant du port série à l'aide de la bibliothèque aREST :

```
rest.handle(Serial);
```

Vous pouvez retrouver le code complet de cette partie dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

Il est temps à présent de tester ce sketch. Vous pouvez désormais le charger sur la carte Arduino. Réglez ensuite l'interrupteur de la carte d'extension XBee sur « UART » pour que le module XBee communique directement, *via* le port série, avec le microcontrôleur de la carte Arduino.

> Si vous devez reprogrammer la carte Arduino, il est indispensable de repasser en position « **DLINE** ».

Vérifiez aussi que la carte XBee Explorer est bien connectée à votre ordinateur. Si vous travaillez sous Windows ou OS X, vous serez peut-être amené à installer des drivers supplémentaires pour cette carte USB.

Vous trouverez à cette adresse toutes les informations (en anglais) relatives au téléchargement et à l'installation de ces drivers : http://www.ftdichip.com/FTDrivers.htm

Vous devez maintenant localiser le port série qui correspond à la carte XBee Explorer connectée à votre ordinateur. Réalisez cette opération en vous rendant dans le menu Outils > Port Série du logiciel Arduino IDE. Le nom du port série devrait se présenter sous cette forme : **/dev/cu.usbserial-A702LF8B**. Sous Windows, vous obtiendrez un résultat semblable à « COM3 ». Je vous conseille de le noter puisque vous en aurez besoin lors de la création de l'interface pour vos détecteurs de mouvement.

Ouvrez le moniteur série du logiciel Arduino IDE. Vérifiez que la vitesse est réglée à 9600 et que le caractère de fin de ligne est défini sur « Carriage return ». Notez que nous sommes à présent connectés à la carte XBee Explorer USB, ce qui signifie que toutes les commandes envoyées parviennent à l'ensemble des modules XBee de votre domicile.

Saisissez dans le moniteur série :

```
/id
```

Cette entrée est chargée de demander le numéro d'identification des modules XBee détectés chez vous. Lorsque j'ai effectué le test, je ne disposais que d'un seul module chez moi. Voici la réponse reçue :

```
{"id": "1", "name": "", "connected": true}
```

Continuons avec la lecture du statut du détecteur de mouvement. N'oublions pas que celui-ci est connecté à la broche n° 8. Pour lire à partir de cette broche, Saisissez :

```
/digital/8
```

Le détecteur devrait répondre par ce message :

```
{"return_value": 1, "id": "1", "name": "", "connected": true}
```

Si jusqu'ici les détecteurs répondent aux requêtes, vous pouvez en conclure qu'ils fonctionnent correctement et que vous arrivez à y accéder sans fil.

Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Vérifiez tout d'abord que vous avez chargé la dernière version du code. Vous la trouverez dans le dépôt GitHub du livre.
- Assurez-vous aussi que la carte XBee Explorer est connectée à votre ordinateur en vérifiant que les drivers ont bien été installés.
- Pour finir, vérifiez que l'interrupteur de la carte d'extension XBee est positionné sur « UART ».

METTRE AU POINT L'INTERFACE CENTRALE

Nous allons maintenant créer l'interface qui nous permettra de piloter les détecteurs de mouvement à partir d'un ordinateur. Grâce à cette interface, vous pourrez visualiser l'état de chacun des détecteurs *via* XBee au sein de votre navigateur web.

Nous allons nous servir de Node.js afin de la développer. Cette plateforme permet de programmer des applications côté serveur en JavaScript.

Créons d'abord le fichier principal, nommé app.js, que nous exécuterons plus tard en utilisant la commande `node` dans un terminal. Celui-ci contient le cœur de la partie logicielle du projet.

Vous trouverez ci-dessous la version complète du code pour ce fichier :

```
var express = require('express');
var app = express();

// Define port
var port = 3000;

// View engine
app.set('view engine', 'jade');

// Set public folder
app.use(express.static(__dirname + '/public'));

// Rest
var rest = require("arest")(app);
rest.addDevice('xbee','/dev/tty.usbserial-A702LF8B');

// Serve interface
app.get('/', function(req, res){
  var devices = rest.getDevices();
  res.render('interface', {devices: devices});
});
```

```
// Start server
app.listen(port);
console.log("Listening on port " + port);
```

Commençons par l'importation du module Express :

```
var express = require('express');
```

Créons notre application à partir de la structure Express et définissons le port à 3000 :

```
var app = express();
var port = 3000;
```

Il nous faut aussi indiquer à notre logiciel l'emplacement de l'interface graphique que nous programmerons ultérieurement. Définissons également **Jade** en tant que moteur de vue (*view engine*) par défaut, ce qui, vous le verrez, rend la programmation en HTML bien plus simple :

```
app.set('view engine', 'jade');
app.use(express.static(__dirname + '/public'));
```

À ce stade, il nous faut importer le module `node-aREST`, chargé de traiter les échanges entre l'interface et le module. Nous devons définir le port série sur lequel la carte XBee Explorer USB est connectée :

```
var rest = require("arest")(app);
rest.addDevice('xbee','/dev/tty.usbserial-A702LF8B');
```

Une recherche automatique est ainsi lancée pour détecter l'ensemble des modules XBee au sein du même réseau que notre radio « coordinator » et les ajouter ensuite au serveur afin d'y accéder à partir de l'interface.

Nous allons maintenant créer la « route » principale de notre serveur. Définissons-la en associant l'URL racine de notre serveur au fichier **Jade** correspondant[1]. Nous voulons créer l'interface de manière automatique en fonction du nombre de modules présents. Pour ce faire, récupérons premièrement l'information sur l'ensemble des modules et transférons ces données vers le fichier Jade pour assurer le rendu de l'interface :

```
app.get('/', function(req, res){
  var devices = rest.getDevices();
  res.render('interface', {devices: devices});
});
```

Ensuite, toujours dans le fichier **app.js**, il ne reste plus qu'à lancer l'application sur le port série défini précédemment et écrire un message dans la console :

```
app.listen(port);
console.log("Listening on port " + port);
```

1. Jade est un langage de « *templating* » permettant d'écrire du HTML de manière concise. Grâce à lui, on peut insérer du code directement dans le fichier de l'interface pour générer du HTML en fonction de variables données.

Voilà pour ce qui est du fichier principal du serveur. Nous allons maintenant créer l'interface. Voyons d'abord ce que ce fichier **Jade** contient. Vous le trouverez dans le dossier **/views** de notre projet.

Vous trouverez ci-dessous la version complète du code pour ce fichier :

```
doctype
html
  head
    title XBee motion sensors
    link(rel='stylesheet',

href='//maxcdn.bootstrapcdn.com/bootstrap/3.3.0/css/bootstrap.min.css')
    script(src="https://code.jquery.com/jquery-2.1.1.js")
    script(src="/js/interface.js")
  body
    .container
      .row
      h1 XBee motion sensors
        if (devices != '[]')
          each device in devices
            if (device.type == 'xbee')
              .row
                .col-md-4
                  h3 Sensor #{device.id}
                .col-md-4
                  h3.display(id=device.id)
```

Premièrement, on importe les différents fichiers JavaScript qui traiteront les clics sur l'interface et enverront les commandes correspondantes à la carte Arduino :

```
script(src="https://code.jquery.com/jquery-2.1.1.js")
script(src="/js/interface.js")
```

Nous allons aussi utiliser la structure Twitter Bootstrap CSS pour améliorer l'aspect de notre interface :

```
link(rel='stylesheet',
href='//maxcdn.bootstrapcdn.com/bootstrap/3.3.0/css/bootstrap.min.css')
```

La partie principale de l'interface consiste à parcourir la liste de tous les détecteurs reconnus précédemment et à créer une ligne dans l'interface pour chacun d'entre eux. Pour cela, saisissez la ligne de code suivante :

```
jade if (devices != '[]')  each device in devices  if (device.type
== 'xbee')  .row  .col-md-4  h3 Sensor #{device.id}   .col-md-4
h3.display(id=device.id)
```

Intéresserons-nous au code contenu dans le fichier **interface.js** qui définit la manière dont l'interface du projet fonctionne. On y effectue des requêtes vers la carte par le biais du serveur Node.js et on met l'interface à jour en fonction. Le fichier se situe dans le dossier **public/js** de l'interface.

Voici la version complète du code pour ce fichier :

```
$(document).ready(function() {

  $.get('/devices', function( devices ) {

    // Set inputs
    for (i = 0; i < devices.length; i++){

      // Get device
      var device = devices[i];

      // Set input
      $.get('/' + device.name + '/mode/8/i');

    }

    setInterval(function() {

      for (i = 0; i < devices.length; i++){

        // Get device
        var device = devices[i];

        // Get data
        $.get('/' + device.name + '/digital/8', function(json_data)
{

            // Update display
            if (json_data.return_value == 0){
              $("#" + json_data.id).html("No motion");
              $("#" + json_data.id).css("color","red");
            }
            else {
              $("#" + json_data.id).html("Motion detected");
              $("#" + json_data.id).css("color","green");
            }

        });
      }

    }, 2000);

  });

});
```

Ce fichier JavaScript est principalement consacré à la vérification permanente de l'état des détecteurs au moyen de requêtes. On effectue cela dans la fonction `setInterval()` :

```
setInterval(function() {
```

À l'intérieur de cette fonction, on parcourt l'ensemble des modules. Pour chaque module, on envoie une commande destinée à lire les données de la broche à laquelle les détecteurs de mouvement sont connectés :

```
$.get('/' + device.name + '/digital/8', function(json_data) {
```

La bibliothèque aREST nous retourne systématiquement les données au format JSON, ce qui permet de les utiliser facilement dans JavaScript. Il nous faut connaître l'état de ce détecteur de mouvement. Cette donnée est stockée dans le champ `return_value`. Si la valeur de ce champ est nulle, on définit alors l'indicateur sur « No motion » en rouge. Dans le cas contraire, on le définit en vert sur « Motion detected », ce qui signifie que des mouvements ont été enregistrés par ce détecteur :

```
if (json_data.return_value == 0){
    $("#" + json_data.id).html("No motion");
    $("#" + json_data.id).css("color","red");
}
else {
    $("#" + json_data.id).html("Motion detected");
    $("#" + json_data.id).css("color","green");
}
```

Une fois cette étape accomplie, on ferme la fonction `setInterval()`, ce qui a pour effet de répéter la boucle toutes les deux secondes :

```
}, 2000);
```

Vous pouvez retrouver le code complet de cette partie dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

Testons à présent l'interface. Assurez-vous de télécharger l'ensemble des fichiers du dépôt GitHub et chargez le code en incorporant, le cas échéant, vos propres données comme le port série correspondant à votre carte XBee Explorer. Vérifiez aussi que la carte est programmée avec le code donné précédemment dans ce chapitre.

Rendez-vous ensuite dans le dossier de l'interface en utilisant un terminal et saisissez la commande suivante pour installer les modules **node-aREST**, **Express** et **Jade** :

```
sudo npm install arest express jade
```

Si vous utilisez Windows, vous devez laisser de côté **sudo** qui se situe devant les commandes. Il est recommandé d'utiliser le terminal installé par défaut avec Node.js. Pour finir, vous pouvez lancer le serveur Node.js en tapant :

```
sudo node app.js
```

Vous obtiendrez alors le message suivant dans le terminal :

```
Listening on port 3000
```

Vous constaterez aussi que vos radios XBee sont détectées et ajoutées de manière automatique au système. Ouvrez à présent votre navigateur web et saisissez :

```
localhost:3000
```

L'interface de notre projet devrait désormais apparaître :

Pour tester le projet, j'ai utilisé deux détecteurs de mouvement. Afin d'apprécier le comportement d'un des modules, j'ai simplement passé ma main devant le détecteur portant le numéro d'identification n° 1. L'indicateur correspondant est instantanément passé au vert. Si vous disposez de plusieurs détecteurs, l'interface va bien entendu changer automatiquement.

Si vous n'obtenez pas ce résultat à ce stade, il vous faut vérifier plusieurs choses :

- Assurez-vous d'utiliser la dernière version du code à partir du dépôt GitHub du livre.
- Vérifiez aussi que les fichiers ont bien été modifiés selon notamment les paramètres du port série de votre carte XBee Explorer.
- Veillez à installer les modules Node requis avec l'outil npm avant de lancer votre navigateur web.

POUR ALLER PLUS LOIN

Résumons ce que nous avons vu dans ce projet. Nous sommes partis du projet abordé au chapitre 5 et lui avons associé une fonction de communication sans fil grâce à XBee. Nous avons fait interagir des modules XBee avec Arduino et mis au point des détecteurs de mouvement sans fil XBee. Nous avons pu obtenir le suivi de l'état de plusieurs détecteurs de mouvement sans fil à partir d'un navigateur web.

Vous pouvez partir de ce projet pour réaliser une multitude de montages plus complexes. Vous pouvez commencer par incorporer davantage de détecteurs de mouvement au projet afin d'équiper toutes les pièces de votre habitation. Si vous le voulez, vous pouvez ajouter au projet tous types de capteurs numériques. Utilisez le même code pour obtenir par exemple le suivi de l'état d'un capteur de contact que vous pourriez placer contre une porte ou une fenêtre.

Avec vos détecteurs de mouvement sans fil XBee, il est possible d'utiliser des types de capteurs complètement différents. Prenez par exemple un détecteur de fumée (le modèle MQ-7 fonctionne bien avec Arduino). En installant plusieurs détecteurs XBee chez vous, vous rendez possibles à la fois la détection de mouvement, le signalement d'une fenêtre ou d'une porte ouverte et la présence de fumée.

En vous basant sur les principes évoqués dans ce chapitre, vous pouvez aussi, *via* XBee, suivre les données d'un capteur de température.

9 Transmettre des mesures en Bluetooth

Dans ce chapitre, nous allons ajouter une fonction de communication sans fil au projet du chapitre 6 pour créer une station de mesures connectée *via* Bluetooth.

Cette station effectuera des relevés de température, d'humidité et du niveau d'éclairage de la pièce où le montage est installé, puis affichera les données captées sur un afficheur LCD. Cette fois-ci cependant, votre montage contiendra un module Bluetooth qui vous transmettra les valeurs enregistrées depuis n'importe quelle pièce de votre logement. Nous verrons comment créer sur votre ordinateur une interface permettant de connaître les conditions météorologiques de votre habitat à partir d'un navigateur web.

MATÉRIEL ET LOGICIEL REQUIS

Pour ce projet, il vous faudra bien sûr une carte **Arduino Uno**. La réalisation de ce projet est possible avec une carte Arduino Mega ou Leonardo.

Pour prendre des mesures de température et d'humidité, vous aurez besoin d'un **capteur DHT11**, accompagné d'une **résistance de 4,7 kΩ**. L'utilisation d'un capteur DHT22, plus précis, est possible, vous devrez cependant modifier une ligne de code.

Pour mesurer le niveau d'éclairage, j'ai choisi une **photorésistance** dotée d'une résistance de 10 kΩ. Elle renverra un signal proportionnel à la quantité de lumière entrante.

Il est nécessaire de s'équiper d'un **écran LCD** pour afficher les mesures. J'ai opté pour un modèle 4 lignes-20 caractères, ce qui permet d'afficher simultanément quatre mesures. Vous pouvez utiliser un écran de plus petite taille mais vous ne serez pas en mesure de voir en même temps les données de température et d'humidité par exemple.

L'écran LCD que j'ai choisi pour ce projet dispose d'une **interface I2C** permettant d'interagir avec la carte Arduino. Je recommande fortement de prendre un écran équipé de cette interface car pour l'utiliser, seules deux broches seront raccordées à la carte Arduino.

En ce qui concerne le **module Bluetooth**, j'ai choisi le modèle 2.1 de la marque Adafruit. Ce module interagit directement avec le port série de la carte Arduino, ce qui rend son utilisation très pratique. Vous pouvez en principe utiliser n'importe quel module (y compris les modules Bluetooth 4.0) capable de communiquer directement avec le port série du microcontrôleur de la carte Arduino.

Les modules nécessitant leur propre bibliothèque pour Arduino (dotés du circuit nRF8001 par exemple) ne fonctionneront pas immédiatement sans modification supplémentaire dans le code du projet.

Voici un aperçu du module que j'ai utilisé :

Pour finir, il nous faut une **plaque d'essai** ainsi que des fils de raccordement mâle/mâle pour effectuer les différents branchements.

LISTE DES COMPOSANTS

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Capteur DHT11 (http://snootlab.com/adafruit/255-capteur-de-temperateur-et-d-humidite-dht11-extras-.html)
- Photorésistance (http://snootlab.com/composants/97-photoresistance.html)
- Résistance de 10 kΩ (http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html)
- Afficheur LCD (http://www.es-france.com/produit2332/product_info.html)
- Module Bluetooth Adafruit EZ-Link (http://shop.mchobby.be/breakout/396-bluefruit-bluetooth-serie-programmeur-arduino.html)
- Plaque d'essai (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

Côté logiciel, vous aurez besoin de la bibliothèque aREST pour Arduino, de la bibliothèque pour le capteur DHT et de la bibliothèque LiquidCrystal pour l'afficheur LCD.

> Téléchargez la bibliothèque aREST à cette adresse :
> https://github.com/marcoschwartz/aREST
>
> Téléchargez bibliothèque pour le capteur DHT à cette adresse :
> https://github.com/adafruit/DHT-sensor-library
>
> Enfin, téléchargez la bibliothèque LiquidCrystal à cette adresse :
> https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads

Pour installer une bibliothèque, il vous suffit de placer son dossier dans **/libraries**, situé dans le dossier racine Arduino.

MONTER UNE STATION BLUETOOTH

Les connexions matérielles du projet sont relativement simples : le capteur DHT11, la partie gérant la mesure de luminosité avec la photorésistance, l'afficheur LCD et enfin le module Bluetooth.

L'illustration suivante synthétise les connexions matérielles (sans l'écran LCD).

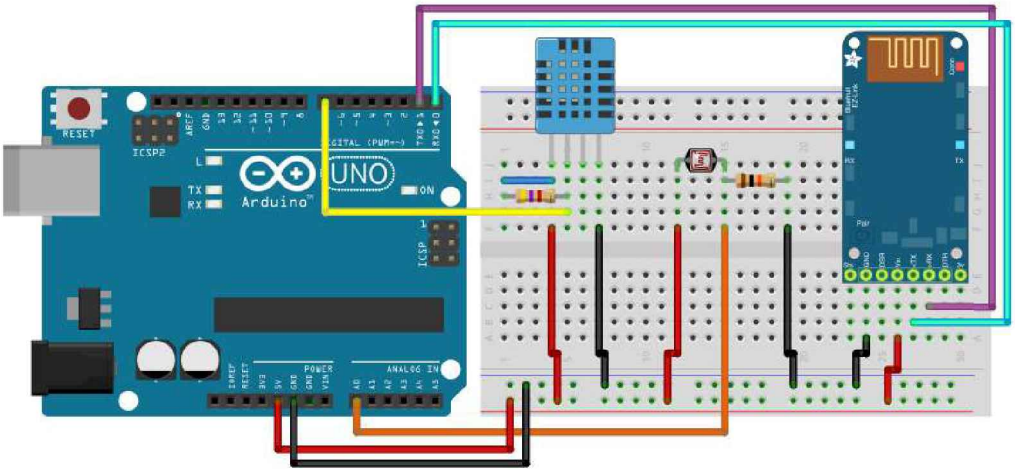
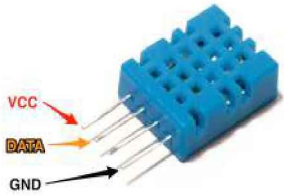


Figure 9.1 Image créée avec le logiciel Fritzing (http://fritzing.org/)

Raccordons tout d'abord la broche 5 V de la carte Arduino Uno à la rangée rouge de la plaque d'essai et la broche de masse à la rangée bleue. Reportez-vous à l'image suivante pour savoir quelle broche du capteur DHT11 raccorder :



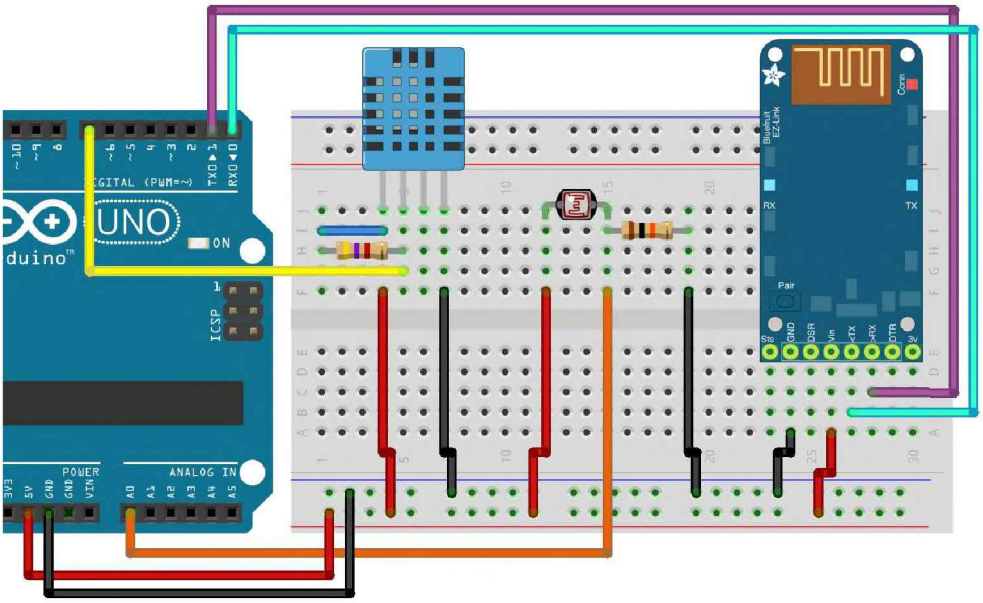
Ensuite, connectez la broche n° 1 du capteur DHT11 (VCC) à la rangée rouge de la plaque d'essai et la broche n° 4 (GND) à la rangée bleue. Après cette étape, connectez la broche n° 2 du capteur à la broche n° 7 de la carte Arduino. Pour en finir avec le branchement du capteur, insérez la résistance de 4,7 kΩ entre les broches n° 1 et n° 2 du capteur.

Connectez la photorésistance en série avec la résistance de 10 kΩ sur la plaque d'essai. L'autre patte de la photorésistance sera raccordée à la rangée rouge de la plaque d'essai et la seconde patte de la résistance à la masse. Enfin, reliez la broche commune entre la photorésistance et la résistance à l'entrée analogique A0 de la carte Arduino.

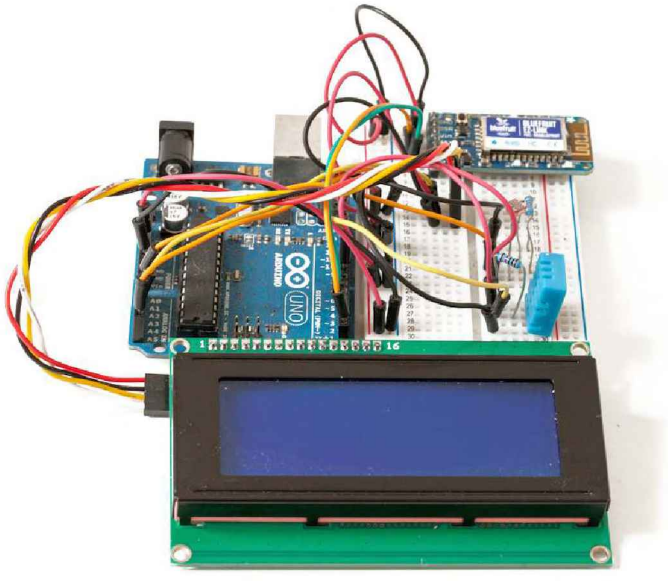
Nous allons maintenant raccorder l'afficheur LCD. Grâce à l'interface I2C de notre afficheur, seuls deux câbles devront être raccordés pour le signal et deux autres pour l'alimentation. Relions tout d'abord la broche VCC du LCD à la rangée rouge de la plaque d'essai et la broche GND à la rangée bleue. Ensuite, raccordons la broche SDA de l'afficheur LCD à l'entrée analogique A4 de la carte Arduino et la broche SCL à l'entrée A5.

Le module Bluetooth est lui aussi assez simple à raccorder. Occupons-nous d'abord de l'alimentation électrique. Connectez la broche VIN (ou VCC, suivant le module) à la rangée rouge de la plaque et la broche GND à la rangée bleue, puis, le port série du

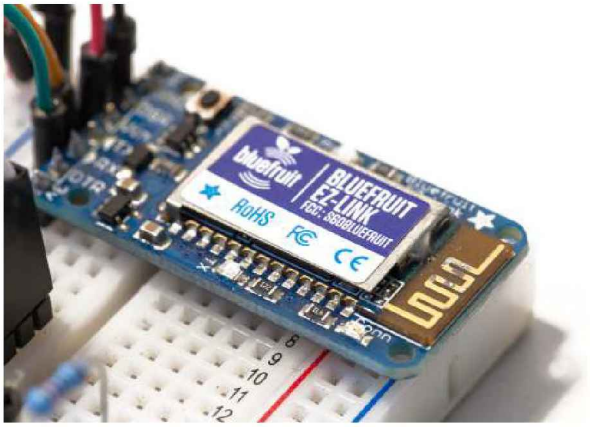
module Bluetooth à la carte Arduino. Raccordez la broche RX du module à la broche TX (broche n° 1) de la carte Arduino et la broche TX du module à la broche RX (broche n° 0) de la carte.



L'illustration suivante vous donne un aperçu visuel du projet entièrement assemblé :



L'image suivante vous permet de voir plus en détail le module Bluetooth sur la plaque d'essai :



JUMELER UN MODULE BLUETOOTH

Avant de continuer, testons le matériel assemblé. Afin d'effectuer le test des différents capteurs du projet, reportez-vous au chapitre 6. Ici, nous allons simplement vérifier l'accès au module Bluetooth.

Mettez le projet sous tension, en branchant par exemple un câble USB entre la carte Arduino et votre ordinateur. Accédez ensuite au menu Préférences de votre ordinateur. Vous devriez voir apparaître le module Bluetooth dans la liste des périphériques détectés :

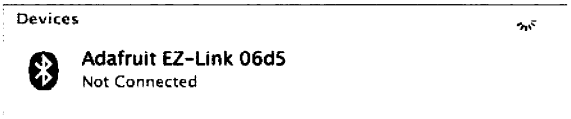


Figure 9.2 Jumelage Bluetooth

Cliquez sur « Jumeler » (« *Pair* » en anglais) afin de jumeler le module Bluetooth pour qu'il fonctionne avec votre ordinateur. Ouvrez ensuite le logiciel Arduino IDE. Sélectionnez Outils > Port Série. Vous devriez constater l'apparition d'un nouveau port série contenant le nom « AdafruitEZ » (ou le nom de votre module Bluetooth) :

Si vous n'obtenez pas ce nouveau port dans la liste, fermez et ré-ouvrez Arduino IDE. Si le module Bluetooth est bel et bien jumelé avec votre ordinateur, il n'y a aucune raison qu'il n'apparaisse pas dans la liste.

MESURER LA TEMPÉRATURE À DISTANCE

En sachant que nous pouvons désormais jumeler le module Bluetooth avec votre ordinateur, nous allons écrire un sketch afin de recevoir des commandes *via* Bluetooth. Pour cela, nous aurons une nouvelle fois recours à la bibliothèque aREST, précédemment utilisée pour traiter les commandes Bluetooth entrantes.

Vous trouverez ci-dessous la version complète du code de cette partie :

```
// Libraries
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "DHT.h"
#include <aREST.h>

// DHT sensor
#define DHTPIN 7
#define DHTTYPE DHT11

// LCD display instance
LiquidCrystal_I2C lcd(0x27,20,4);

// DHT instance
DHT dht(DHTPIN, DHTTYPE);

// Create aREST instance
aREST rest = aREST();

// Variables to be exposed to the API
int temperature;
int humidity;
int light;

void setup()
{
// Start Serial
  Serial.begin(115200);

// Expose variables to REST API
  rest.variable("temperature",&temperature);
  rest.variable("humidity",&humidity);
  rest.variable("light",&light);

// Set device name & ID
  rest.set_id("1");
  rest.set_name("weather_station");

// Initialize the lcd
  lcd.init();

// Print a message to the LCD.
lcd.backlight();
  lcd.setCursor(1,0);
```

```
  lcd.print("Hello !");
  lcd.setCursor(1,1);
  lcd.print("Initializing ...");

// Init DHT
  dht.begin();

// Clear LCD
  lcd.clear();
}


void loop()
{

// Measure from DHT
  temperature = (int)dht.readTemperature();
  humidity = (int)dht.readHumidity();

// Measure light level
float sensor_reading = analogRead(A0);
  light = (int)(sensor_reading/1024*100);

// Handle REST calls
  rest.handle(Serial);

// Display temperature
  lcd.setCursor(1,0);
  lcd.print("Temperature: ");
  lcd.print((int)temperature);
  lcd.print((char)223);
  lcd.print("C");

// Display humidity
  lcd.setCursor(1,1);
  lcd.print("Humidity: ");
  lcd.print(humidity);
  lcd.print("%");

// Display light level
  lcd.setCursor(1,2);
  lcd.print("Light: ");
  lcd.print(light);
  lcd.print("%");

// Wait 100 ms
  delay(100);

}
```

Premièrement, importons les bibliothèques dans le projet :

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "DHT.h"
#include <aREST.h>
```

Définissons les broches auxquelles le capteur DHT est connecté, ainsi que le type de capteur utilisé :

```
#define DHTPIN 7
#define DHTTYPE DHT11
```

Puis, il nous faut créer une instance de l'afficheur LCD :

```
LiquidCrystal_I2C lcd(0x27,20,4);
```

Créons aussi une instance du capteur DHT :

```
DHT dht(DHTPIN, DHTTYPE);
```

Une instance de la bibliothèque aREST est également nécessaire :

```
aREST rest = aREST();
```

Pour finir, nous déclarons les différentes variables qui vont enregistrer les mesures prises par la station météorologique, juste avant la fonction `setup()` :

```
int temperature;
int humidity;
int light;
```

À présent, initialisons le port série dans la fonction `setup()` du sketch :

```
Serial.begin(115200);
```

Il est important ici de commencer par un nombre d'informations transmises par seconde de 115 200 bps, étant donné que le module Bluetooth fonctionne à cette vitesse. Modifiez la vitesse du port série si vous utilisez un module Bluetooth autre que celui dont je me sers.

Rendons ensuite les différentes variables accessibles en Bluetooth *via* l'interface de programmation aREST :

```
rest.variable("temperature",&temperature);
rest.variable("humidity",&humidity);
rest.variable("light",&light);
```

Donnez maintenant un numéro d'identification (ID) à la station, et nommez-la :

```
rest.set_id("1");
rest.set_name("weather_station");
```

Pour conclure la fonction `setup()`, on initialise l'écran LCD et on lance le capteur DHT :

```
lcd.init();
dht.begin();
```

À présent, dans la fonction `loop()`, on effectue les mesures requises, on traite les requêtes provenant du module Bluetooth et enfin, on affiche les mesures sur l'écran LCD. Nous obtenons dans un premier temps les valeurs du capteur DHT :

```
temperature = (int)dht.readTemperature();
humidity = (int)dht.readHumidity();
```

On récupère également les données de la photorésistance avant de les convertir en pourcentage :

```
float sensor_reading = analogRead(A0);
light = (int)(sensor_reading/1024*100);
```

Ensuite, nous traitons les requêtes émises par le module Bluetooth :

```
rest.handle(Serial);
```

Nous terminons le sketch par l'affichage des différentes mesures sur l'afficheur LCD. J'ai choisi, par exemple, de faire apparaître la température sur la première ligne :

```
lcd.setCursor(1,0);
lcd.print("Temperature: ");
lcd.print((int)temperature);
lcd.print((char)223);
lcd.print("C");
```

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

Il est temps de tester le projet. Avant cela, apportons une légère modification à notre matériel. Ici, le port série est directement connecté au module Bluetooth, ce qui nous évite de programmer la carte Arduino *via* un câble USB. Déconnectons d'abord les broches du module Bluetooth de la carte Arduino (TX et RX).

À présent, vous pouvez transférer le sketch vers la carte Arduino. Une fois cette manipulation réalisée, vous devez reconnecter les broches TX et RX entre le module Bluetooth et le module Arduino. Après cette étape, jumelez une nouvelle fois le module Bluetooth avec votre ordinateur et sélectionnez le port série Bluetooth (qui commence par « cu. » et qui contient le nom du module).

Ouvrez ensuite le moniteur série et sélectionnez la vitesse du port que vous avez définie dans le sketch. Vérifiez que le caractère de fin de ligne est configuré sur « Carriage return », tout près de la vitesse du port série. Saisissez simplement ceci :

```
/id
```

Le projet devrait vous donner en réponse le numéro d'identification et le nom attribués à la station météorologique :

```
{"id": "1", "name": "weather_station", "connected": true}
```

On peut également effectuer une requête vers l'interface de programmation aREST et recevoir en retour la valeur d'une variable comme le niveau d'éclairage par exemple, grâce à cette commande :

```
/light
```

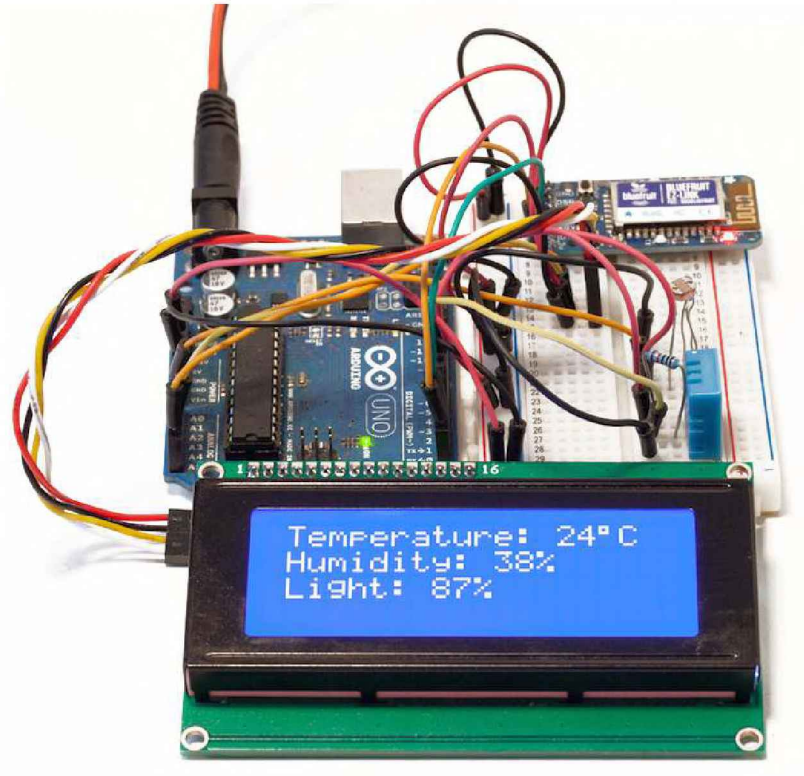
Vous devriez obtenir comme réponse :

```
{"light": 83, "id": "1", "name": "weather_station", "connected":
true}
```

Toutes les mesures du projet doivent bien entendu apparaître simultanément sur l'écran LCD.

Si vous n'obtenez pas ce résultat à ce stade, il vous faut vérifier plusieurs choses :

- Assurez-vous d'avoir correctement assemblé le montage, en suivant les instructions données précédemment dans ce chapitre.
- Vérifiez que le code de cette partie est bien chargé sur la carte et rebranchez le module Bluetooth sur les broches TX et RX de la carte Arduino.



CRÉER UNE INTERFACE

Procédons maintenant à la création de l'interface de notre station Bluetooth qui nous permettra de suivre, à partir d'un navigateur web, les mesures prises par la station. Les mesures seront toujours visibles depuis notre afficheur LCD, cependant nous pourrons y avoir accès également à distance.

Cette partie est semblable à celle du chapitre précédent consacrée à la réalisation d'une interface. Vous pouvez donc ignorer les premières explications sur le début du code si vous vous sentez à l'aise dans ce domaine.

Nous allons nous servir de Node.js pour développer cette interface ainsi que toutes celles du livre. Tout d'abord, nous allons créer le fichier principal **app.js** que nous exécuterons plus tard en utilisant la commande node dans un terminal. Celui-ci contient le cœur de la partie logicielle du projet.

Vous trouverez ci-dessous la version complète du code de ce fichier :

```
// Modules
var express = require('express');
var app = express();

// Define port
var port = 3000;

// View engine
app.set('view engine', 'jade');

// Set public folder
app.use(express.static(__dirname + '/public'));

// Rest
var rest = require("arest")(app);
rest.addDevice('serial','/dev/tty.usbmodem1a12121', 115200);

// Serve interface
app.get('/', function(req, res){
  res.render('interface');
});

// Start server
app.listen(port);
console.log("Listening on port " + port);
```

On commence par importer le module Express :

```
var express = require('express');
```

Nous créons ensuite notre application basée sur la structure du module Express, puis nous définissons la valeur du port à 3000 :

```
var app = express();
var port = 3000;
```

Il nous faut aussi indiquer au logiciel l'endroit où il trouvera l'interface graphique que nous programmerons plus loin. Enfin, nous définissons le langage de *templating* Jade en tant que moteur de vue (*view engine*) par défaut :

```
app.use(express.static(__dirname + '/public'));
app.set('view engine', 'jade');
```

Il est également nécessaire d'importer le module « node-aREST » et d'ajouter le module connecté à votre port série Bluetooth. Notez qu'il vous faut inscrire ici l'adresse de votre port série afin que l'interface puisse communiquer avec le module Bluetooth :

```
var rest = require("arest")(app);
rest.addDevice('serial','/dev/tty.AdafruitEZ-Link06d5-SPP',
115200);
```

Créons maintenant la « route » principale de notre serveur. On définit cette route en associant l'URL racine du serveur au fichier **Jade** correspondant :

```
app.get('/', function(req, res){
  res.render('interface');
});
```

Toujours dans le fichier **app.js**, on lance l'application sur le port défini précédemment, puis on inscrit un message dans la console :

```
app.listen(port);
console.log("Listening on port " + port);
```

Voilà pour ce qui est du fichier principal du serveur. Nous allons maintenant créer l'interface en elle-même. Voyons en premier lieu ce que le fichier **Jade** contient. Celui-ci se situe dans le dossier **/views** de notre projet.

Vous trouverez ci-dessous la version complète du code de ce fichier :

```
doctype
html
  head
    title Bluetooth Weather Station
    link(rel='stylesheet', href='/css/interface.css')
    link(rel='stylesheet',
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0
/css/bootstrap.min.css")
    script(src="https://code.jquery.com/jquery-2.1.1.min.js")
    script(src="/js/interface.js")
  body
    .container
      h1 Bluetooth Weather Station
      .row.voffset
        .col-md-6
          .display#temperatureDisplay Temperature:
.col-md-6
          .display#humidityDisplay Humidity:
.row
        .col-md-6
```

```
            .display#lightDisplay Light level:
.col-md-6
            .status#status Station Offline
```

Au début de ce fichier, on importe les différents fichiers JavaScript qui vont traiter les clics sur l'interface et envoyer les commandes correspondantes à la carte Arduino :

```
script(src="https://code.jquery.com/jquery-2.1.1.min.js")
script(src="/js/interface.js")
```

Nous allons utiliser la structure **bootstrap** pour embellir notre interface :

```
link(rel='stylesheet', href='/css/interface.css')
link(rel='stylesheet',
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0
/css/bootstrap.min.css")
```

La partie principale de l'interface sera constituée de plusieurs blocs affichant l'état des capteurs. Nous allons simplement faire passer la couleur de l'indicateur du gris à l'orange pour montrer qu'un capteur est actif. Voici le code pour les capteurs :

```
.row.voffset
    .col-md-6
        .display#temperatureDisplay Temperature:
.col-md-6
        .display#humidityDisplay Humidity:
.row
    .col-md-6
        .display#lightDisplay Light level:
.col-md-6
        .status#status Station Offline
```

Examinons à présent le code contenu dans le fichier interface.js qui décrit la façon dont l'interface du projet fonctionne. On y effectue des requêtes vers la carte par le biais du serveur Node.js et on met l'interface à jour en fonction. Le fichier se situe dans le dossier **public/js** de l'interface.

Vous trouverez ci-dessous la version complète du code de ce fichier :

```
var devices = [];

$.get('/devices', function( json_data ) {
  devices = json_data;
});

$(document).ready(function() {

  function updateSensors() {

    // Update light level and status
    $.get('/' + devices[0].name + '/light', function(json_data) {

      console.log(json_data.light);
```

```
        $("#lightDisplay").html("Light level: " + json_data.light +
"%");

        // Update status
        if (json_data.connected == 1){
          $("#status").html("Station Online");
          $("#status").css("color","green");
        }
        else {
          $("#status").html("Station Offline");
          $("#status").css("color","red");
        }

        $.get('/' + devices[0].name
                  + '/temperature', function(json_data) {
          $("#temperatureDisplay").html("Temperature: "
                                + json_data.temperature + "°C");

          $.get('/' + devices[0].name
                    + '/humidity', function(json_data) {
            $("#humidityDisplay").html("Humidity: "
                                + json_data.humidity + "%");
          });
        });
      });
    }

    setTimeout(updateSensors, 500);
    setInterval(updateSensors, 5000);

});
```

On signale tout d'abord le recours aux communications série, puis on annonce quel port série est utilisé ainsi que sa vitesse :

```
type = 'serial';
address = '/dev/cu.AdafruitEZ-Link06d5-SPP';
speed = 115200;
```

Remarquez qu'il est nécessaire d'inscrire l'adresse du port série de votre module Bluetooth dans la variable `address`.

La majeure partie de ce fichier JavaScript consiste à effectuer des requêtes en continu dans le but d'obtenir la valeur des différentes variables de la carte, en envoyant des requêtes *via* l'interface de programmation aREST. On effectue cela dans la fonction `setInterval()` :

```
setInterval(function() {
```

Dans cette fonction, on envoie une requête *via* la connexion Bluetooth de notre ordinateur. Par exemple, pour obtenir le niveau d'éclairage enregistré par la carte Arduino, on envoie la commande `/light`, utilisée précédemment :

```
$.get('/' + devices[0].name + '/light', function(json_data) {
```

La bibliothèque aREST nous retourne systématiquement les données en format JSON, ce qui permet de les utiliser facilement dans JavaScript. Une fois les données récupérées, on peut mettre à jour ce qui apparaît à l'écran :

```
$("#lightDisplay").html("Light level: " + json_data.light + "%");
```

La même procédure est effectuée pour les données relatives à la température et à l'humidité.

Nous voulons également savoir si la station est connectée ou non. Pour cela, mettez à jour l'affichage de l'interface en fonction de ce que vous pouvez lire sous l'attribut `connected` lorsqu'une valeur est retournée :

```
if (json_data.connected == 1){
  $("#status").html("Station Online");
  $("#status").css("color","green");
}
else {
  $("#status").html("Station Offline");
  $("#status").css("color","red");
}
```

Une fois cette étape réalisée, on ferme la fonction `setInterval()`, ce qui a pour effet de répéter la boucle toutes les 5 secondes.


Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

Testons à présent l'interface. Veillez à télécharger l'ensemble des fichiers du dépôt GitHub et à charger le code en incorporant, le cas échéant, vos propres données comme le port série correspondant à votre module Bluetooth. Vérifiez aussi que la carte est programmée avec le code donné précédemment dans ce chapitre.

Rendez-vous ensuite dans le dossier de l'interface *via* un terminal et entrez la commande suivante pour installer les modules aREST, Express et Jade :

```
sudo npm install arest jade express
```

Si vous travaillez sous Windows, vous devez laisser de côté **sudo** qui se situe devant les commandes. Il est recommandé d'utiliser le terminal installé par défaut avec Node.js.

Pour finir, vous pouvez lancer le serveur Node.js en saisissant :

```
node app.js
```

Vous devriez alors obtenir le message suivant dans le terminal :

```
Listening on port 3000
```

Ouvrez votre navigateur web et saisissez :

```
localhost:3000
```

Après quelques instants (la connexion Bluetooth 2.1 peut être assez lente), vous devriez voir apparaître l'interface de la station météorologique :

Vous pouvez voir si les valeurs affichées sur l'écran LCD correspondent à celles qui apparaissent sur l'interface.

Si vous n'obtenez pas ce résultat à ce stade, il vous faut vérifier plusieurs choses :

- Assurez-vous d'utiliser la dernière version du code à partir du dépôt GitHub du livre.
- Vérifiez que les fichiers ont bien été modifiés selon notamment les paramètres du port série de votre module Bluetooth.
- Pour finir, vous devez avoir installé les modules Node.js avec l'outil **npm** avant de lancer l'interface web.

POUR ALLER PLUS LOIN

Résumons ce que nous avons vu dans ce projet. Nous sommes partis du projet abordé dans le chapitre 6 et lui avons associé une fonction de communication sans fil. Nous avons appris à faire interagir un module Bluetooth avec Arduino et à construire une station de mesures. Nous avons pu obtenir le suivi des données de la station à partir de notre navigateur web.

Vous pouvez partir de ce projet pour réaliser une multitude de montages plus complexes. Chaque module Bluetooth ayant son propre nom (et un port série différent), vous pouvez mettre en application les connaissances acquises dans ce chapitre pour installer ces stations où vous le souhaitez.

Pour que les fonctionnalités de votre station se rapprochent de celles des systèmes vendus dans le commerce, ajoutez au projet un capteur de pression (baromètre) intégrant un circuit BMP180, par exemple. Ce capteur vous permettra non seulement de mesurer la pression atmosphérique mais aussi l'altitude à laquelle se situe la station. Si vous placez la station en extérieur, vous pouvez aussi incorporer un anémomètre à votre montage afin de mesurer la vitesse du vent.

10 Commander une lampe en WiFi

Dans ce chapitre, nous partirons du projet du chapitre 7 consacré à la création d'une lampe intelligente. Cependant, dans ce projet, nous allons remplacer l'afficheur LCD utilisé pour afficher le statut de la lampe par un module WiFi.

Il vous sera ainsi possible de piloter votre lampe à distance depuis un ordinateur, mais aussi à partir de n'importe quel appareil connecté au réseau WiFi local. L'interface que nous allons créer vous donnera accès au suivi de la consommation énergétique de la lampe et du niveau d'éclairage ambiant. Enfin, vous pourrez créer vos propres paramétrages en fonction des données reçues, et programmer par exemple l'extinction de la lampe lorsque le niveau d'éclairage atteint une valeur définie. Plongeons maintenant au cœur du projet.

MATÉRIEL ET LOGICIEL REQUIS

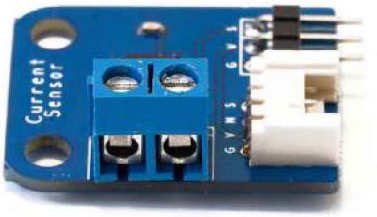
Pour ce projet, il vous faudra bien sûr une carte **Arduino Uno**. Il est possible aussi d'utiliser une carte Arduino Mega ou Leonardo.

En ce qui concerne le **module relais**, j'ai choisi un modèle 5 V de la marque Polulu qui intègre un relais à une carte. Ce module comporte tous les composants nécessaires au pilotage du relais à partir de la carte Arduino. Voici un aperçu de ce modèle :

Afin de mesurer le courant qui circule à travers la lampe, j'ai utilisé une carte s'inspirant du capteur AC712 du fabricant ITead Studio. Ce capteur s'avère très pratique dans notre cas puisqu'il renvoie une tension proportionnelle au courant mesuré. Grâce à une formule bien précise, nous pourrons estimer l'intensité du courant qui traverse la lampe en nous basant sur la tension mesurée par la carte Arduino.

Les autres cartes possédant le même capteur sont bien évidemment compatibles.

Voici un aperçu de la carte dont je me suis servi pour ce projet :



Pour mesurer le niveau d'éclairage, j'ai choisi une **photorésistance** dotée d'une résistance de 10 kΩ. Elle renverra un signal proportionnel à la quantité de lumière entrante.

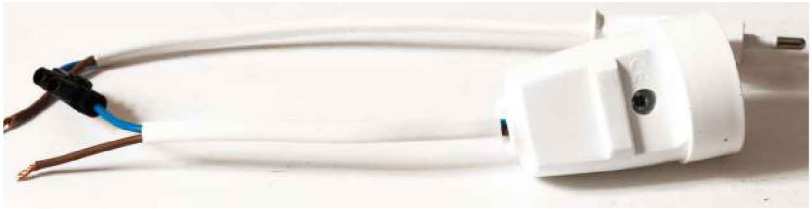
Pour la suite du projet, il vous faudra vous équiper d'un **module WiFi** semblable au modèle CC3000. Comme toujours, il n'y a pas une seule et unique façon de procéder.

Je vous conseille toutefois d'utiliser le module WiFi CC3000 qui est le seul à avoir fonctionné sans problème dans mon cas. De taille réduite, ce module dispose d'un régulateur de tension et d'une antenne intégrée.

Les tests effectués avec la carte officielle TI CC3000 n'ont pas été concluants. Celle-ci requiert l'utilisation d'un convertisseur de niveau (les deux cartes fonctionnant à des tensions différentes : 3,3 V / 5 V).

L'autre alternative serait de créer votre propre carte à l'aide de l'un des nombreux schémas de circuits imprimés disponibles sur Internet.

Pour connecter la lampe au projet, je me suis servi de deux fiches secteur munies de simples câbles et comprenant une prise femelle (où la lampe viendra se brancher) et une prise mâle (à brancher sur la prise murale). Voici un aperçu des câbles que j'ai utilisés :



Pour finir, j'ai utilisé une plaque d'essai ainsi que des fils de raccordement pour effectuer les différents branchements.

LISTE DES COMPOSANTS

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Module relais (http://snootlab.com/tinker-it/229-mod.html)
- Capteur de courant (http://www.robotshop.com/eu/fr/capteur-courant-5a-acs712-electronic-brick.html)
- Photorésistance (http://snootlab.com/composants/97-photoresistance.html)
- Résistance de 10 kΩ (http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html)

☞

☞

- Module WiFi CC3000 (http://www.gotronic.fr/art-module-wifi-cc3000-ada1469-21904.html)
- Plaque d'essai (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

Côté logiciel, installez sur votre ordinateur, si ce n'est pas déjà fait, la dernière version de Arduino IDE ainsi que la bibliothèque aREST. Ce projet nécessite aussi le téléchargement de la bibliothèque de la puce CC3000. Appropriez-vous également la bibliothèque CC3000 mDNS.

Téléchargez la bibliothèque aREST à cette adresse :
https://github.com/marcoschwartz/aREST

Ainsi que la bibliothèque de la puce CC3000 :
https://github.com/adafruit/Adafruit_CC3000_Library

Enfin téléchargez la bibliothèque CC3000 mDNS :
https://github.com/adafruit/CC3000_MDNS

Afin d'installer une bibliothèque, il vous suffit d'extraire son dossier vers **/libraries**, situé dans le dossier racine Arduino (créez ce dossier s'il n'existe pas).

ASSEMBLER LE PROJET

Passons à l'assemblage du matériel. À l'image du projet précédent impliquant un relais, nous procéderons en deux étapes. Il nous faut connecter les différents composants comme le module relais à la carte Arduino avant d'ajouter la lampe à notre montage.

L'assemblage du matériel de cette première partie est assez simple : il consiste à connecter le module relais, le capteur de courant, le module WiFi et la photorésistance. Raccordons tout d'abord la broche 5 V de la carte Arduino Uno à la rangée rouge de la plaque d'essai et la broche de masse à la rangée bleue.

Connectez la photorésistance en série avec la résistance de 10 kΩ sur la plaque d'essai. L'autre patte de la photorésistance sera raccordée à la rangée rouge de la plaque d'essai et la seconde patte de la résistance à la masse. Enfin, reliez la broche commune entre la photorésistance et la résistance à l'entrée analogique A0 de la carte Arduino.

Intéressons-nous maintenant au module WiFi.

Connectez tout d'abord la broche IRQ de la carte CC3000 à la broche n° 3 de la carte Arduino, puis la broche VBAT à la broche n° 5 et enfin la broche CS à la broche n° 10.

Connectez ensuite les broches de l'interface SPI à la carte Arduino : MOSI, MISO, et CLK doivent être branchées respectivement aux broches n° 11, 12 et 13.

En ce qui concerne les broches d'alimentation électrique : VIN doit être connectée à la broche Arduino 5 V (rangée rouge) et GND à GND (rangée bleue).

Voici le schéma de montage du projet, exempt du module relais et du capteur de courant qui sont d'ores et déjà connectés :

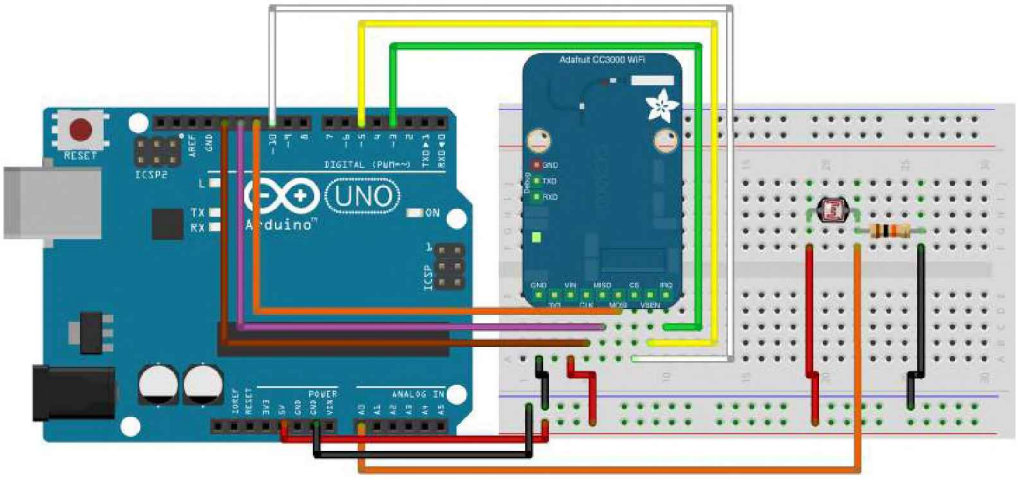
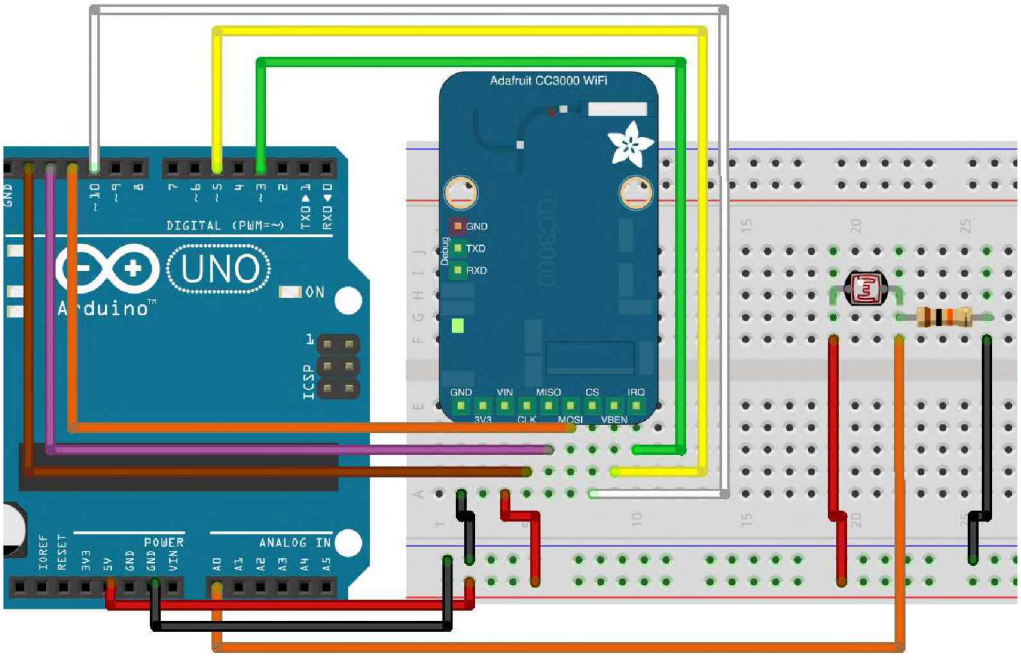


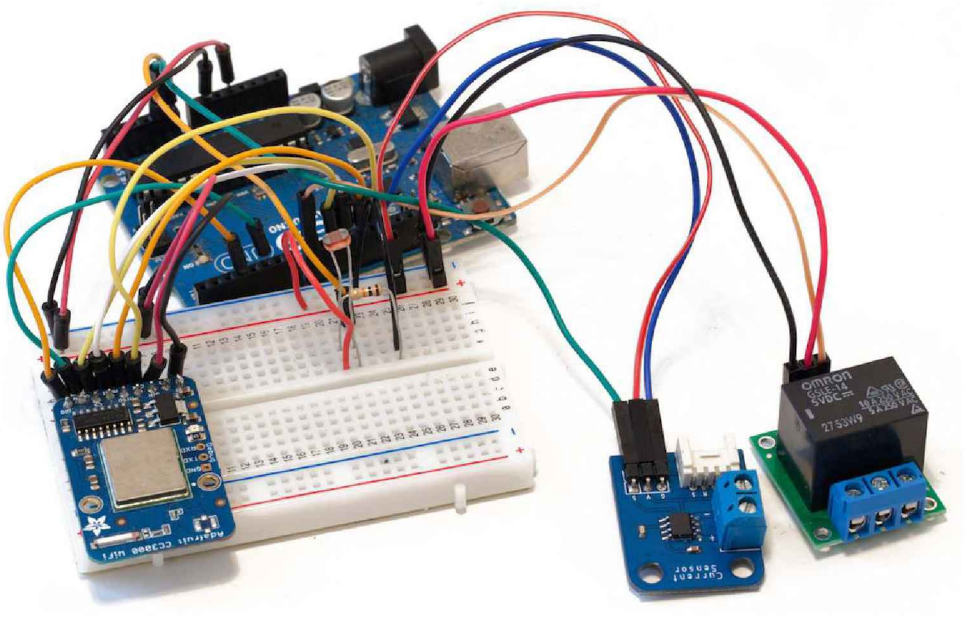
Figure 10.1 Image créée avec le logiciel Fritzing (http://fritzing.org/)

Pour le module relais, vous aurez trois broches à connecter : VCC, GND et SIG. Raccordez la broche VCC à la rangée rouge de la plaque d'essai car celle-ci doit être connectée à la broche 5 V de la carte Arduino. Raccordez la broche GND à la rangée bleue de la plaque d'essai car elle doit être connectée à la broche de masse de la carte Arduino. Pour finir, connectez la broche SIG à la broche n° 8 de la carte Arduino.

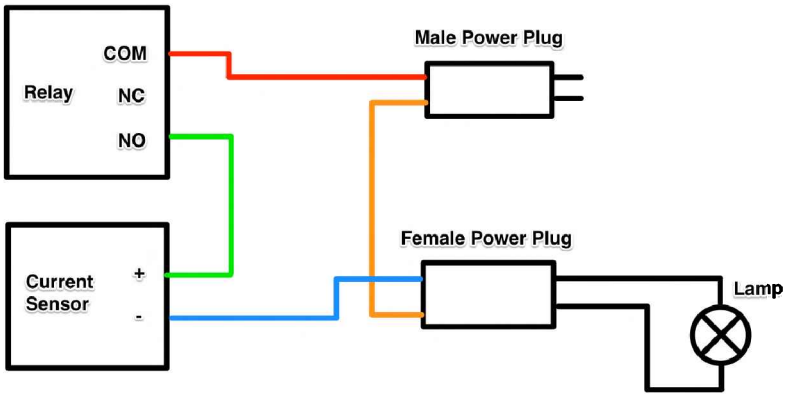


Le principe est le même pour le module de capteur de courant. Celui-ci dispose de trois broches : VCC, GND et OUT. Comme pour le relais, il vous faut brancher VCC à la rangée rouge de la plaque d'essai puisque cette broche doit être connectée à la broche 5 V de la carte Arduino. Raccordez la broche GND à la rangée bleue de la plaque d'essai car elle doit être connectée à la broche de masse de la carte Arduino. Pour finir, connectez la broche OUT à l'entrée analogique A1 de la carte Arduino.

Voici un aperçu du projet entièrement assemblé, jusqu'ici sans la lampe :



Ajoutons-la au projet. L'idée est de faire passer l'alimentation électrique principale (de la prise murale) par le relais, puis par le capteur de courant pour atteindre enfin la lampe. Le schéma suivant vous aidera à effectuer les branchements :



Puisque des tensions dangereuses sont impliquées (110 V ou 230 V suivant le continent où vous résidez), certaines précautions énoncées dans l'introduction du livre sont à prendre. Il est bien entendu possible de réaliser et de tester ce projet sans connecter d'appareils à puissance élevée.

TESTER LE MODULE WIFI

Afin de mettre au point notre lampe intelligente sans fil, nous allons premièrement nous assurer du bon fonctionnement du composant principal du projet : le module WiFi. Pour ce faire, écrivons un sketch Arduino qui initialise la puce, permet de se connecter au réseau local en WiFi et affiche l'adresse IP du module, témoignant de la réussite de la connexion.

Vous trouverez ci-dessous la version complète du code de cette partie :

```
// Import required libraries
#include <Adafruit_CC3000.h>
#include <SPI.h>

// These are the pins for the CC3000 chip
// if you are using a breakout board
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10

// Create CC3000 instance
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
  ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT, SPI_CLOCK_DIV2);

// Your WiFi SSID and password
#define WLAN_SSID       "yourSSID"
#define WLAN_PASS       "yourPassword"
#define WLAN_SECURITY   WLAN_SEC_WPA2

void setup(void)
{
// Start Serial
  Serial.begin(115200);

// Set up CC3000 and get connected to the wireless network
Serial.println("Initializing chip ...");
if (!cc3000.begin())
  {
while(1);
  }

// Connect to WiFi
  Serial.println("Connecting to WiFi network ...");
if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
while(1);
  }
while (!cc3000.checkDHCP())
```

```
  {
    delay(100);
  }
  Serial.println("Connected !");

// Display connection details
  displayConnectionDetails();
  Serial.println(F("Test completed"));
}

void loop() {

}

// Display connection details
bool displayConnectionDetails(void)
{
uint32_t ipAddress, netmask, gateway, dhcpserv, dnsserv;

if(!cc3000.getIPAddress(&ipAddress, &netmask, &gateway,
                                    &dhcpserv, &dnsserv))
  {
    Serial.println(F("Unable to retrieve the IP Address!\r\n"));
return false;
  }
else
  {
    Serial.print(F("\nIP Addr: "));
           cc3000.printIPdotsRev(ipAddress);
    Serial.print(F("\nNetmask: ")); cc3000.printIPdotsRev(netmask);
    Serial.print(F("\nGateway: ")); cc3000.printIPdotsRev(gateway);
    Serial.print(F("\nDHCPsrv: "));
           cc3000.printIPdotsRev(dhcpserv);
    Serial.print(F("\nDNSserv: ")); cc3000.printIPdotsRev(dnsserv);
    Serial.println();
return true;
  }
}
```

Premièrement, on inclut les bibliothèques de la puce CC3000 :

```
#include <Adafruit_CC3000.h>
#include <SPI.h>
```

On définit ensuite les broches auxquelles le module CC3000 est connecté :

```
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10
```

Nous pouvons ensuite créer une instance pour la puce WiFi CC3000 :

```
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT, SPI_CLOCK_DIV2);
```

À présent, vous devez modifier le sketch en inscrivant le nom de votre réseau WiFi ainsi que le mot de passe associé. Si votre réseau n'utilise pas le mode d'authentification WPA2, vous devrez également changer ce paramètre de sécurité :

```
#define WLAN_SSID       "yourSSID"
#define WLAN_PASS       "yourPassword"
#define WLAN_SECURITY   WLAN_SEC_WPA2
```

Dans la fonction `setup()` du sketch, vous devez lancer le port série que nous utiliserons à des fins de débogage :

```
Serial.begin(115200);
```

On initialise ensuite la puce WiFi CC3000 :

```
Serial.println("Initializing chip ...");
if (!cc3000.begin())
{
while(1);
}
```

Le sketch permettra normalement de connecter la puce WiFi au réseau défini précédemment et d'obtenir une adresse IP :

```
Serial.println("Connecting to WiFi network ...");
if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
while(1);
}
while (!cc3000.checkDHCP())
{
  delay(100);
}
Serial.println("Connected !");
```

Une fois la puce connectée au réseau, nous recevons l'adresse IP du module WiFi que nous ferons apparaître également sur le port série.

Pour finir, entrons les données de fin de test :

```
displayConnectionDetails();
Serial.println(F("Test completed"));
```

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

Il est temps à présent de tester le premier sketch de ce projet. Assurez-vous de l'avoir modifié en fonction du nom de votre réseau WiFi et du mot de passe correspondant. Transférez ensuite le sketch vers la carte Arduino et ouvrez le port série.

Vous devriez voir apparaître ceci :

```
Initializing chip ...
Connecting to WiFi network ...
Connected !

IP Addr: 192.168.1.100
Netmask: 255.255.255.0
Gateway: 192.168.1.1
DHCPsrv: 192.168.1.1
DNSserv: 192.168.0.254
Test completed
```

Si vous voyez apparaître une adresse IP, votre module WiFi est correctement raccordé et apte à se connecter à votre réseau WiFi. Intéressons-nous désormais au code de la partie dédiée au pilotage de la lampe intelligente.

Si vous n'obtenez pas ce résultat à ce stade, il vous faut vérifier plusieurs choses :

- Vérifiez premièrement que le module WiFi CC3000 est convenablement branché. Le montage implique une multitude de branchements, vous pouvez donc les avoir intervertis.
- Assurez-vous également que votre connexion Internet est opérationnelle. Dans le cas contraire, la puce WiFi ne sera pas en mesure de se connecter au site web de test.

COMMANDER LA LAMPE À DISTANCE

Commence maintenant l'écriture du sketch Arduino permettant de piloter la lampe *via* le WiFi, de connaître sa consommation énergétique ainsi que le niveau d'éclairage ambiant.

Pour mettre cela en pratique, la bibliothèque aREST va encore une fois nous être utile afin d'avoir accès facilement aux broches de la carte Arduino et aux variables qui contiennent les mesures.

Le sketch de ce projet se base sur celui consacré au test des connexions WiFi. De ce fait, je détaillerai seulement la partie complémentaire du sketch.

Vous trouverez ci-dessous la version complète du code de cette partie :

```
#define NUMBER_VARIABLES 2
#define NUMBER_FUNCTIONS 1

// Import required libraries
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <CC3000_MDNS.h>
#include <aREST.h>

// Relay state
constint relay_pin = 8;

// Define measurement variables
float amplitude_current;
```

```
float effective_value;
float effective_voltage = 230;
     // Set voltage to 230V (Europe) or 110V (US)
float effective_power;
float zero_sensor;

// These are the pins for the CC3000 chip
// if you are using a breakout board
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10

// Create CC3000 instance
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT);

// Create aREST instance
aREST rest = aREST();

// Your WiFi SSID and password
#define WLAN_SSID       "KrakowskiePrzedm51m.15(flat15)"
#define WLAN_PASS       "KrK51flat15_1944_15"
#define WLAN_SECURITY   WLAN_SEC_WPA2

// The port to listen for incoming TCP connections
#define LISTEN_PORT           80

// Server instance
Adafruit_CC3000_Server restServer(LISTEN_PORT);

// DNS responder instance
MDNSResponder mdns;

// Variables to be exposed to the API
int power;
int light;

void setup(void)
{
// Start Serial
  Serial.begin(115200);

// Init variables and expose them to REST API
  rest.variable("light",&light);
  rest.variable("power",&power);

// Set relay & led pins to outputs
  pinMode(relay_pin,OUTPUT);

// Calibrate sensor with null current
  zero_sensor = getSensorValue(A1);
```

```
// Give name and ID to device
  rest.set_id("001");
  rest.set_name("smart_lamp");

// Set up CC3000 and get connected to the wireless network.
if (!cc3000.begin())
  {
while(1);
  }

if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
while(1);
  }
while (!cc3000.checkDHCP())
  {
    delay(100);
  }

// Start multicast DNS responder
if (!mdns.begin("arduino", cc3000)) {
while(1);
  }

// Display connection details
  displayConnectionDetails();

// Start server
  restServer.begin();
  Serial.println(F("Listening for connections..."));
}

void loop() {

// Measure light level
float sensor_reading = analogRead(A0);
  light = (int)(sensor_reading/1024*100);

// Perform power measurement
float sensor_value = getSensorValue(A1);

// Convert to current
  amplitude_crurent
= (float)(sensor_value-zero_sensor)/1024*5/185*1000000;
  effective_value = amplitude_current/1.414;
  effective_power = abs(effective_value*effective_voltage/1000);
  power = (int)effective_power;

// Handle any multicast DNS requests
  mdns.update();

// Handle REST calls
  Adafruit_CC3000_ClientRef client = restServer.available();
```

```
  rest.handle(client);

}

bool displayConnectionDetails(void)
{
uint32_t ipAddress, netmask, gateway, dhcpserv, dnsserv;

if(!cc3000.getIPAddress(&ipAddress, &netmask, &gateway, &dhcpserv,
                        &dnsserv))
  {
    Serial.println(F("Unable to retrieve the IP Address!\r\n"));
return false;
  }
else
  {
    Serial.print(F("\nIP Addr: "));
                          cc3000.printIPdotsRev(ipAddress);
    Serial.print(F("\nNetmask: ")); cc3000.printIPdotsRev(netmask);
    Serial.print(F("\nGateway: ")); cc3000.printIPdotsRev(gateway);
    Serial.print(F("\nDHCPsrv: "));
                           cc3000.printIPdotsRev(dhcpserv);
    Serial.print(F("\nDNSserv: ")); cc3000.printIPdotsRev(dnsserv);
    Serial.println();
return true;
  }
}

// Get the reading from the current sensor
float getSensorValue(int pin)
{
int sensorValue;
float avgSensor = 0;
int nb_measurements = 100;
for (int i = 0; i < nb_measurements; i++) {
    sensorValue = analogRead(pin);
    avgSensor = avgSensor + float(sensorValue);
  }
  avgSensor = avgSensor/float(nb_measurements);
return avgSensor;
}
```

On inclut d'abord les bibliothèques requises.

```
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <CC3000_MDNS.h>
#include <aREST.h>
```

On continue en déclarant la broche à laquelle le relais est connecté :

```
constint relay_pin = 8;
```

Déclarons ensuite les différentes variables nécessaires à la partie consacrée au relevé de la consommation énergétique :

```
float amplitude_current;
float effective_value;
float effective_voltage = 230;
                    // Set voltage to 230V (Europe) or 110V (US)
float effective_power;
float zero_sensor;
```

Créons à présent une instance de l'objet aREST qui nous servira à traiter les requêtes provenant de la connexion WiFi :

```
aREST rest = aREST();
```

Définissons aussi le port sur lequel nous voulons accepter les requêtes *via* WiFi. Afin de simplifier les choses, nous allons prendre le port 80 qui permettra de commander directement notre carte Arduino à partir d'un navigateur web :

```
#define LISTEN_PORT  80
```

Il nous faut aussi créer une instance du serveur CC3000 :

```
Adafruit_CC3000_Server restServer(LISTEN_PORT);
```

Créons une autre instance pour le serveur MDNS afin de pouvoir accéder à la carte Arduino sans devoir saisir son adresse IP :

```
MDNSResponder mdns;
```

Ensuite, il ne reste plus qu'à déclarer deux variables qui contiendront les données de consommation énergétique et de niveau d'éclairage :

```
int power;
int light;
```

À présent, dans la fonction `setup()` du sketch, on rend les deux variables accessibles de l'extérieur en WiFi *via* l'interface de programmation REST :

```
rest.variable("light",&light);
rest.variable("power",&power);
```

Après cette étape, nous déclarons la broche du relais en tant que sortie :

```
pinMode(relay_pin,OUTPUT);
```

Utilisons une fonction qui se chargera de nous donner une moyenne des valeurs relevées par le capteur de courant. On obtiendra la mesure à laquelle le capteur indique une valeur de courant nulle :

```
zero_sensor = getSensorValue(A1);
```

Nous pouvons également donner un nom et un numéro d'identification à la carte. Ceux-ci nous seront retournés à chaque requête reçue par la carte *via* l'interface de programmation aREST :

```
rest.set_id("001");
rest.set_name("smart_lamp");
```

Nommons désormais la carte Arduino sur le réseau, de cette façon par exemple : `arduino`. Elle apparaîtra sur votre réseau sous le nom **arduino.local** :

```
if (!mdns.begin("arduino", cc3000)) {
while(1);
}
```

Pour finir, toujours dans la fonction `setup()`, nous lançons le serveur CC3000 et attendons des connexions entrantes :

```
restServer.begin();
Serial.println(F("Listening for connections..."));
```

Dans la fonction `loop()` du sketch, on récupère d'abord la valeur du niveau d'éclairage, exprimée en pourcentage :

```
float sensor_reading = analogRead(A0);
light = (int)(sensor_reading/1024*100);
```

On obtient cette fois-ci les mesures du capteur de courant, toujours par le biais d'une fonction qui calcule la moyenne des résultats sur plusieurs relevés de mesures :

```
float sensor_value = getSensorValue(A1);
```

On continue avec le calcul de la puissance du courant mesuré :

```
amplitude_current =
(float)(sensor_value-zero_sensor)/1024*5/185*1000000;
effective_value = amplitude_current/1.414;
effective_power = abs(effective_value*effective_voltage/1000);
power = (int)effective_power;
```

Mettons à jour le serveur MDNS :

```
mdns.update();
```

Traitons alors n'importe quelle connexion entrante à l'aide de la bibliothèque aREST :

```
Adafruit_CC3000_ClientRef client = restServer.available();
rest.handle(client);
```

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

Il est temps à présent de tester ce sketch sur notre projet. Téléchargez le code à partir du dépôt GitHub et assurez-vous de bien avoir modifié le nom et le mot de passe du réseau WiFi sur lequel la puce WiFi va se connecter. Chargez le code sur la carte Arduino et ouvrez le moniteur série. Vous devriez voir apparaître ceci :

```
Listening for connections...
```

À présent, fermez le moniteur série et ouvrez votre navigateur web. Il est maintenant possible d'effectuer des requêtes directes vers l'interface de programmation REST fonctionnant sur la carte afin d'avoir le contrôle sur les broches de la carte Arduino. Pour allumer la lampe par exemple, saisissez ceci :

```
http://arduino.local/digital/8/1
```

Vous devriez entendre le relais s'actionner, voir la lampe s'allumer et un message de confirmation s'afficher dans votre navigateur web :

```
Pin D8 set to 1
```

Pour éteindre la lampe, saisissez :

```
http://arduino.local/digital/8/0
```

Notez que ces commandes fonctionneront à partir de n'importe quel appareil connecté au même réseau local que la carte Arduino. Utilisez par exemple votre Smartphone pour piloter votre carte Arduino avec les mêmes commandes.

En cas d'échec, utilisez en premier lieu l'adresse IP de la carte au lieu d'utiliser l'URL arduino.local dans le code. L'adresse IP se trouve dans les messages affichés par le moniteur série au démarrage du projet.

Si vous n'obtenez pas ce résultat à ce stade, il vous faut vérifier plusieurs choses :

- Assurez-vous d'avoir correctement branché les composants du projet d'après les instructions données précédemment dans ce chapitre.
- Vérifiez également que les bibliothèques requises pour le projet sont bien installées, notamment la bibliothèque aREST.

METTRE EN PLACE UNE INTERFACE POUR LA LAMPE INTELLIGENTE

Vous allez maintenant créer l'interface qui vous permettra de piloter le projet à partir d'un ordinateur. Grâce à elle, non seulement vous commanderez la lampe en WiFi mais vous pourrez aussi apprécier en temps réel sa consommation énergétique ainsi que le niveau d'éclairage ambiant obtenu grâce à la photorésistance.

Cette partie est semblable à celle du chapitre précédent dédiée à la réalisation d'une interface. Vous pouvez donc ignorer les explications relatives au début du code si vous vous sentez à l'aise dans ce domaine.

Nous allons nous servir de Node.js pour développer cette interface, ainsi que toutes celles du livre. Pour commencer, nous allons créer le fichier principal **app.js** que nous exécuterons plus tard en utilisant la commande node dans un terminal. Celui-ci contient le cœur de la partie logicielle du projet.

Voici la version complète du code de ce fichier :

```
// Module
var express = require('express');
var app = express();

// Define port
var port = 3000;
```

```
// View engine
app.set('view engine', 'jade');

// Set public folder
app.use(express.static(__dirname + '/public'));

// Rest
var rest = require("arest")(app);
rest.addDevice('http','192.168.1.103');

// Serve interface
app.get('/', function(req, res){
  res.render('interface');
});

// Start server
app.listen(port);
console.log("Listening on port " + port);
```

Commençons par l'importation du module `express` :

```
var express = require('express');
```

Créons maintenant notre application à partir de la structure `express` et définissons le port à 3000 :

```
var app = express();
var port = 3000;
```

Indiquons également à notre logiciel où trouver les fichiers de l'interface graphique dans lesquels nous inscrirons du code ; définissons également Jade en tant que moteur de vue (*view engine*) par défaut :

```
app.use(express.static(__dirname + '/public'));
app.set('view engine', 'jade');
```

Le recours au module node-aREST est également indispensable si l'on veut pouvoir gérer toutes les communications entre l'interface et notre lampe intelligente. À ce stade, vous aurez également besoin d'entrer l'adresse IP de votre puce WiFi :

```
var rest = require("arest")(app);
rest.addDevice('http','192.168.1.103');
```

Créons à présent la « route » principale de notre serveur. Définissons cette route en associant l'URL racine du serveur à l'interface du projet :

```
app.get('/', function(req, res){
  res.render('interface');
});
```

Pour finir avec le fichier **app.js**, on lance l'application sur le port défini plus tôt et on inscrit un message dans la console :

```
app.listen(port);
console.log("Listening on port " + port);
```

Voilà pour ce qui est du fichier principal du serveur. Nous allons maintenant créer l'interface. Voyons en premier lieu ce que le fichier Jade contient. Ce fichier se situe dans le dossier **/views** de notre projet.

Voici la version complète du code de ce fichier :

```
doctype
html
  head
    title Smart Lamp
    link(rel='stylesheet', href='/css/interface.css')
    link(rel='stylesheet',
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0
                                    /css/bootstrap.min.css")
    script(src="https://code.jquery.com/jquery-2.1.1.min.js")
    script(src="/js/interface.js")
  body
    .container
      h1 Smart Lamp
      .row.voffset
        .col-md-6
          button.btn.btn-block.btn-lg.btn-primary#1 On
        .col-md-6
          button.btn.btn-block.btn-lg.btn-danger#2 Off
      .row
        .col-md-4
          h3#powerDisplay Power:
.col-md-4
          h3#lightDisplay Light level:
.col-md-4
          h3#status Offline
```

On commence par importer les différents fichiers JavaScript qui vont traiter les clics sur l'interface et envoyer les commandes correspondantes à la carte Arduino :

```
script(src="https://code.jquery.com/jquery-2.1.1.min.js")
script(src="/js/interface.js")
```

Utilisons la structure Twitter Bootstrap afin d'embellir notre interface :

```
link(rel='stylesheet', href='/css/interface.css')
link(rel='stylesheet',
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/
css/bootstrap.min.css")
```

Nous allons maintenant créer deux boutons : l'un permettant d'activer le relais et d'allumer ainsi la lumière et le second pour éteindre cette dernière.

Voici le code pour ces deux boutons :

```
.col-md-6
    button.btn.btn-block.btn-lg.btn-primary#1 On
.col-md-6
    button.btn.btn-block.btn-lg.btn-danger#2 Off
```

Nous définirons dans le fichier JavaScript de ce projet la manière dont les clics sur ces deux boutons seront traités.

Terminons en créant des zones de texte faisant apparaître les différentes mesures données par notre projet Arduino, comme la puissance et le niveau d'éclairage. Ajoutons aussi un indicateur permettant de vérifier que le projet est bien connecté :

```
.col-md-4
    h3#powerDisplay Power:
.col-md-4
    h3#lightDisplay Light level:
.col-md-4
    h3#status Offline
```

Examinons à présent le contenu du fichier **interface.js** qui se trouve dans le dossier **public/js** du projet.

Voici la version complète du code pour ce fichier :

```
var devices = [];

$.get('/devices', function( json_data ) {
  devices = json_data;
});

$(document).ready(function() {

  // Update sensors and repeat every 5 seconds
  setTimeout(updateSensors, 500);
  setInterval(updateSensors, 5000);

  // Function to control the lamp
  $('#1').click(function(){
    $.get('/' + devices[0].name + '/digital/8/1');
  });

  $('#2').click(function(){
    $.get('/' + devices[0].name + '/digital/8/0');
  });

  function updateSensors(){

    // Update light level
    $.get('/' + devices[0].name + '/light', function(json_data) {

      $("#lightDisplay").html("Light level: " + json_data.light
                                              + " %");
```

```
        // Update status
        if (json_data.connected == 1){
          $("#status").html("Lamp Online");
          $("#status").css("color","green");
        }
        else {
          $("#status").html("Lamp Offline");
          $("#status").css("color","red");
        }

        // Update power
        $.get('/' + devices[0].name + '/power', function(json_data) {
          $("#powerDisplay").html("Power: " + json_data.power
                                             + " W");
        });

    });
}

});
```

Ce fichier se compose de deux grandes parties : la première définit la manière dont les clics sur les boutons sont gérés et la seconde est consacrée à la réactualisation des capteurs.

Étudions la première partie. L'idée est d'associer chacun des boutons à la commande `digitalWrite()` correspondante de la carte Arduino. Pour cela, saisissez les lignes de code suivantes :

```
$('#1').click(function(){
    $.get('/' + devices[0].name + '/digital/8/1');
});

$('#2').click(function(){
    $.get('/' + devices[0].name + '/digital/8/0');
});
```

Il nous faut également programmer la partie destinée à mettre à jour l'interface avec les prises de mesures effectuées par la carte. Pour cela, nous allons effectuer des requêtes à intervalles réguliers pour obtenir ces mesures à partir de la carte, par le biais de la fonction JavaScript `setInterval` :

```
setInterval(updateSensors, 5000);
```

Étudions maintenant en détail cette fonction `updateSensors`. Dans le but de récupérer des données telles que le niveau d'éclairage à partir de la carte, on se sert de la fonction vue précédemment, en envoyant la commande `/light`. Cette fois-ci cependant, nous utilisons les données JSON qui nous ont été retournées :

```
$.get('/' + devices[0].name + '/light', function(json_data) {
```

On met à jour l'affichage en fonction de ces données :

```
$("#lightDisplay").html("Light level: " + json_data.light + " %");
```

Ces données JSON contiennent aussi des informations sur le statut de la carte. Si l'attribut `connected` apparaît, on définit alors le statut de l'indicateur sur `Online` et la couleur sur `green`. Si ces données n'apparaissent pas ou sont corrompues, on définit alors le statut sur `Offline` et on applique la couleur rouge à cet indicateur :

```
if (json_data.connected == 1){
  $("#status").html("Lamp Online");
  $("#status").css("color","green");
}
else {
  $("#status").html("Lamp Offline");
  $("#status").css("color","red");
}
```

On effectue la même opération avec les valeurs de puissance électrique sur la carte :

```
$.get('/' + devices[0].name + '/power', function(json_data) {
    $("#powerDisplay").html("Power: " + json_data.power + " W");
});
```

Cette fonction est répétée toutes les 5 secondes. Suivant ces mesures, vous pouvez définir ici vos propres fonctions pour piloter votre lampe. Vous pouvez par exemple programmer l'extinction automatique de la lampe si le niveau d'éclairage atteint une valeur déterminée.


Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

Testons à présent l'interface. Assurez-vous de télécharger l'ensemble des fichiers contenus dans le dépôt GitHub et de mettre à jour si nécessaire le code à l'aide de vos propres données, telles que l'adresse de la carte Arduino. Vérifiez aussi que la carte est programmée avec le code donné précédemment dans ce chapitre.

Puis, rendez-vous dans le dossier de l'interface à l'aide d'un terminal et entrez la commande suivante pour installer les modules aREST, Express et Jade :

```
sudo npm install arest express jade
```

Si vous utilisez Windows, vous devez laisser de côté **sudo** qui se situe devant les commandes. Il est recommandé d'utiliser l'application Terminal installée par défaut avec Node.js. Pour finir, vous pouvez lancer le serveur Node.js en saisissant :

```
sudo node app.js
```

Vous devriez alors obtenir le message suivant dans le terminal :

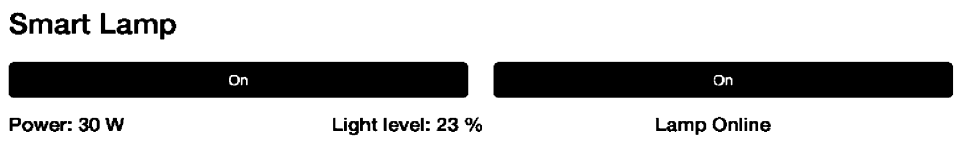
```
Listening on port 3000
```

Ouvrez votre navigateur web et saisissez :

```
localhost:3000
```

L'interface devrait désormais apparaître au sein de votre navigateur web, accompagnée des deux boutons pour le pilotage de la lampe. Lorsque vous ouvrez l'interface

pour la première fois, la lampe apparaît hors ligne et l'indicateur n'affiche aucune donnée. Ceci est tout à fait normal, ne vous inquiétez pas. Après un certain laps de temps, l'interface effectuera une requête vers la carte Arduino et mettra à jour les données en conséquence :



Testez les boutons de l'interface. La lampe est éteinte par défaut. Cliquez alors sur le bouton « On » pour l'allumer instantanément. Le relais devrait également émettre un clic audible. Lorsque la lampe est allumée, vous devriez également constater une évolution de la consommation énergétique sur l'interface. Cliquez ensuite sur le bouton « Off » afin de rééteindre la lampe.

Si vous avez défini du code à l'intérieur du fichier JavaScript de l'interface, visant par exemple à éteindre la lampe automatiquement à un niveau d'éclairage donné, vous devriez déjà voir quelque chose apparaître.

Si vous n'obtenez pas ce résultat à ce stade, il vous faut vérifier plusieurs choses :

- Assurez-vous d'utiliser la dernière version du code à partir du dépôt GitHub du livre.
- Vérifiez aussi que les fichiers ont bien été modifiés en fonction de vos propres paramètres (en indiquant bien l'adresse de votre module WiFi par exemple).
- Veillez à télécharger les modules Node.js demandés avec l'outil **npm** avant de lancer votre navigateur web.

Pour aller plus loin

Résumons ce que nous avons vu dans ce projet. Nous sommes partis du projet abordé au chapitre 7 et nous lui avons associé une fonction WiFi. Vous savez désormais comment faire interagir une carte WiFi avec Arduino et contrôler le projet depuis un navigateur web. Nous avons réussi à manœuvrer à distance notre lampe et à accéder aux différentes mesures à partir d'un navigateur web. Pour finir, nous avons créé un logiciel exécutable sur ordinateur donnant la possibilité d'avoir le contrôle sur l'ensemble du projet à partir d'une interface graphique contenue dans un navigateur web.

Vous pouvez, si vous le souhaitez, ajouter d'autres capteurs à la carte Arduino et tenter de récupérer leurs données à distance. Complétez par exemple le projet avec un capteur de température et faites apparaître ses données sur l'interface graphique. L'ajout d'une seconde lampe est tout à fait possible. Pilotez-les ensuite de manière indépendante.

En modifiant le code en cours d'exécution sur votre ordinateur dans le fichier JavaScript, vous pourrez définir des comportements de la lampe plus avancés. Vous pouvez agir par rapport au niveau d'éclairage relevé mais aussi stocker par exemple la consommation électrique de votre lampe sur le cloud. Vous pouvez tout à fait commander l'extinction de la lampe à partir d'une certaine heure de la journée. Programmez-la de façon à ce qu'elle s'allume le matin et vous serve ainsi de réveil.

Entourez-vous de plusieurs de ces projets en donnant des noms différents à vos cartes Arduino et en ajoutant d'autres éléments à l'interface graphique. D'autres comportements ou boutons peuvent être créés. Vous pouvez par exemple éteindre toutes les lampes de votre maison en un simple clic.

11 Construire un tableau de bord

Dans ce chapitre, nous allons mettre en application ce que nous avons appris jusqu'ici afin de mettre au point une petite installation domotique intégrée.

Ce système contiendra des détecteurs de mouvement sans fil XBee ainsi qu'une lampe pilotée en WiFi.

Tous ces composants seront commandés à partir d'une interface centrale.

Dans un premier temps, nous listerons les éléments nécessaires à cette installation.

Puis, nous mettrons en place les différents modules.

Après les avoir testés brièvement, nous mettrons au point une interface vous permettant de suivre tout ce qui se passe à partir de votre ordinateur.

Pour finir, je vous donnerai quelques idées pour vous permettre d'aller plus loin et de réaliser des installations domotiques plus avancées.

MATÉRIEL ET LOGICIEL REQUIS

Dressons tout d'abord une liste du matériel dont nous aurons besoin dans ce chapitre.

Nous allons en fait mettre au point un certain nombre de détecteurs de mouvement sans fil XBee et un contrôleur de lampe WiFi.

LISTE DES COMPOSANTS

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Détecteur de mouvement PIR (http://snootlab.com/adafruit/285-capteur-de-presence-pir.html)
- Carte d'extension XBee pour Arduino (http://ultra-lab.net/fr/tienda/shield-xbee)
- Module XBee Series 2 avec antenne (http://www.robotshop.com/eu/fr/module-xbee-2mw-antenne-serie-2-zibbee-mesh.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

☞

☞

Afin d'utiliser XBee sur votre ordinateur, il vous faudra ces composants :

- Carte XBee Explorer USB (http://www.gotronic.fr/art-module-xbee-explorer-usb-20253.htm)
- Module XBee Series 2 avec antenne (http://www.robotshop.com/eu/fr/module-xbee-2mw-antenne-serie-2-zibbee-mesh.html)

Pour le contrôleur de lampe WiFi, vous aurez besoin des composants suivants :

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Module relais (http://snootlab.com/tinker-it/229-mod.html)
- Capteur de courant (http://www.robotshop.com/eu/fr/capteur-courant-5a-acs712-electronic-brick.html)
- Photorésistance (http://snootlab.com/composants/97-photoresistance.html)
- Résistance de 10 kΩ (http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html)
- Module WiFi CC3000 (http://www.gotronic.fr/art-module-wifi-cc3000-ada1469-21904.html)
- Plaque d'essai (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

Côté logiciel, installez sur votre ordinateur, si ce n'est pas déjà fait, la dernière version de Arduino IDE ainsi que la bibliothèque aREST. Ce projet nécessite aussi le téléchargement de la bibliothèque de la puce CC3000. Appropriez-vous également la bibliothèque CC3000 mDNS.

Téléchargez la bibliothèque aREST à cette adresse :
https://github.com/marcoschwartz/aREST

Ainsi que la bibliothèque de la puce CC3000 :
https://github.com/adafruit/Adafruit_CC3000_Library

Enfin téléchargez la bibliothèque CC3000 mDNS :
https://github.com/adafruit/CC3000_MDNS

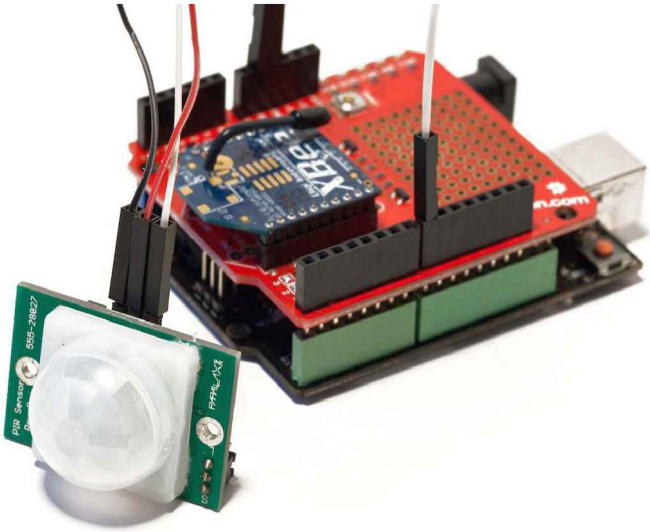
Afin d'installer une bibliothèque, il vous suffit d'extraire son dossier vers **/libraries**, situé dans le dossier racine Arduino (créez ce dossier s'il n'existe pas).

ASSEMBLER LE PROJET

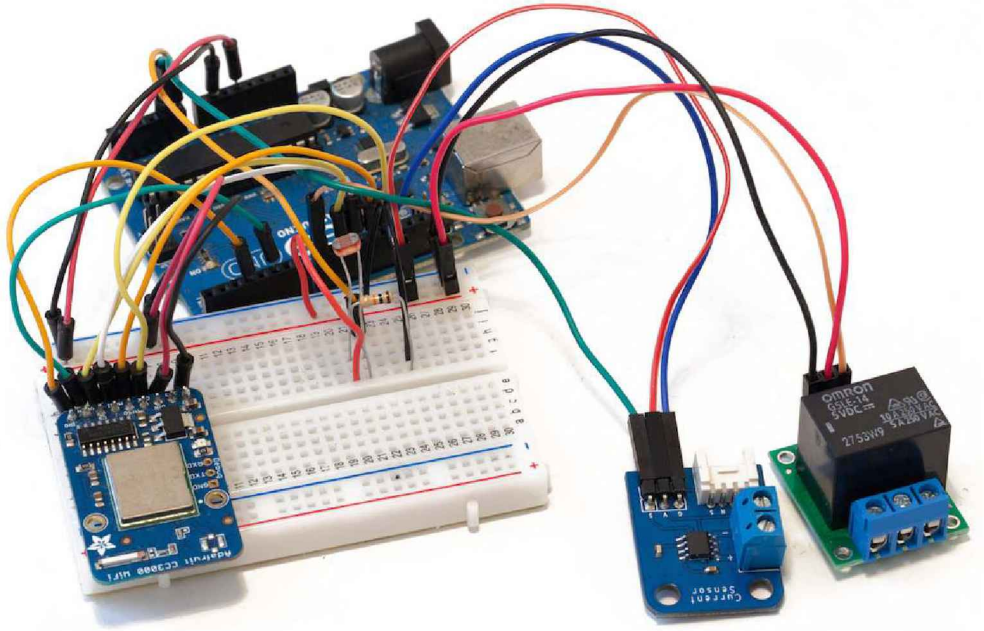
On poursuit par l'assemblage des différents modules de notre installation domotique.

La méthode d'assemblage des différents projets ayant été abordée dans les chapitres précédents, je vous invite donc à faire un retour en arrière pour l'assemblage du matériel.

En ce qui concerne les détecteurs de mouvement XBee, veuillez vous référer au chapitre 8. Vous devriez obtenir ce résultat :



En ce qui concerne le contrôleur de lampe WiFi, veuillez vous référer au chapitre 10. Voici ce que vous devriez obtenir :



TESTER DES MODULES

Testons à présent l'un des détecteurs de mouvement sans fil XBee et le contrôleur de lampe WiFi. Puisque nous avons vu précédemment comment ces modules fonctionnaient, nous allons dès maintenant les tester sans fil afin de vérifier qu'il n'y a pas de problèmes.

Commençons par le code du détecteur de mouvement sans fil XBee :

```
// Libraries
#include <SPI.h>
#include <aREST.h>

// Motion sensor ID
String xbee_id = "1";

// Create ArduREST instance
aREST rest = aREST();

void setup() {

  // Start Serial
  Serial.begin(9600);

  // Give name and ID to device
  rest.set_id(xbee_id);
  rest.set_name("motion_sensor");
}

void loop() {

  // Handle REST calls
  rest.handle(Serial);
}
```

Dans le sketch, on commence par inclure les bibliothèques correspondantes :

```
#include <SPI.h>
#include <aREST.h>
```

Définissons également le numéro d'identification du capteur. Si vous utilisez plusieurs détecteurs de mouvement chez vous, il est primordial de leur donner des numéros d'identification différents :

```
String xbee_id = "1";
```

Il nous faut aussi créer une instance de la bibliothèque aREST :

```
aREST rest = aREST();
```

À présent, initialisons le port série dans la fonction `setup()` du sketch. Notez qu'il est essentiel ici de définir la vitesse à 9600, étant donné que les modules XBee sont réglés par défaut à cette vitesse :

```
Serial.begin(9600);
```

Donnons aussi un numéro d'identification au détecteur de mouvement sans fil XBee défini plus tôt :

```
rest.set_id(xbee_id);
```

Pour finir, dans la fonction `loop()` du sketch, nous traitons simplement les requêtes venant du port série à l'aide de la bibliothèque aREST :

```
rest.handle(Serial);
```

Il est temps à présent de tester ce sketch. Chargez-le sur la carte Arduino et mettez l'interrupteur de la carte d'extension XBee sur la position « UART » afin de faire interagir le module XBee directement avec le microcontrôleur de la carte Arduino *via* le port série.

> Si vous devez reprogrammer la carte Arduino, il est indispensable de repasser en position **DLINE**.

Il est nécessaire à présent de localiser le port série associé à la carte XBee Explorer connectée à l'ordinateur. Effectuez ceci depuis le menu Outils > Port Série du logiciel Arduino IDE. Le nom du port devrait se présenter sous cette forme : **/dev/cu.usbserial-A702LF8B**. Je vous conseille de le noter puisque vous en aurez besoin lors de la création de l'interface de vos détecteurs de mouvement.

Ouvrez à présent le moniteur série du logiciel Arduino IDE. Assurez-vous que la vitesse du port série est réglée à 9600. Notez que nous sommes à présent connectés à la carte XBee Explorer USB, ce qui signifie que toutes les commandes envoyées parviennent à l'ensemble des modules XBee de votre domicile.

Saisissez dans le moniteur série :

```
/id
```

Cela effectuera une requête de numéro d'identification auprès des modules XBee détectés chez vous. Lorsque j'ai effectué le test, je ne disposais que d'un seul module chez moi. Voici la réponse reçue :

```
{"id": "1", "name": "", "connected": true}
```

Continuons avec la lecture du statut du détecteur de mouvement. N'oublions pas que celui-ci est connecté à la broche n° 8. Pour lire à partir de cette broche, entrez :

```
/digital/8
```

Le détecteur devrait répondre par ce message :

```
{"return_value": 1, "id": "1", "name": "", "connected": true}
```

Si jusqu'ici le détecteur répond aux requêtes, vous pouvez en conclure qu'il fonctionne convenablement et que vous arrivez à y accéder sans fil. Assurez-vous également de configurer vos modules XBee de sorte qu'ils se trouvent tous dans le même PAN ID XBee. Consultez le chapitre 8 pour obtenir plus d'informations à ce sujet.

Testons à présent le contrôleur de lampe WiFi. Voici le code pour ce module :

```
// Import required libraries
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <CC3000_MDNS.h>
#include <aREST.h>

// These are the pins for the CC3000 chip
// if you are using a breakout board
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10

// Create CC3000 instance
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
  ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT,SPI_CLOCK_DIV2);

// Create ArduREST instance
aREST rest = aREST();

// Your WiFi SSID and password
#define WLAN_SSID       "yourSSID"
#define WLAN_PASS       "yourPassword"
#define WLAN_SECURITY   WLAN_SEC_WPA2

// The port to listen for incoming TCP connections
#define LISTEN_PORT           80

// Server instance
Adafruit_CC3000_Server restServer(LISTEN_PORT);

// DNS responder instance
MDNSResponder mdns;

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Give name and ID to device
  rest.set_id("2");
  rest.set_name("relay_module");

  // Set up CC3000 and get connected to the wireless network.
if (!cc3000.begin())
  {
    while(1);
  }

  if (!cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
    while(1);
  }
  while (!cc3000.checkDHCP())
```

```
  {
    delay(100);
  }

  // Start multicast DNS responder
  if (!mdns.begin("arduino", cc3000)) {
    while(1);
  }

  // Start server
  restServer.begin();
  Serial.println(F("Listening for connections..."));

  // Init DHT sensor & output pin
  pinMode(7,OUTPUT);
}

void loop() {

  // Handle any multicast DNS requests
  mdns.update();

  // Handle REST calls
  Adafruit_CC3000_ClientRef client = restServer.available();
  rest.handle(client);

}
```

Pour ce qui est du contrôleur de lampe, nous allons simplement nous servir du relais pour allumer et éteindre la lampe et non pas des différents détecteurs impliqués dans le projet. Vous les ajouterez plus tard dans votre interface centrale pour vous exercer.

On inclut d'abord les bibliothèques requises :

```
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <CC3000_MDNS.h>
#include <aREST.h>
```

On continue en déclarant la broche à laquelle le relais est connecté :

```
constint relay_pin = 8;
```

Créons à présent une instance de l'objet aREST qui nous servira à traiter les requêtes provenant de la connexion WiFi :

```
aREST rest = aREST();
```

Définissons aussi le port sur lequel nous voulons accepter les requêtes *via* WiFi. Afin de simplifier les choses, nous choisirons le port 80 qui permettra de commander directement notre carte Arduino à partir d'un navigateur web :

```
#define LISTEN_PORT  80
```

Il nous faut aussi créer une instance du serveur CC3000 :

```
Adafruit_CC3000_Server restServer(LISTEN_PORT);
```

Créez-en une pour le serveur MDNS afin de pouvoir accéder à la carte Arduino sans devoir saisir son adresse IP :

```
MDNSResponder mdns;
```

Nous devons définir la broche du relais en tant que sortie dans la fonction `setup()` du sketch :

```
pinMode(relay_pin,OUTPUT);
```

Nous pouvons nommer la carte et lui attribuer un numéro d'identification. Ces informations nous seront retournées à chaque requête reçue par la carte *via* l'interface de programmation aREST :

```
rest.set_id("2");
rest.set_name("relay_module");
```

Donnons maintenant un nom à la carte Arduino sur le réseau, par exemple `arduino`. La carte apparaîtra ainsi sur votre réseau sous le nom **arduino.local** :

```
if (!mdns.begin("arduino", cc3000)) {
while(1);
}
```

Pour finir, toujours dans la fonction `setup()`, nous lançons le serveur CC3000 et attendons des connexions entrantes :

```
restServer.begin();
Serial.println(F("Listening for connections..."));
```

À présent, mettons à jour le serveur MDNS dans la fonction `loop()` du sketch :

```
mdns.update();
```

Traitons alors n'importe quelle connexion entrante à l'aide de la bibliothèque aREST :

```
Adafruit_CC3000_ClientRef client = restServer.available();
rest.handle(client);
```

Il est temps de tester ce sketch sur notre projet. Téléchargez le code à partir du dépôt GitHub du livre et veillez à bien modifier le nom et le mot de passe du réseau WiFi sur lequel la puce WiFi va se connecter. Chargez le code sur la carte Arduino et ouvrez le moniteur série. Vous devriez voir apparaître ceci :

```
Listening for connections...
```

À présent, fermez le moniteur série et ouvrez votre navigateur web. Il est maintenant possible d'effectuer des requêtes directes vers l'interface de programmation REST fonctionnant sur la carte afin d'avoir le contrôle sur les broches de la carte Arduino. Pour allumer la lampe par exemple, saisissez ceci :

```
http://arduino.local/digital/8/1
```

Vous devriez entendre le relais s'actionner, voir la lampe s'allumer et un message de confirmation s'afficher dans votre navigateur web :

```
Pin D8 set to 1
```

Pour éteindre la lampe, entrez :

```
http://arduino.local/digital/8/0
```

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

CRÉER L'INTERFACE CENTRALE

Vous allez maintenant créer l'interface qui vous permettra de piloter l'installation entière depuis un ordinateur. Grâce à elle, vous piloterez la lampe en WiFi et obtiendrez un suivi du statut de vos détecteurs de mouvement sans fil XBee et ce, à partir d'une même page.

Cette partie est semblable à celle des chapitres précédents consacrés à la réalisation d'une interface. Vous pouvez donc ignorer les premières explications sur le début du code si vous vous sentez à l'aise dans ce domaine.

Nous allons nous servir de Node.js pour développer cette interface, ainsi que toutes celles du livre. Tout d'abord, nous allons créer le fichier principal **app.js** que nous exécuterons plus tard. Celui-ci contient le cœur de la partie logicielle du projet.

Vous trouverez ci-dessous la version complète du code de ce fichier :

```
// Module
var express = require('express');
var app = express();

// Define port
var port = 3000;

// View engine
app.set('view engine', 'jade');

// Set public folder
app.use(express.static(__dirname + '/public'));

// Rest
var rest = require("arest")(app);
rest.addDevice('http','192.168.1.103');
rest.addDevice('xbee','/dev/tty.usbserial-A702LF8B');

// Serve interface
app.get('/', function(req, res){
  var devices = rest.getDevices();
  res.render('interface', {devices: devices});
});
```

```
// Start server
app.listen(port);
console.log("Listening on port " + port);
```

Commençons par l'importation du module `express` :

```
var express = require('express');
```

Créons maintenant notre application à partir de la structure `express` et définissons le port à 3000 :

```
var app = express();
var port = 3000;
```

Indiquons également à notre logiciel où trouver les fichiers de l'interface graphique dans lesquels nous inscrirons du code ; définissons également **Jade** en tant que moteur de vue (*view engine*) par défaut :

```
app.use(express.static(__dirname + '/public'));
app.set('view engine', 'jade');
```

À ce stade, il nous faut importer le module node-aREST chargé de traiter les communications entre l'interface, les modules XBee et la puce WiFi. Nous devons ici définir le port série sur lequel la carte XBee Explorer USB est connectée ainsi que l'adresse IP de la puce WiFi :

```
var rest = require("arest")(app);
rest.addDevice('xbee','/dev/tty.usbserial-A702LF8B');
rest.addDevice('http','192.168.1.103');
```

Créons maintenant la « route » principale de notre serveur. On définit cette route en associant l'URL racine de notre serveur au fichier Jade correspondant. Nous voulons créer l'interface de manière automatique en fonction du nombre de détecteurs de mouvement présents. Pour ce faire, récupérons tout d'abord l'ensemble des données des modules et transférons-les vers le fichier Jade :

```
app.get('/', function(req, res){
  var devices = rest.getDevices();
  res.render('interface', {devices: devices});
});
```

Pour finir avec le fichier app.js, on lance l'application sur le port défini plus tôt et on inscrit un message dans la console :

```
app.listen(port);
console.log("Listening on port " + port);
```

Voilà pour ce qui est du fichier principal du serveur. Nous allons maintenant créer l'interface. Voyons en premier lieu ce que le fichier Jade contient. Le fichier se situe dans le dossier **/views** de notre projet.

Vous trouverez ci-dessous la version complète du code pour ce fichier :

```
doctype
html
  head
    title Home Automation System
    link(rel='stylesheet', href='/css/interface.css')
    link(rel='stylesheet',
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0
                                    /css/bootstrap.min.css")
    script(src="https://code.jquery.com/jquery-2.1.1.min.js")
    script(src="/js/interface.js")
  body
    .container
      h1.text-center Home Automation System
      .row.voffset
        h2 Lamp Control
      .row
        .col-md-6
          button.btn.btn-block.btn-primary.btn-lg#lamp1 On
        .col-md-6
          button.btn.btn-block.btn-danger.btn-lg#lamp2 Off
      .row
        .col-md-4
          h3#powerDisplay Power:
.col-md-4
          h3#lightDisplay Light level:
.col-md-4
          h3#status Offline
      .row.voffset
        h2 XBee Motion Sensors
      if (devices != '[]')
        each device in devices
          if (device.type == 'xbee')
            .row
              .col-md-4
                h3 Sensor #{device.id}
              .col-md-4
                h3.display(id=device.id)
```

Premièrement, on importe les différents fichiers JavaScript qui vont traiter le clic sur l'interface et envoyer les commandes correspondantes à la carte Arduino :

```
script(src="https://code.jquery.com/jquery-2.1.1.min.js")
script(src="/js/interface.js")
```

Nous allons utiliser la structure Twitter Bootstrap pour améliorer l'aspect de notre interface :

```
link(rel='stylesheet', href='/css/interface.css')
link(rel='stylesheet',
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0
                                /css/bootstrap.min.css")
```

Vous remarquerez que ce fichier est constitué principalement de deux blocs. Le premier est destiné à créer deux boutons pour commander la lampe en WiFi et afficher les données des détecteurs :

```
.row.voffset
  h2 Lamp Control
  .row
    .col-md-6
      button.btn.btn-block.btn-primary.btn-lg#lamp1 On
    .col-md-6
      button.btn.btn-block.btn-danger.btn-lg#lamp2 Off
    .row
      .col-md-4
        h3#powerDisplay Power:
.col-md-4
        h3#lightDisplay Light level:
.col-md-4
        h3#status Offline
```

La seconde partie du fichier est chargée de faire apparaître le statut des détecteurs de mouvement XBee :

```
if (devices != '[]')
  each device in devices
    if (device.type == 'xbee')
      .row
        .col-md-4
          h3 Sensor #{device.id}
        .col-md-4
          h3.display(id=device.id)
```

Examinons à présent le contenu du fichier interface.js qui se trouve dans le dossier **public/js** du projet. Vous trouverez ci-dessous la version complète du code pour ce fichier :

```
$(document).ready(function() {

  $.get('/devices', function( devices ) {

    // Update sensors and repeat every 5 seconds
    setTimeout(updateSensors, 500);
    setInterval(updateSensors, 5000);

    // Function to control the lamp
    $('#lamp1').click(function(){
      $.get('/' + devices[0].name + '/digital/8/1');
    });

    $('#lamp2').click(function(){
      $.get('/' + devices[0].name + '/digital/8/0');
    });

    // Update lamp sensors
```

```
    function updateSensors(){

      // Update light level
      $.get('/' + devices[0].name + '/light', function(json_data) {

        $("#lightDisplay").html("Light level: " + json_data.light
                                                + " %");

        // Update status
        if (json_data.connected == 1){
          $("#status").html("Lamp Online");
          $("#status").css("color","green");
        }
        else {
          $("#status").html("Lamp Offline");
          $("#status").css("color","red");
        }

        // Update power
        $.get('/' + devices[0].name + '/power', function(json_data)
{
          $("#powerDisplay").html("Power: " + json_data.power
                                            + " W");
        });

      });
    }

    // Set inputs for motion sensors
    for (i = 0; i < devices.length; i++){

      // Get device
      var device = devices[i];

      // Set input

      if (device.type == 'xbee'){
         $.get('/' + device.name + '/mode/8/i');
      }

    }

    setInterval(function() {

      for (i = 0; i < devices.length; i++){

        // Get device
        var device = devices[i];

        // Get data
        if (device.type == 'xbee'){
          $.get('/' + device.name + '/digital/8',
```

```
                                function(json_data) {

                    // Update display
                    if (json_data.return_value == 0){
                      $("#" + json_data.id).html("No motion");
                      $("#" + json_data.id).css("color","red");
                    }
                    else {
                      $("#" + json_data.id).html("Motion detected");
                      $("#" + json_data.id).css("color","green");
                    }
                  });
              }

          }

        }, 2000);

      });

    });
```

Vous constaterez que ce code renvoie aux fonctions étudiées dans les chapitres précédents et à la partie consacrée à la réalisation d'une lampe intelligente.

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/home-automation-arduino

Testons à présent l'interface. Assurez-vous de télécharger l'ensemble des fichiers contenus dans le dépôt GitHub et de mettre à jour si nécessaire le code à l'aide de vos propres données, telles que l'adresse de la carte Arduino et le port série de la carte XBee Explorer. Vérifiez aussi que la carte est programmée avec le code donné précédemment dans ce chapitre.

Rendez-vous ensuite dans le dossier de l'interface en utilisant un terminal et entrez la commande suivante pour installer les modules aREST, Express et Jade :

```
sudo npm install arest express jade
```

Si vous utilisez Windows, vous devez laisser de côté **sudo** qui se situe devant les commandes. Il est recommandé d'utiliser l'application Terminal installée par défaut avec Node.js. Pour finir, vous pouvez lancer le serveur Node.js en tapant :

```
sudo node app.js
```

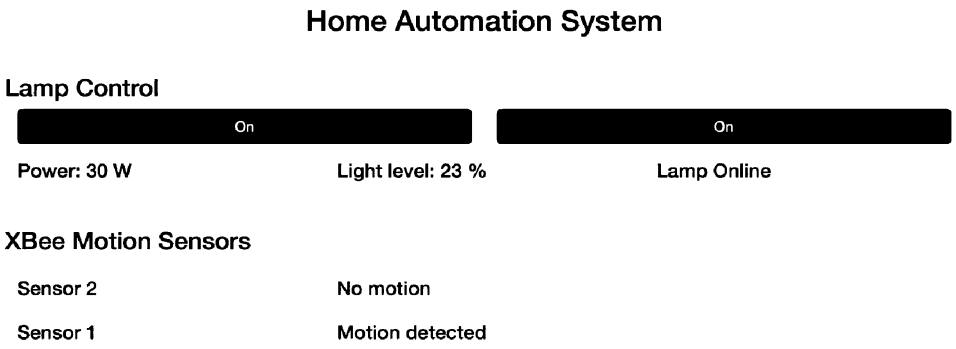
Vous obtiendrez alors le message suivant dans le terminal :

```
Listening on port 3000
```

Ouvrez votre navigateur web et écrivez :

```
localhost:3000
```

L'interface devrait apparaître dans votre navigateur web, accompagnée des boutons pour le pilotage de la lampe et de l'état des détecteurs de mouvement XBee. Faites un mouvement devant le capteur. Vous devriez observer des changements sur l'interface :



Vous pouvez aussi cliquer sur l'un des boutons pour allumer ou éteindre la lampe de manière instantanée.

Si vous n'obtenez pas ce résultat à ce stade, il vous faut vérifier plusieurs choses :

- Assurez-vous d'utiliser la dernière version du code à partir du dépôt GitHub du livre.
- Vérifiez que les fichiers ont bien été modifiés selon notamment les paramètres du port série de votre module Bluetooth et l'adresse de votre module WiFi.
- Veillez à installer les modules Node.js demandés avec l'outil **npm** avant de lancer votre navigateur web.

POUR ALLER PLUS LOIN

Résumons ce que nous avons vu dans ce projet. Nous sommes partis des projets des chapitres 7 et 8 pour aboutir à une petite installation domotique. À partir des détecteurs de mouvement sans fil XBee et du contrôleur de lampe WiFi mis au point précédemment, nous avons testé ces modules puis intégré l'ensemble dans une interface unique ce qui a permis d'avoir un contrôle intégral sur le système à partir d'une même page.

Vous pouvez vous appuyer sur ce projet pour réaliser une multitude de montages plus complexes. En vous servant presque du même code, vous pouvez ajouter d'autres contrôleurs de lampe WiFi et des détecteurs de mouvement sans fil XBee. Ajoutez d'autres capteurs au projet si vous le souhaitez.

Il est également possible de créer des comportements plus complexes pour la lampe, en associant par exemple les mesures du détecteur de mouvement à l'activation du relais. Si ceux-ci se trouvent dans la même pièce, vous pouvez configurer l'installation de façon à ce que le relais s'active automatiquement à chaque fois que des mouvements sont détectés (aux toilettes, par exemple).

Notez qu'il n'est pas recommandé d'avoir recours à plusieurs technologies sans fil au sein d'un même système. Il est préférable par exemple d'utiliser le WiFi pour chaque actionneur (relais) et de choisir XBee pour les capteurs et détecteurs (de mouvement, d'humidité, de température, etc.).

Partie III

Concevoir des installations connectées

12 Concevoir une station de mesures sur le cloud

Dans ce chapitre, nous allons créer une station de mesures qui partagera automatiquement ses données avec un service de cloud computing avec seulement une carte Arduino Uno, un module WiFi et quelques capteurs.

L'objectif ici est de vous montrer qu'il est très simple de créer un projet qui s'inscrit dans le cadre de l'Internet des objets. Vous allez apprendre à stocker des mesures en ligne. Ensuite, nous verrons comment afficher ces données de manière automatique afin de pouvoir les consulter à partir d'une même page web.

MATÉRIEL ET LOGICIEL REQUIS

Pour ce projet, il vous faudra une carte **Arduino Uno**.

Pour effectuer des mesures de température et d'humidité, vous aurez besoin d'un **capteur DHT11**, accompagné d'une **résistance de 4,7 kΩ**. L'utilisation d'un capteur DHT22, plus précis, est possible, vous devrez cependant modifier une ligne de code.

Pour mesurer le niveau d'éclairage, j'ai choisi une **photorésistance** dotée d'une résistance de 10 kΩ. On obtiendra en retour un signal proportionnel au niveau d'éclairage ambiant.

Équipez-vous également d'une **puce CC3000** pour vous connecter en WiFi. Il existe de nombreuses cartes équipées de cette puce sur le marché.

Je vous conseille toutefois d'utiliser le module WiFi CC3000 qui est le seul à avoir fonctionné sans problème dans mon cas. Comme nous l'avons vu précédemment, celui-ci dispose d'un régulateur de tension et d'une antenne intégrée.

Pour finir, vous aurez besoin d'une **plaque d'essai** et de quelques fils de raccordement pour effectuer les différents branchements.

LISTE DES COMPOSANTS

- **Carte Arduino Uno** (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- **Capteur DHT11 avec résistance de 4,7 kΩ** (http://snootlab.com/adafruit/255-capteur-de-temperateur-et-d-humidite-dht11-extras-.html)
- **Photorésistance** (http://snootlab.com/composants/97-photoresistance.html)
- **Résistance de 10 kΩ** (http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html)

☞

☞

- Module WiFi CC3000 (http://www.gotronic.fr/art-module-wifi-cc3000-ada1469-21904.html)
- Plaque d'essai (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

Côté logiciel, vous devrez vous procurer la bibliothèque pour le capteur DHT ainsi que la bibliothèque de la puce WiFi CC3000.

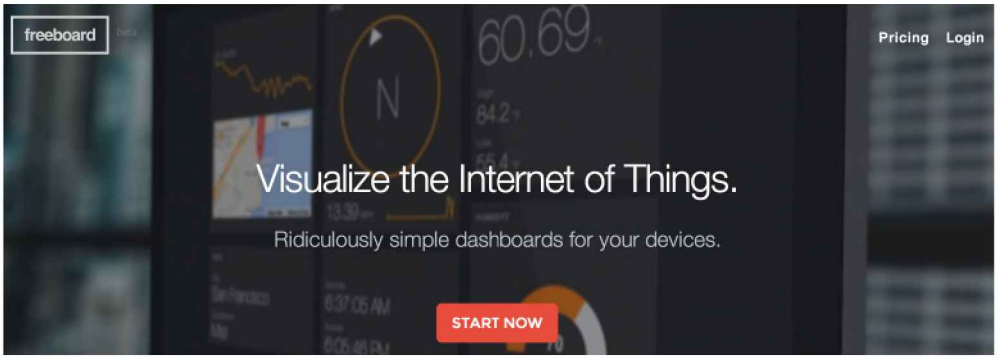
Téléchargez la bibliothèque pour le capteur DHT :
https://github.com/adafruit/DHT-sensor-library
Téléchargez la bibliothèque de la puce WiFi CC3000 :
https://github.com/adafruit/Adafruit_CC3000_Library

Pour installer une bibliothèque, il vous suffit d'extraire son dossier vers **/libraries**, situé dans le dossier racine Arduino.

Il sera nécessaire de créer un compte Freeboard car c'est bien cette plateforme que nous utiliserons pour afficher les données relevées.

Rendez-vous à l'adresse suivante : https://www.freeboard.io/

Ce lien vous conduira à la page d'accueil du site, où vous pourrez créer un compte :



CONNECTER DES CAPTEURS À ARDUINO

Les connexions matérielles du projet sont relativement simples : il nous faut connecter le capteur DHT11, la partie gérant la mesure du niveau d'éclairage avec la photorésistance et enfin la carte WiFi CC3000. L'illustration suivante synthétise les connexions matérielles.

Raccordons tout d'abord la broche 5 V de la carte Arduino Uno à la rangée rouge de la plaque d'essai et la broche de masse à la rangée bleue. Placez ensuite le capteur DHT ainsi que le module WiFi sur la plaque d'essai.

Puis, connectez la broche n° 1 du capteur DHT11 (voir le schéma) à la rangée rouge de la plaque d'essai et la broche n° 4 (GND) à la rangée bleue. Après cette étape, connectez la broche n° 2 du capteur à la broche n° 7 de la carte Arduino. Pour en finir

avec le branchement du capteur DHT11, insérez la résistance de 4,7 kΩ entre les broches n° 1 et n° 2 du capteur.

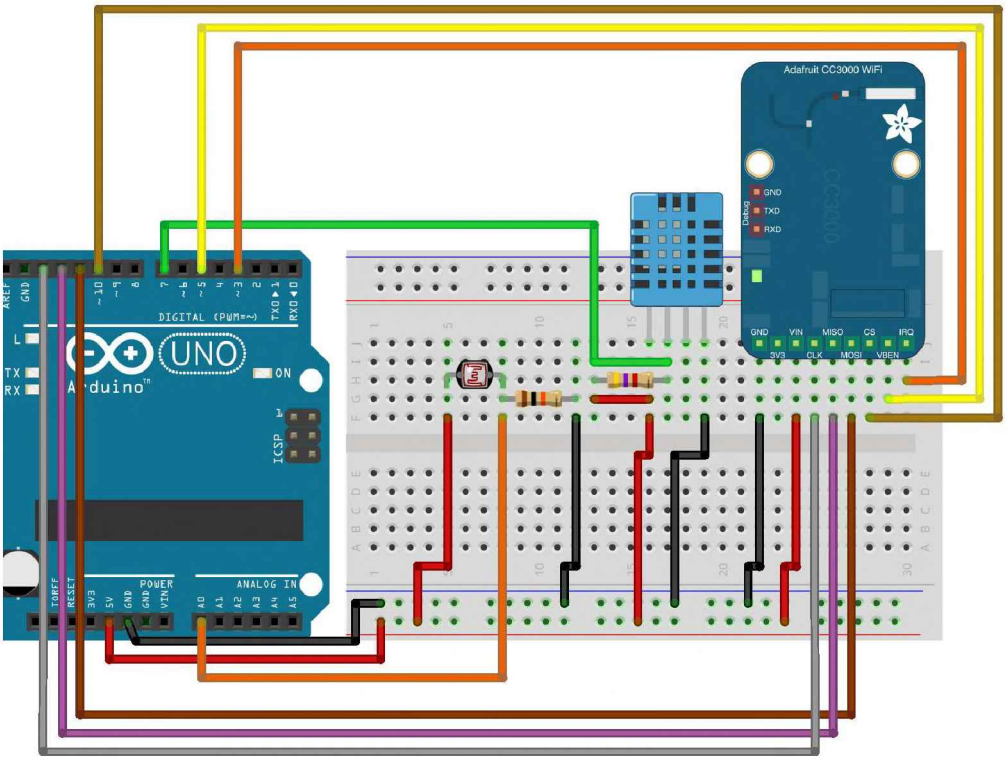


Figure 12.1 Image créée avec le logiciel Fritzing (http://fritzing.org/)

Connectez d'abord la photorésistance en série avec la résistance de 10 kΩ sur la plaque d'essai. L'autre patte de la photorésistance sera raccordée à la rangée rouge de la plaque d'essai et la seconde patte de la résistance à la masse. Enfin, connectez la broche commune à l'entrée analogique A0 de la carte Arduino.

Intéressons-nous maintenant au module WiFi. Connectez tout d'abord la broche IRQ de la carte CC3000 à la broche n° 3 de la carte Arduino, puis la broche VBAT à la broche n° 5 et enfin la broche CS à la broche n° 10.

Il vous faut à présent connecter les broches de l'interface SPI à la carte Arduino : MOSI, MISO, et CLK doivent être branchées respectivement aux broches n° 11, 12 et 13. En ce qui concerne les broches d'alimentation électrique : VIN doit être connectée à la broche Arduino 5 V (rangée rouge) et GND à GND (rangée bleue).

TESTER DES CAPTEURS

Une fois le matériel du projet entièrement assemblé, il nous faut procéder au test des différents capteurs de la carte. Pour ce faire, nous allons écrire un sketch Arduino. Il s'agira de lire les données des capteurs et de les faire apparaître sur le port série.

Le capteur DHT11 est un capteur numérique nécessitant sa propre bibliothèque pour récupérer ses données.

En ce qui concerne la photorésistance, qui n'est autre qu'un capteur purement analogique, nous aurons recours à la fonction `analogRead()` de Arduino afin d'effectuer les relevés de mesures.

Voici la version complète du code de cette partie :

```
// Libraries
#include "DHT.h"

// DHT sensor
#define DHTPIN 7
#define DHTTYPE DHT11

// DHT instance
DHT dht(DHTPIN, DHTTYPE);

void setup()
{
  // Initialize the Serial port
  Serial.begin(9600);

  // Init DHT
  dht.begin();
}


void loop()
{
  // Measure from DHT
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  // Measure light level
  float sensor_reading = analogRead(A0);
  float light = sensor_reading/1024*100;

  // Display temperature
  Serial.print("Temperature: ");
  Serial.print((int)temperature);
  Serial.println(" C");

   // Display humidity
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println("%");
```

```
    // Display light level
    Serial.print("Light: ");
    Serial.print(light);
    Serial.println("%");
    Serial.println("");

    // Wait 500 ms
    delay(500);

}
```

Étudions cela en détail. Premièrement, importons la bibliothèque pour le capteur DHT:

```
#include "DHT.h"
```

Créons une instance :

```
DHT dht(DHTPIN, DHTTYPE);
```

Nous allons initialiser le capteur dans la fonction `setup()` du sketch :

```
dht.begin();
```

Faisons de même avec le port série :

```
Serial.begin(9600);
```

Dans la fonction `loop()`, nous allons lire les données des capteurs en continu et les faire apparaître dans le port série. Commençons par récupérer les données de température et d'humidité :

```
float temperature = dht.readTemperature();
float humidity = dht.readHumidity();
```

En ce qui concerne la photorésistance, nous allons lire en premier lieu les données provenant de l'entrée analogique A0, soit une valeur comprise entre 0 et 1 023 étant donné que le convertisseur analogique-numérique de la carte Arduino Uno possède une résolution de 10 bits, c'est-à-dire 1 024 niveaux.

Nous divisons ensuite cette valeur par 1 024 et multiplions le résultat par 100 pour obtenir un niveau d'éclairage en pourcentage :

```
float sensor_reading = analogRead(A0);
float light = sensor_reading/1024*100;
```

Après cela, il nous faut inscrire ces données sur le port série. Commençons par la température :

```
Serial.print("Temperature: ");
Serial.print((int)temperature);
Serial.print((char)223);
Serial.println("C");
```

Procédons de la même manière avec les données relatives à l'humidité :

```
Serial.print("Light: ");
Serial.print(light);
Serial.println("%");
```

Pour finir, nous ajoutons un délai de 500 ms entre chaque série de mesures :

```
delay(500);
```


Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/iot-book

Il est temps à présent de tester ce premier sketch Arduino. Chargez le code sur la carte Arduino, ouvrez le moniteur série du logiciel Arduino IDE (en vérifiant que la vitesse du port série est semblable à celle définie dans le code) et voici le résultat que vous devriez obtenir :

```
Temperature: 25 C
Humidity: 36.00%
Light: 83,79%
```

Si vous obtenez ce résultat, c'est que vos capteurs fonctionnement correctement, bravo ! Tentez par exemple de passer votre main devant la photorésistance. Vous devriez observer une évolution soudaine du niveau d'éclairage.

Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Premièrement, assurez-vous d'avoir téléchargé et installé correctement les bibliothèques citées dans ce chapitre.
- Veillez à ce que les capteurs soient correctement raccordés à la carte Arduino en vous référant aux indications données dans le chapitre.
- Enfin, assurez-vous d'utiliser la version la plus récente du code à partir du dépôt GitHub du livre.

TRANSFÉRER DES DONNÉES SUR LE CLOUD

Abordons la partie centrale du chapitre : transférer des données en WiFi vers le cloud. Pour cela, nous aurons recours au service du site Dweet.io, permettant de stocker des données en ligne *via* une interface de programmation assez simple.

Voici l'adresse : https://dweet.io/

Le site Dweet.io ne requiert aucun compte ni aucune configuration : vous pouvez charger vos données sans perdre de temps. Sur ce site, on appelle « things » les objets vers lesquels on transférera de nouvelles données par l'intermédiaire de requêtes HTTP. Si vous transférez des données sur un nouvel objet encore inexistant, il se créera automatiquement.

Rédigeons à présent le sketch qui nous permettra de prendre des mesures de manière automatique, de se connecter au serveur Dweet.io et d'y transférer des données.

Voici la version complète du code de cette partie :

```
// Libraries
#include <Adafruit_CC3000.h>
#include <ccspi.h>
#include <SPI.h>
#include "DHT.h"
#include <avr/wdt.h>

// Define CC3000 chip pins
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10

// DHT sensor
#define DHTPIN 7
#define DHTTYPE DHT11

// Create CC3000 instances
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
  ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT, SPI_CLOCK_DIV2);

// DHT instance
DHT dht(DHTPIN, DHTTYPE);

// WLAN parameters
#define WLAN_SSID       "yourWiFiNetwork"
#define WLAN_PASS       "yourPassword"
#define WLAN_SECURITY   WLAN_SEC_WPA2

// Xively parameters
#define thing_name  "yourThingName"

// Variables to be sent
int temperature;
int humidity;
int light;

// IP variable
int32_t ip;

void setup(void)
{
// Initialize
  Serial.begin(115200);

// Start CC3000 chip
  Serial.println(F("\nInitializing..."));
if (!cc3000.begin())
  {
    Serial.println(F("Couldn't begin()! Check your wiring?"));
while(1);
```

```
  }
}

void loop(void)
{

// Connect to WiFi network
  cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY);
  Serial.println(F("Connected!"));

// Start watchdog
  wdt_enable(WDTO_8S);

// Wait for DHCP to complete
  Serial.println(F("Request DHCP"));
while (!cc3000.checkDHCP())
  {
    delay(100);
  }

// Reset watchdog
  wdt_reset();

// Get IP
uint32_t ip = 0;
  Serial.print(F("www.dweet.io -> "));
while  (ip  ==  0)  {
if  (!  cc3000.getHostByName("www.dweet.io", &ip))  {
      Serial.println(F("Couldn't resolve!"));
    }
    delay(500);
  }
  cc3000.printIPdotsRev(ip);
  Serial.println(F(""));

// Reset watchdog
  wdt_reset();

// Measure from DHT sensor
float t = dht.readTemperature();
float h = dht.readHumidity();
  temperature = (int)t;
  humidity = (int)h;

// Measure light level
float sensor_reading = analogRead(A0);
  light = (int)(sensor_reading/1024*100);
  Serial.println(F("Measurements done"));

// Reset watchdog
  wdt_reset();
```

```
// Send request to Dweet.io
  Adafruit_CC3000_Client client = cc3000.connectTCP(ip, 80);
if (client.connected()) {
    Serial.print(F("Sending request... "));

    client.fastrprint(F("GET /dweet/for/"));
    client.print(thing_name);
    client.fastrprint(F("?temperature="));
    client.print(temperature);
    client.fastrprint(F("&humidity="));
    client.print(humidity);
    client.fastrprint(F("&light="));
    client.print(light);
    client.fastrprintln(F(" HTTP/1.1"));
    client.fastrprintln(F("Host: dweet.io"));
    client.fastrprintln(F("Connection: close"));
    client.fastrprintln(F(""));

    Serial.println(F("done."));

  } else {
    Serial.println(F("Connection failed"));
return;
  }

// Reset watchdog
  wdt_reset();
// Read answer
  Serial.println(F("Reading answer..."));
while (client.connected()) {
while (client.available()) {
char c = client.read();
      Serial.print(c);
    }
  }
  Serial.println(F(""));

// Reset watchdog
  wdt_reset();

// Close connection and disconnect
  client.close();
  Serial.println(F("Disconnecting"));
  Serial.println(F(""));
  cc3000.disconnect();

// Reset watchdog & disable
  wdt_reset();
  wdt_disable();

// Wait 10 seconds until next update
```

```
  delay(10000);

}
```

Voyons maintenant ce code en détail. On importe d'abord les bibliothèques requises :

```
#include <Adafruit_CC3000.h>
#include <ccspi.h>
#include <SPI.h>
#include "DHT.h"
#include <avr/wdt.h>
```

Puis, on définit les broches auxquelles le module CC3000 est connecté :

```
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10
```

Nous pouvons ensuite créer une instance de la puce WiFi CC3000 :

```
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS,
ADAFRUIT_CC3000_IRQ, ADAFRUIT_CC3000_VBAT, SPI_CLOCK_DIV2);
```

À présent, vous devez modifier le sketch en inscrivant le nom de votre réseau WiFi ainsi que le mot de passe associé. Si votre réseau n'utilise pas le mode d'authentification WPA2, vous devrez également changer ce paramètre :

```
#define WLAN_SSID       "yourWiFiNetwork"
#define WLAN_PASS       "yourPassword"
#define WLAN_SECURITY   WLAN_SEC_WPA2
```

Il nous faut aussi nommer notre objet. Sur Dweet.io, tous les objets sont par défaut accessibles à tous, ce qui n'est pas un problème ici puisque nous souhaitons transférer et surveiller de simples données, non sensibles. Je recommande toutefois d'éviter de donner un nom trop commun à l'objet. Voici un exemple de nom que vous pouvez donner : **weather_station_l5ir457xda**. Une fois que vous avez choisi un nom approprié, insérez-le dans le sketch :

```
#define thing_name  "yourThingName"
```

Nous devons également définir quelques variables contenant les mesures effectuées par le projet :

```
int temperature;
int humidity;
int light;
```

À présent, on initialise la puce CC3000 dans la fonction `setup()` du sketch :

```
Serial.println(F("\nInitializing..."));
if (!cc3000.begin())
{
  Serial.println(F("Couldn't begin()! Check your wiring?"));
```

```
while(1);
}
```

Dans la fonction `loop()`, on connecte d'abord la puce CC3000 à notre réseau WiFi local :

```
cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY);
```

On lance ensuite le chien de garde (*watchdog*). Il s'agit d'un circuit indépendant du code faisant partie du microcontrôleur Arduino. Il se chargera de réinitialiser le sketch Arduino automatiquement après une période donnée, à moins de lui envoyer un signal de réinitialisation. Grâce à cela, si le sketch Arduino se bloque (en cas d'échec de la connexion aux serveurs Dweet.io par exemple), il se réinitialisera pour permettre au projet de continuer à fonctionner. Premièrement, il nous faut activer le chien de garde avec un délai de 8 secondes :

```
wdt_enable(WDTO_8S);
```

Lorsque vous serez connecté, effectuez un relevé de température, d'humidité et de niveau d'éclairage :

```
float t = dht.readTemperature();
float h = dht.readHumidity();
temperature = (int)t;
humidity = (int)h;

float sensor_reading = analogRead(A0);
light = (int)(sensor_reading/1024*100);
Serial.println(F("Measurements done"));
```

Une fois en possession des données, on les transfère vers le service Dweet.io. Il vous suffit alors de vous connecter aux serveurs et d'envoyer les données *via* une requête HTTP GET classique. Pour cela, saisissez les lignes de code suivantes :

```
Adafruit_CC3000_Client client = cc3000.connectTCP(ip, 80);
if (client.connected()) {
  Serial.print(F("Sending request... "));

  client.fastrprint(F("GET /dweet/for/"));
  client.print(thing_name);
  client.fastrprint(F("?temperature="));
  client.print(temperature);
  client.fastrprint(F("&humidity="));
  client.print(humidity);
  client.fastrprint(F("&light="));
  client.print(light);
  client.fastrprintln(F(" HTTP/1.1"));

  client.fastrprintln(F("Host: dweet.io"));
  client.fastrprintln(F("Connection: close"));
  client.fastrprintln(F(""));

  Serial.println(F("done."));
```

```
} else {
  Serial.println(F("Connection failed"));
return;
}
```

Pensez à réinitialiser le chien de garde à la suite de cette longue requête :

```
wdt_reset();
```

Après l'envoi de ces données, on lit la réponse du serveur Dweet.io pour s'assurer qu'elles ont bien été reçues de l'autre côté. On met fin à la connexion sans oublier de déconnecter la puce CC3000 du réseau WiFi afin d'économiser de l'énergie :

```
Serial.println(F("Reading answer..."));
while (client.connected()) {
while (client.available()) {
char c = client.read();
    Serial.print(c);
  }
}
Serial.println(F(""));

// Reset watchdog
wdt_reset();

// Close connection and disconnect
client.close();
Serial.println(F("Disconnecting"));
Serial.println(F(""));
cc3000.disconnect();
```

Après cette étape, on désactive le chien de garde :

```
wdt_disable();
```

Pour finir, on répétera cette boucle après un délai de 10 secondes. Vous pouvez adapter ce délai pour des prises de mesure moins fréquentes :

```
delay(10000);
```

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/iot-book

Testons le code à présent. Assurez-vous de bien avoir entré dans le sketch le nom de votre propre objet ainsi que les données relatives à votre réseau WiFi. Vous pouvez désormais charger le code sur la carte Arduino. Ouvrez le moniteur série. Après un certain laps de temps, vous devriez recevoir la réponse du serveur Dweet.io :

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 174
Date: Thu, 24 Jul 2014 12:08:32 GMT
```

```
Connection: keep-alive

{"this":"succeeded","by":"dweeting","the":"dweet","with":
{"thing":"yourThingName","created":"2014-07-24T12:08:32.443Z",
"content":{"temperature":25,"humidity":35,"light":59}}}
```

Vérifiez en ligne que les données ont bien été enregistrées. Pour cela, saisissez cette URL dans votre navigateur :

```
https://dweet.io/get/dweets/for/yourThingName
```

Vous devez bien entendu remplacer le nom par celui de votre objet que vous avez d'ailleurs inscrit dans le sketch Arduino. Vous devriez remarquer au moins une valeur pour cet objet. Par exemple :

```
{
"thing": "weather_station_1a2cx3s8",
"created": "2014-07-13T12:58:10.924Z",
"content": {
"temperature": 26,
"humidity": 40,
"light": 69
  }
}
```

Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Assurez-vous que votre connexion Internet est opérationnelle et que votre carte Arduino est bien connectée au réseau WiFi.
- Vérifiez aussi l'exactitude de la réponse donnée par le site Dweet.io.
- Pour terminer, assurez-vous que vous avez correctement saisi le nom de votre objet dans le sketch Arduino.

ACCÉDER AUX DONNÉES SUR LE CLOUD

Vous avez appris à stocker des données sur le cloud à l'aide du service Dweet.io. Votre carte Arduino envoie désormais des données sur ce site en permanence.

Toutefois, si la lecture des données renvoyées est correcte, elle pourrait être optimisée par une visualisation graphique. C'est la raison pour laquelle nous avons créé un compte Freeboard. Retournez sur ce site et saisissez vos identifiants :

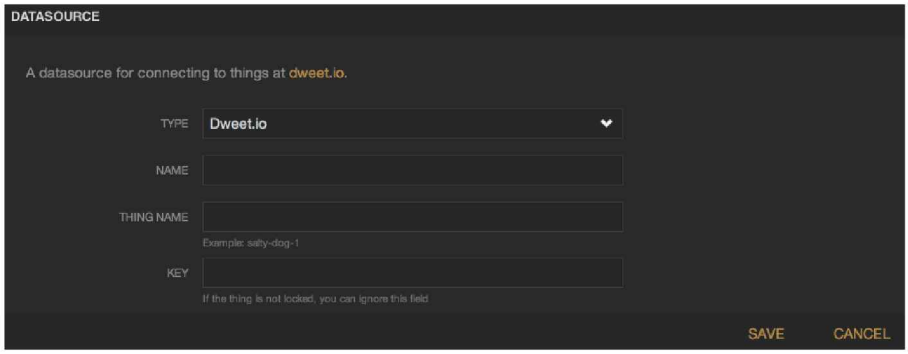
```
https://www.freeboard.io/
```

Dans cette interface, vous aurez la possibilité de créer un nouveau tableau de bord (*dashboard*). Vous pourrez tout superviser à partir de cette interface. Donnez alors un nom à votre tableau de bord puis cliquez sur « Create New ».

Il vous faut maintenant créer deux types d'objets : des « panes[1] » (volets) et des « datasources » (sources de données).

1. Un « pane » est un espace contenant vos données sous forme de graphique(s).

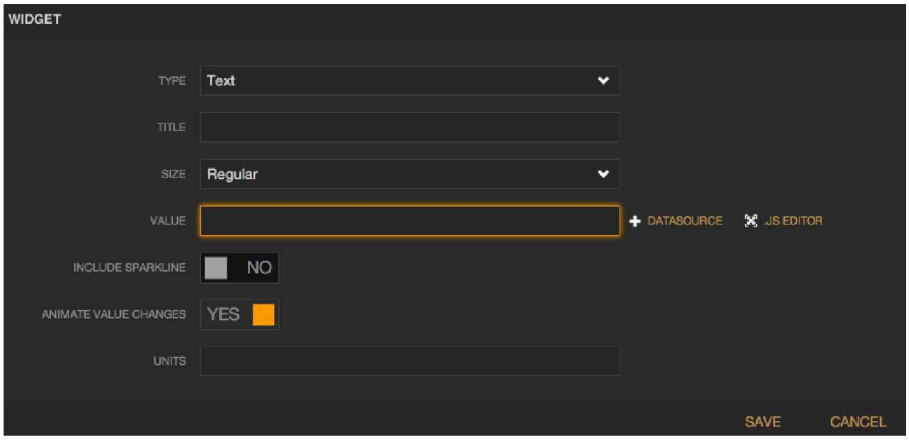
Commençons par les datasources. Par définition, c'est de là que viendront les données qui seront affichées sur le tableau de bord. Cliquez sur « Add » pour ajouter une nouvelle source puis sélectionnez dans le menu « Dweet.io » :



Dans cette zone, il vous faudra également entrer un nom ainsi que celui de l'objet de votre sketch Arduino. Cliquez ensuite sur « Save ».

Créons à présent un nouveau volet. Cliquez sur le bouton, puis sur l'icône « + » afin d'ajouter un widget (petit programme). Ces programmes sont chargés d'afficher vos données.

J'ai créé trois volets différents afin de faire apparaître simultanément l'ensemble des données enregistrées. J'ai commencé par ce qu'on appelle les widgets de texte :



Il est essentiel ici de sélectionner le champ « Value » en cliquant sur « + Datasource » sur la droite.

Cette manipulation servira à associer le widget aux données provenant de Dweet.io.

Vous pouvez par exemple nommer votre premier widget « temperature » puis l'associer au champ « temperature » de la « datasource » que nous avons définie plus tôt.

Voici le résultat que j'ai obtenu avec trois widgets de texte :

J'ai également effectué quelques manipulations dans l'interface et transformé ces widgets en widgets de jauge basés sur les mêmes données :

Vous remarquerez que les données sont mises à jour instantanément. Vous êtes maintenant en mesure de superviser ces données à partir de n'importe quel ordinateur connecté à Internet en rentrant simplement l'URL de votre tableau de bord.

Si vous rencontrez des problèmes, vérifiez en premier lieu que les données apparaissent sur le site web Dweet.io. Puis, assurez-vous d'avoir correctement saisi le nom de votre objet sur le tableau de bord de **Freeboard.io**.

POUR ALLER PLUS LOIN

Résumons ce que nous avons vu dans ce chapitre. Vous venez d'apprendre à déposer sur le cloud des données provenant de votre carte Arduino et à les rendre par la même occasion accessibles à l'extérieur de votre réseau WiFi local. Nous avons vu également comment suivre ces données à partir d'un tableau de bord central en ligne, afin d'y avoir accès de n'importe où.

Vous pouvez partir de ce projet pour réaliser une multitude de montages plus complexes. Un premier montage consisterait à ajouter d'autres capteurs au projet, tels qu'un anémomètre ou un détecteur de fumée à installer dans plusieurs pièces de votre domicile et qui enverront des données vers un objet différent sur Dweet.io. Enfin, mettez à profit les connaissances que vous venez d'acquérir pour suivre toutes ces données à partir d'un même endroit.

13 Piloter une lampe depuis le web

Jusqu'ici, nous avons réalisé des projets s'inscrivant dans le cadre de l'Internet des objets qui étaient chargés de transférer des données vers le cloud pour assurer un stockage sécurisé et un accès en ligne. Cette fois-ci, nous nous intéresserons à quelque chose de différent. L'objectif est de s'approprier le contrôle d'un appareil, en l'occurrence une lampe, depuis le web.

Pour atteindre cet objectif, nous utiliserons un service appelé Teleduino. Nous aurons besoin d'une carte d'extension Ethernet afin de connecter la carte Arduino à Internet, connecter une lampe au projet *via* un module relais et commander ce relais à distance. Teleduino fonctionne à partir de n'importe quel navigateur web. Vous pourrez donc piloter votre lampe depuis n'importe quel pays dans le monde. Plongeons au cœur du projet.

MATÉRIEL ET LOGICIEL REQUIS

Dressons tout d'abord une liste des composants requis pour ce projet. En ce qui concerne la carte Arduino, j'ai utilisé le modèle Uno R3. Pour la connectivité Internet, j'ai choisi la carte d'extension Ethernet pour Arduino, très pratique pour connecter nos projets à Internet et en réaliser d'autres *via* l'Internet des objets.

En ce qui concerne le module relais, j'ai choisi un modèle 5 V de la marque Polulu qui associe un relais à une carte. Ce module comporte tous les composants nécessaires au pilotage du relais à partir de la carte Arduino.

Pour connecter la lampe au projet, je me suis servi de deux fiches secteur munies de simples câbles et comprenant une prise femelle (où la lampe viendra se brancher) et une prise mâle (à relier à la prise murale) identiques à celles déjà présentées précédemment.

Remarquez qu'il est tout à fait possible de tester ce projet sans connecter quoi que ce soit au relais : assemblez tout d'abord la partie Arduino, puis choisissez l'appareil auquel vous souhaitez vous connecter.

LISTE DES COMPOSANTS

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Carte d'extension Ethernet pour Arduino (http://snootlab.com/arduino/163-ethernet-shield-r3.html)
- Module relais (http://snootlab.com/tinker-it/229-mod.html)
- Fils de raccordement mâle/femelle (http://snootlab.com/cables/23-kit-10-cordons-6-m-f.html)

Côté logiciel, équipez-vous de Arduino IDE. Procurez-vous également la bibliothèque **Teleduino.**

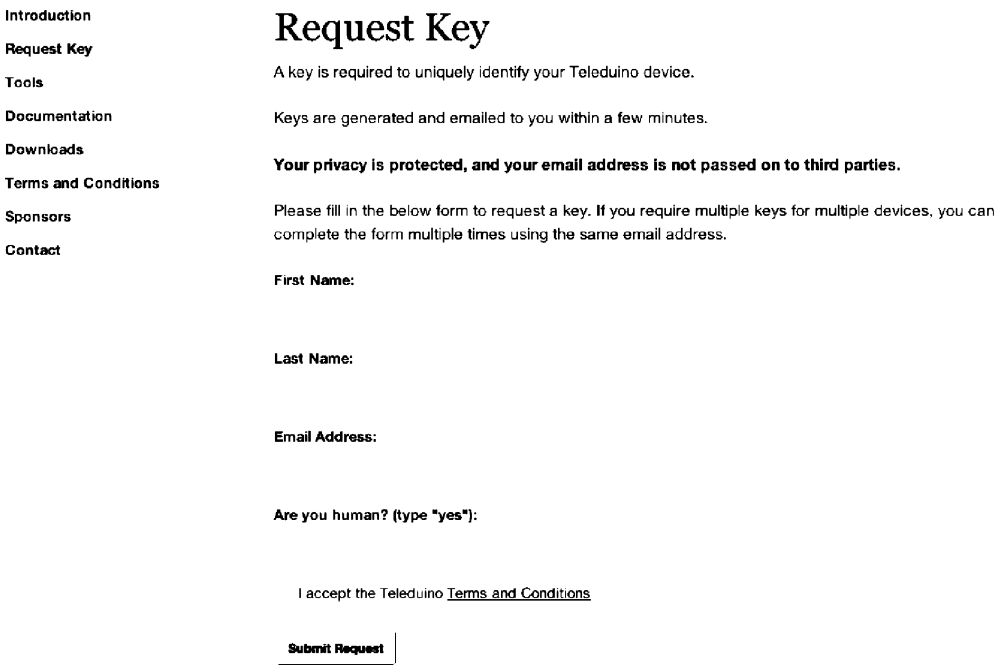
Téléchargez la bibliothèque Teleduino, à cette adresse :
https://www.teleduino.org/downloads/

Afin d'installer une bibliothèque, il vous suffit d'extraire son dossier vers **/libraries**, situé dans votre dossier racine Arduino (créez ce dossier s'il n'existe pas).

Il vous faudra également une clé d'interface de programmation (ou clé API) pour pouvoir profiter du service Teleduino.

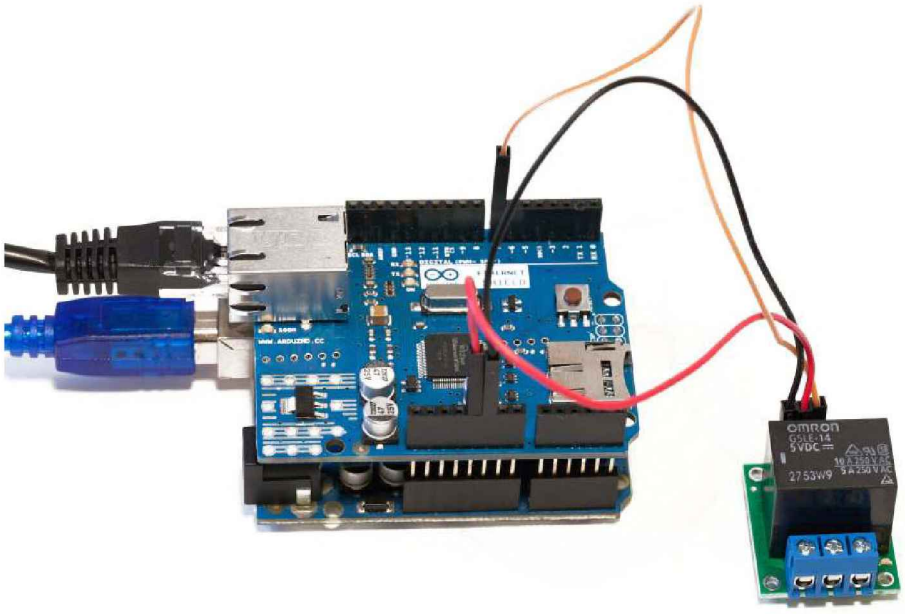
Rendez-vous sur cette page : https://www.teleduino.org/tools/request-key

Vous avez la possibilité de demander une clé :

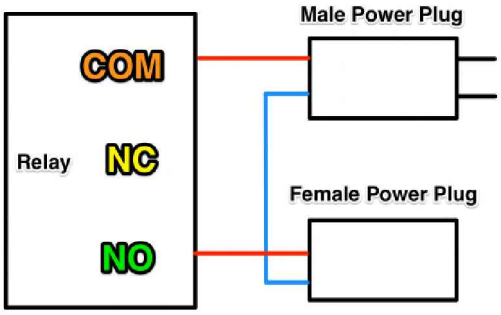


CONNECTER UN RELAIS OU UNE LAMPE À ARDUINO

L'assemblage du matériel de ce projet est très simple. Imbriquez tout d'abord la carte d'extension Ethernet sur la carte Arduino puis connectez-la au routeur Internet *via* un câble Ethernet. Pour le module relais, vous aurez trois broches à connecter : VCC, GND et SIG. Connectez la broche VCC à la broche 5 V de la carte Arduino et la broche GND à la broche de masse. Pour finir, connectez la broche SIG à la broche n° 7 de la carte Arduino. Voici un aperçu du projet jusqu'ici :

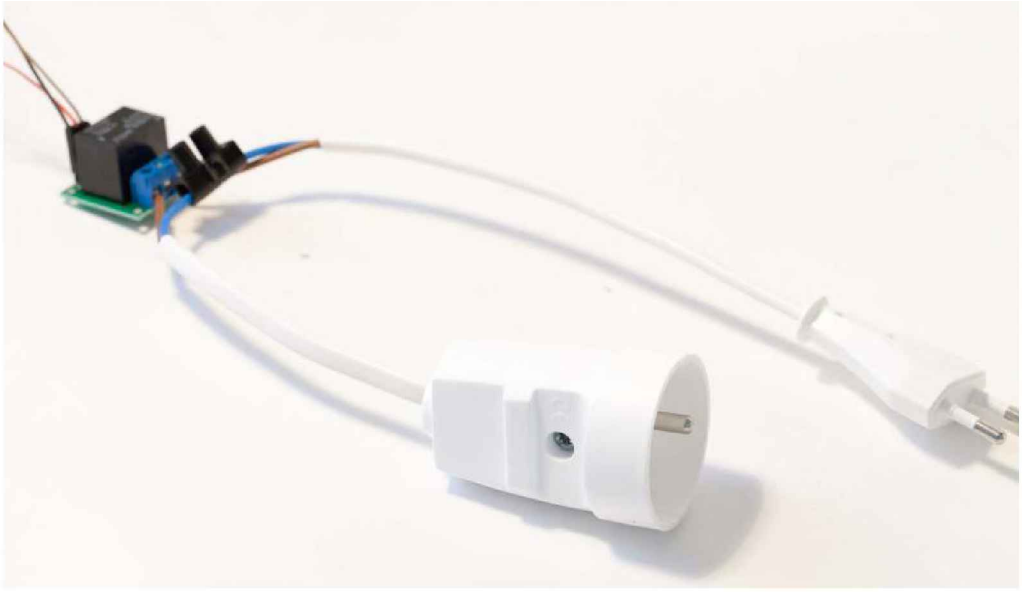


Nous allons maintenant ajouter la lampe au montage. L'idée est de faire passer l'alimentation électrique principale (de la prise murale) par le relais *via* la broche COM en entrée, puis la broche NO en sortie, pour atteindre finalement la lampe. Reportez-vous à l'illustration suivante pour un aperçu visuel des connexions à effectuer :



Remarquez que je n'ai raccordé ni la prise mâle, ni la prise femelle à la terre pour ce projet. Notez aussi que ce projet n'est proposé qu'à des fins pédagogiques. Il pourrait en effet présenter un danger chez vous car un raccord direct au réseau électrique occasionne des risques. Si vous souhaitez installer ce projet chez vous, prenez soin de recouvrir le projet d'un boîtier plastique afin que personne ne le touche lorsqu'il est activé.

Voici un aperçu du projet entièrement assemblé :



TESTER LE RELAIS

Voyons à présent si le projet fonctionne. Nous allons tester l'élément principal du montage : le module relais. Pour ce faire, activons et désactivons le relais par intervalles de 5 secondes simplement pour voir s'il fonctionne. À ce stade, vous pouvez d'ores et déjà raccorder un appareil (comme une lampe) au projet, connecter l'interrupteur à une prise de courant et constater que toutes les connexions sont bien établies.

Voici la version complète du code de cette partie :

```
// Sketch to test the relay

// Relay pin
constint relay_pin = 7;

void setup() {
  pinMode(relay_pin,OUTPUT);

}

void loop() {
```

```
// Activate relay
  digitalWrite(relay_pin, HIGH);

// Wait for 5 seconds
  delay(5000);

// Deactivate relay
  digitalWrite(relay_pin, LOW);

// Wait for 5 seconds
  delay(5000);
}
```

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/iot-book

Il est temps de tester le projet. Assurez-vous que la lampe est correctement connectée au projet et que la fiche mâle est branchée à la prise murale. Après cela, chargez le sketch Arduino sur la carte. Vous devriez voir le relais s'activer et se désactiver à 5 secondes d'intervalle, allumant et éteignant l'équipement connecté.

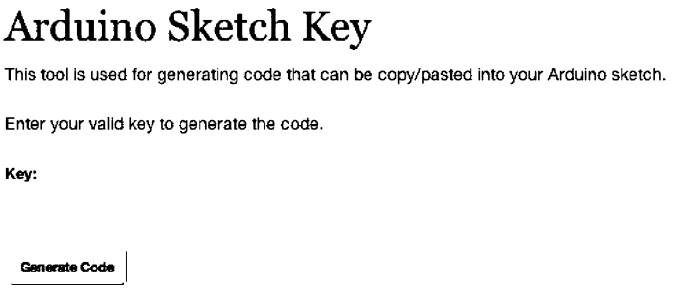
PILOTER VOTRE PROJET DEPUIS N'IMPORTE OÙ

Après avoir vérifié le bon fonctionnement du relais, abordons ce qui nous intéresse le plus dans ce chapitre : piloter le relais depuis n'importe quel ordinateur connecté à Internet et ce, à partir d'un navigateur web. Seule votre clé API suffit.

Il nous reste une opération à effectuer avant de nous pencher sur le sketch Arduino de cette partie. Il vous faut convertir la clé API au format hexadécimal afin de l'insérer dans le sketch Arduino.

Teleduino dispose d'une page consacrée à ce sujet : https://www.teleduino.org/tools/arduino-sketch-key

De cette adresse vous serez redirigé vers une page où vous pourrez insérer votre clé API :



Vous obtiendrez du code en retour. Sauvegardez alors temporairement ce code ou laissez la page web ouverte car nous en aurons besoin très prochainement.

Voyons maintenant les principaux éléments du sketch Arduino de cette partie. Le code en lui-même, assez complexe, nous est donné par Teleduino. Référez-vous au dépôt GitHub pour obtenir le code complet du projet.

Dans le sketch, on commence par inclure les bibliothèques requises pour le projet :

```
#include <EEPROM.h>
#include <Servo.h>
#include <Wire.h>
#include <Teleduino328.h>
#include <SPI.h>
#include <Ethernet.h>
```

Entrez ensuite l'adresse MAC de votre carte d'extension Ethernet pour Arduino. Vous la trouverez en dessous de la carte d'extension. Reportez-la dans la variable mac :

```
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xFE, 0x40 };
```

Nous pourrons, un peu plus loin dans le code, faire un copier/coller de la clé API au format hexadécimal :

```
byte key[] = { 0x64, 0x26, 0xFF, 0xC9,
0x20, 0x4C, 0xF2, 0xCF,
0xAE, 0x42, 0xD4, 0x1A,
0xED, 0x6C, 0xB0, 0xB7 };
```

On déclare à présent la broche à laquelle vous pouvez connecter une LED afin de contrôler le statut de Teleduino. Je n'ai pas effectué cette manipulation pour ce projet. Cependant, libre à vous de connecter une LED à cette broche :

```
byte statusLedPin = 8;
```

Déclarons également un client Ethernet dont nous nous servirons pour nous connecter à Teleduino :

```
EthernetClient Client;
```

À présent, initialisons la carte d'extension Ethernet dans la fonction `setup()` à l'aide de la fonction `begin()`. Si cette fonction aboutit (la carte d'extension Ethernet a donc obtenu une adresse IP), on réinitialise l'objet Teleduino :

```
if(!Ethernet.begin(mac))
{
  Teleduino328.setStatusLed(2, false, 10000);
  Teleduino328.reset();
}
```

Dans la fonction `loop()` du sketch, on vérifie tout d'abord que le client est bien connecté à Internet :

```
if(Client.connected())
```

Le reste du code est consacré au traitement des requêtes par Teleduino mais nous ne rentrerons pas dans les détails.

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/iot-book

Il est temps de tester le projet. Assurez-vous d'avoir bien utilisé le code issu du dépôt GitHub et d'avoir inséré l'adresse MAC de votre carte d'extension ainsi que la clé API à l'intérieur du code. À présent, vous pouvez transférer ce dernier vers la carte Arduino.

Patientez quelques instants et ouvrez votre navigateur web. La première étape consiste à définir la broche n° 7 en tant que sortie. Pour ce faire, il vous faut rentrer la commande appropriée dans votre navigateur web (veillez à remplacer `yourKey` par votre propre clé API) :

```
http://us01.proxy.teleduino.org/api/1.0/328.php?k=yourKey&
r=definePinMode&pin=7&mode=1
```

Après ça, vous devriez recevoir une confirmation dans votre navigateur web. Pour activer le relais, vous pouvez à présent taper la commande suivante :

```
http://us01.proxy.teleduino.org/api/1.0/328.php?k=yourKey&
r=setDigitalOutput&pin=7&output=1
```

La lampe devrait s'allumer instantanément. Pour désactiver une nouvelle fois le relais, entrez simplement :

```
http://us01.proxy.teleduino.org/api/1.0/328.php?k=yourKey&
r=setDigitalOutput&pin=7&output=0
```

Félicitations, vous pouvez désormais piloter cette lampe de n'importe quel endroit de la planète !

Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Assurez-vous que votre connexion Internet est active.
- Assurez-vous aussi d'avoir correctement saisi la clé API Teleduino.
- Enfin, assurez-vous de toujours utiliser la dernière version du code à partir du dépôt GitHub du livre.

Pour aller plus loin

Dans ce chapitre, nous avons conçu un projet permettant de piloter une lampe (ou tout autre équipement électrique chez vous) à partir de n'importe quel ordinateur ayant accès à Internet. Nous avons utilisé pour cela une carte d'extension Ethernet chargée de connecter le projet au web et au service Teleduino de façon à ce que vous puissiez le contrôler depuis un navigateur. Le service Teleduino est accessible dans le monde entier. Ainsi, vous pouvez choisir l'endroit à partir duquel vous voulez piloter votre lampe.

Vous pouvez partir de ce projet pour réaliser d'autres montages plus complexes. Vous pouvez tout à fait connecter plusieurs relais au projet pour prendre le contrôle de plusieurs équipements électriques à la fois. Connectez plusieurs lampes au projet par exemple, ainsi que d'autres appareils électriques tels qu'un chauffe-eau ou une machine à café.

☞

☞

À l'aide de Teleduino, vous pourrez aussi obtenir à distance les données de capteurs. Vous pouvez envisager par exemple d'ajouter au projet un capteur de courant dans le but de suivre à distance la consommation énergétique de votre lampe. Pour plus de détails sur la manière de procéder, je vous invite à prendre connaissance de la documentation sur l'interface de programmation Teleduino mise à disposition sur le site officiel : https://www.teleduino.org/documentation/api/328-full

14 Publier des relevés de mesures en ligne

Dans ce chapitre, nous allons mettre au point une station de mesures qui transmettra ses données sur le cloud de manière automatique. Pour cela, nous allons nous équiper de la puissante carte Arduino Yun et utiliser le service appelé Temboo.

Mise sur le marché par Arduino en 2013, la carte Yun dispose de la puissance de Linux ainsi que du WiFi intégré. Il est particulièrement simple de faire interagir cette carte avec le service Temboo que nous utiliserons dans ce chapitre. Nous programmerons un envoi automatique de données vers une feuille de calcul Google Sheets accessible de n'importe où.

MATÉRIEL ET LOGICIEL REQUIS

Pour ce projet, il vous faut donc une carte **Arduino Yun**. Concernant les mesures d'humidité, équipez-vous d'un **capteur DHT11** (ou DHT22) doté d'une résistance de 4,7 kΩ.

Pour les mesures de température et de pression, j'ai choisi un **capteur BMP085**. Vous pouvez également opter pour le capteur BMP180, plus récent, qui fonctionne avec la même bibliothèque.

Pour mesurer le niveau d'éclairage, j'ai choisi une **photorésistance** dotée d'une résistance de 10 kΩ. Enfin, je me suis muni d'une **plaque d'essai** ainsi que de fils de raccordement mâle/mâle.

LISTE DES COMPOSANTS

- **Arduino Yun** (http://www.francerobotique.com/contr%C3%B4leurs-interfaces/333-carte-arduino-yun.html)
- **Capteur DHT11 avec résistance de 4,7 kΩ** (http://snootlab.com/adafruit/255-capteur-de-temperateur-et-d-humidite-dht11-extras-.html)
- **Photorésistance** (http://snootlab.com/composants/97-photoresistance.html)
- **Résistance de 10 kΩ** (http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html)
- **Capteur de pression barométrique BMP180** (http://snootlab.com/adafruit/586-capteur-de-pression-bmp180-fr.html)
- **Fils de raccordement mâle/mâle** (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

☞

☞

- Plaque d'essai (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)

Côté logiciel, téléchargez la dernière version du logiciel Arduino IDE. Vous aurez également besoin de la bibliothèque DHT, de la bibliothèque BMP085 ou BMP180 ainsi que la bibliothèque Adafruit « Unified Sensor ».

Téléchargez le logiciel Arduino IDE à cette adresse :
http://arduino.cc/en/Main/Software# toc3

Téléchargez la bibliothèque DHT à cette adresse :
https://github.com/adafruit/DHT-sensor-library

Téléchargez la bibliothèque BMP085 ou BMP180 à cette adresse :
https://github.com/adafruit/Adafruit_BMP085_Unified

Téléchargez la bibliothèque Adafruit « Unified Sensor » à cette adresse :
https://github.com/adafruit/Adafruit_Sensor

Pour installer une bibliothèque, il vous suffit de placer son dossier dans **/libraries**, situé dans le dossier racine Arduino. Nous partons ici du principe que votre carte Arduino Yun est déjà connectée à votre réseau WiFi.

Si vous avez besoin d'aide dans ce domaine, consultez ce guide en anglais :
http://arduino.cc/en/Guide/ArduinoYun

Ce projet nécessite également un compte Google.

Pour créer votre compte, rendez-vous par exemple sur la page Google Drive :
https://drive.google.com/

CONFIGURER LE MATÉRIEL

Les connexions matérielles de ce projet sont assez simples. Nous devons connecter le capteur DHT11, le capteur de pression ainsi que la partie gérant la mesure du niveau d'éclairage. L'illustration suivante synthétise les connexions matérielles.

Commencez par installer les capteurs DHT11 et BMP sur la plaque d'essai. Raccordez la broche 5 V de la carte Arduino Yun à la rangée rouge de la plaque d'essai et la broche de masse à la rangée bleue.

Puis, connectez la broche n° 1 du capteur DHT11 (voir le schéma) à la rangée rouge de la plaque d'essai et la broche n° 4 (GND) à la rangée bleue. Après cette étape, connectez la broche n° 2 du capteur à la broche n° 8 de la carte Arduino. Pour en finir avec le branchement du capteur DHT11, insérez la résistance de 4,7 kΩ entre les broches n° 1 et n° 2 du capteur.

Connectez d'abord la photorésistance en série avec la résistance de 10 kΩ sur la plaque d'essai. L'autre patte de la photorésistance sera raccordée à la rangée rouge de la plaque d'essai et la seconde patte de la résistance à la masse. Enfin, reliez la broche commune à l'entrée analogique A0 de la carte Arduino.

Intéressons-nous désormais au capteur BMP085 ou BMP180. Connectez la broche VIN à la broche 5 V et la broche GND à la masse. Poursuivez en connectant la broche SCL à la broche n° 3 de la carte Arduino, puis la broche SDA à la broche n° 2.

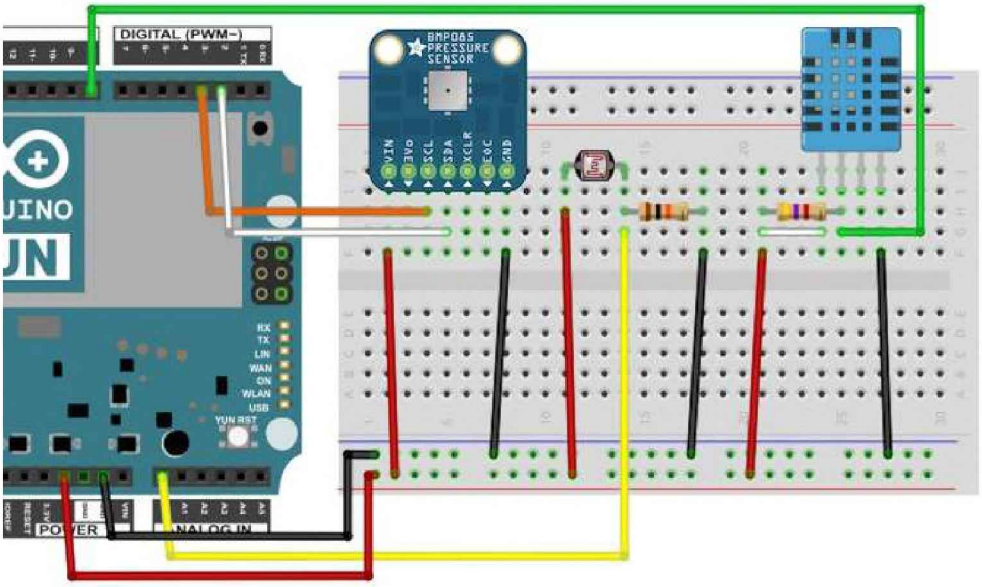
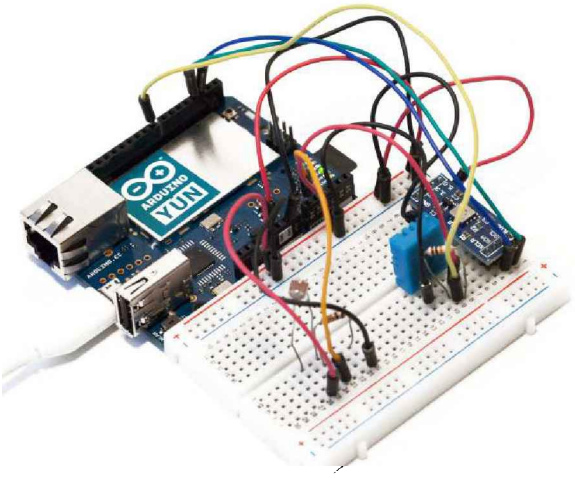


Figure 14.1 Image créée avec le logiciel Fritzing (http://fritzing.org/)

Voici un aperçu du projet assemblé :



TESTER LES CAPTEURS

Avant de procéder au partage du projet sur le cloud, testons les capteurs un par un. Pour ce faire, nous allons simplement écrire un sketch Arduino.

Vous trouverez ci-dessous la version complète du code de cette partie :

```
// Include required libraries
#include "DHT.h"
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP085_U.h>

// Variables
int lightLevel;
float humidity;
float temperature;

unsignedlong time;

// DHT11 sensor pins
#define DHTPIN 8
#define DHTTYPE DHT11

// DHT & BMP instances
DHT dht(DHTPIN, DHTTYPE);
Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);

void setup(void)
{

// Initialize DHT sensor
  dht.begin();

// Init serial
  Serial.begin(115200);

// Initialise the sensor
if(!bmp.begin())
  {
    Serial.print("Ooops, no BMP085 detected ...
                  Check your wiring or I2C ADDR!");
while(1);
  }

}

void loop(void)
{

// Measure the humidity
float humidity = dht.readHumidity();

// Measure light level
int lightLevel = analogRead(A0);

// Measure pressure & temperature from BMP sensor
```

```
    sensors_event_t event;
    bmp.getEvent(&event);
float pressure = event.pressure;

float temperature;
    bmp.getTemperature(&temperature);

float seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;
float altitude;
    altitude = bmp.pressureToAltitude(seaLevelPressure,
                                      event.pressure,
                                         temperature);

// Print measurements
    Serial.print("Humidity: ");
    Serial.println(humidity);
    Serial.print("Light level: ");
    Serial.println(lightLevel);
    Serial.print("Barometric pressure: ");
    Serial.println(pressure);
    Serial.print("Temperature: ");
    Serial.println(temperature);
    Serial.print("Altitude: ");
    Serial.println(altitude);
    Serial.println("");

// Repeat 50 ms
    delay(50);

}
```

Voyons maintenant ce code en détail. La première partie est dédiée à l'importation des bibliothèques :

```
#include "DHT.h"
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP085_U.h>
```

Il nous faut maintenant définir la broche du capteur DHT et la catégorie à laquelle il appartient :

```
#define DHTPIN 8
#define DHTTYPE DHT11
```

Créons maintenant des instances pour les capteurs DHT et BMP :

```
DHT dht(DHTPIN, DHTTYPE);
Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);
```

Dans la fonction `setup()`, on initialise le capteur BMP :

```
bmp.begin()
```

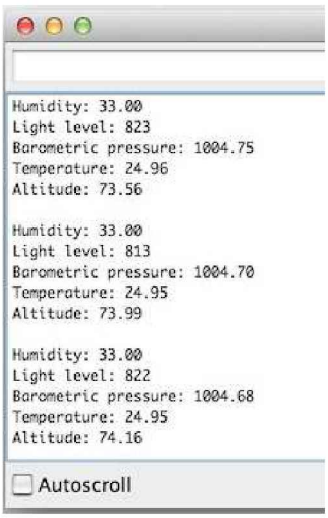
À présent, on programme la prise de mesure dans la fonction `loop()` du sketch :

```
// Measure the humidity
float humidity = dht.readHumidity();

// Measure light level
int lightLevel = analogRead(A0);

// Measure pressure & temperature from BMP sensor
sensors_event_t event;
bmp.getEvent(&event);
float pressure = event.pressure;
```

Les valeurs des mesures prises apparaîtront par la suite sur le moniteur série.

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/iot-book

Vous pouvez désormais transférer le sketch vers la carte et ouvrir le moniteur série (en veillant à ce que la vitesse série corresponde à celle définie dans le code). Vous devriez voir apparaître ceci :



Si vous parvenez à ce résultat, vous pouvez en déduire que votre matériel a été correctement assemblé et ainsi passer à la partie suivante.

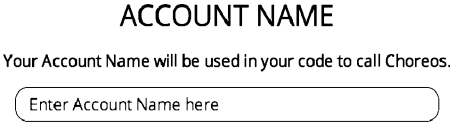
Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Premièrement, que toutes les bibliothèques Arduino nécessaires au projet sont correctement installées.
- Assurez-vous également que les instructions relatives à l'assemblage du projet ont bien été suivies.

METTRE EN PLACE VOTRE COMPTE TEMBOO

Le transfert de données vers Google Sheets nécessite au préalable la création d'un compte Temboo. Rendez-vous sur le site Temboo et entrez votre adresse e-mail pour lancer le processus de création de votre compte :

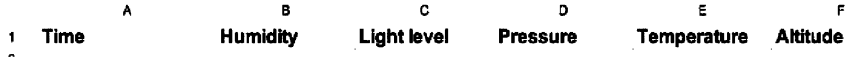
Il vous sera également demandé d'attribuer un nom à votre compte.



Après cela, sélectionnez l'onglet « Account » puis « Applications ». Vous y retrouverez le nom de votre première application (qui devrait être par défaut « myFirstApp ») ainsi que votre clé API. Conservez à proximité votre identifiant, le nom de votre application et sa clé API car ces informations vous seront bientôt demandées.

STOCKER DES DONNÉES DANS GOOGLE SHEETS

Nous allons maintenant configurer la partie logicielle du projet. On commence alors par effectuer quelques manipulations sur Google Sheets. Créez une nouvelle feuille de calcul Google Sheets à laquelle vous attribuerez un nom (j'ai nommé le mien « Yun »). Comme le montre cette image, il vous faut donner un titre à chacune des colonnes :



Il vous faut maintenant obtenir une jeton d'accès pour Google Drive.

Pour cela, suivez la procédure que vous trouverez à :
https://www.temboo.com/library/Library/Google/Drive/

Au final, vous obtiendrez une clé API, une clé API secret, et un jeton d'accès. Vous aurez besoin de ces données un peu plus loin dans ce chapitre.

Traitons maintenant le sketch Arduino. Vous trouverez ci-dessous la version complète du code de cette partie :

```
// Include required libraries
#include <Bridge.h>
#include <Temboo.h>
#include <Process.h>
#include <Wire.h>
```

```
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP085_U.h>

// Contains Temboo account information
#include "TembooAccount.h"

// Variables
int lightLevel;
float humidity;
float temperature;
unsigned long time;

// Process to get the measurement time
Process date;

// Your Google Docs data
const String GOOGLE_CLIENT_ID = "your-google-client-id";
const String GOOGLE_CLIENT_SECRET = "your-google-client-secret";
const String GOOGLE_REFRESH_TOKEN = "your-google-refresh-token";

const String SPREADSHEET_TITLE = "your-spreadsheet-title";

// Include required libraries
#include "DHT.h"

// DHT11 sensor pins
#define DHTPIN 8
#define DHTTYPE DHT11

// DHT & BMP instances
DHT dht(DHTPIN, DHTTYPE);
Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);

// Debug mode ?
boolean debug_mode = false;

void setup() {

  // Start Serial
  if (debug_mode == true){
    Serial.begin(115200);
    delay(4000);
    while(!Serial);
  }

  // Initialize DHT sensor
  dht.begin();

  // Start bridge
  Bridge.begin();

  // Start date process
```

```
  time = millis();
  if (!date.running())  {
    date.begin("date");
    date.addParameter("+%D-%T");
    date.run();
  }

  if (debug_mode == true){
    Serial.println("Setup complete. Waiting for sensor
input...\n");
  }
  // Initialise the sensor
  if(!bmp.begin())
  {
    while(1);
  }
}

void loop() {
  // Measure the humidity & temperature
  humidity = dht.readHumidity();

  // Measure light level
  int lightLevel = analogRead(A0);
  // Measure pressure & temperature from BMP sensor
  sensors_event_t event;
  bmp.getEvent(&event);
  float pressure = event.pressure;

   bmp.getTemperature(&temperature);

   float seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;
   float altitude;
   altitude = bmp.pressureToAltitude(seaLevelPressure,
                                     event.pressure,
                                     temperature);

  if (debug_mode == true){
    Serial.println("\nCalling the AppendRow Choreo...");
  }

  // Append data to Google Docs sheet
  runAppendRow(humidity, lightLevel, pressure, temperature,
altitude);

  // Repeat every 10 minutes
  delay(600000);
}

// Function to add data to Google Docs
void runAppendRow(float humidity, int lightLevel,
  float pressure, float temperature, float altitude) {
```

```
  TembooChoreo AppendRowChoreo;

  // Invoke the Temboo client
  AppendRowChoreo.begin();

  // Set Temboo account credentials
  AppendRowChoreo.setAccountName(TEMBOO_ACCOUNT);
  AppendRowChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
  AppendRowChoreo.setAppKey(TEMBOO_APP_KEY);

  // Identify the Choreo to run
  AppendRowCho-
reo.setChoreo("/Library/Google/Spreadsheets/AppendRow");

  // your Google Client ID
  AppendRowChoreo.addInput("ClientID", GOOGLE_CLIENT_ID);

  // your Google Client Secret
  AppendRowChoreo.addInput("ClientSecret", GOOGLE_CLIENT_SECRET);

  // your Google Refresh Token
  AppendRowChoreo.addInput("RefreshToken", GOOGLE_REFRESH_TOKEN);

  // the title of the spreadsheet you want to append to
  AppendRowChoreo.addInput("SpreadsheetTitle", SPREADSHEET_TITLE);

  // Restart the date process:
  if (!date.running())  {
    date.begin("date");
    date.addParameter("+%D-%T");
    date.run();
   }

  // Get date
  String timeString = date.readString();

  // Format data
  String data = "";
  data = data + timeString + "," +
  String(humidity) + "," +
  String(lightLevel) + "," +
  String(pressure) + "," +
  String(temperature) + "," +
  String(altitude);

  // Set Choreo inputs
  AppendRowChoreo.addInput("RowData", data);

  // Run the Choreo
  unsigned int returnCode = AppendRowChoreo.run();

  // A return code of zero means everything worked
```

```
  if (returnCode == 0) {
    if (debug_mode == true){
      Serial.println("Completed execution of the AppendRow
Choreo.\n");
    }
  } else {
    // A non-zero return code means there was an error
    // Read and print the error message
    while (AppendRowChoreo.available()) {
      char c = AppendRowChoreo.read();
      if (debug_mode == true){ Serial.print(c); }
    }
    if (debug_mode == true){ Serial.println(); }
  }
  AppendRowChoreo.close();
}
```

Voyons maintenant ce code en détail. Il vous faut importer toutes les bibliothèques pour les fonctionnalités Internet et « Bridge » de la carte Arduino Yun :

```
#include <Bridge.h>
#include <Temboo.h>
#include <Process.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP085_U.h>
```

J'ai aussi rentré les informations liées à mon compte Temboo dans un fichier séparé nommé Temboo.h. Assurez-vous d'avoir mis à jour dans ce fichier les informations personnelles de votre compte. Vous trouverez ci-dessous le contenu de ce fichier :

```
#define TEMBOO_ACCOUNT "tembooAccountName"
      // Your Temboo account name
#define TEMBOO_APP_KEY_NAME "myFirstApp"
      // Your Temboo app key name
#define TEMBOO_APP_KEY "tembooAccountKey"
      // Your Temboo app key
```

Nous devons maintenant les introduire dans un sketch :

```
#include "TembooAccount.h"
```

Poursuivez en définissant les informations liées à votre compte Google :

```
const String GOOGLE_CLIENT_ID = "your-google-client-id";
const String GOOGLE_CLIENT_SECRET = "your-google-client-secret";
const String GOOGLE_REFRESH_TOKEN = "your-google-refresh-token";
```

Notez que si vous choisissez la validation en deux étapes de Google, il vous faudra alors un mot de passe spécifique à l'application. Vous pouvez en obtenir un ici :

https://security.google.com/settings/security/apppasswords

Dans la fonction `setup()`, on initialise la bibliothèque Bridge permettant de faire le lien entre la partie Arduino et la machine Linux :

```
Bridge.begin();
```

Nous devons aussi démarrer un processus nommé « date » pour envoyer de manière automatique les données de temps vers Google Sheets :

```
time = millis();
if (!date.running())  {
  date.begin("date");
  date.addParameter("+%D-%T");
  date.run();
}
```

Dans la fonction `loop()`, voici la ligne la plus importante :

```
runAppendRow(humidity, lightLevel, pressure, temperature,
altitude);
```

Cette fonction appelle une autre fonction qui transmet automatiquement les données vers la feuille de calcul Google Sheets. Nous ne la verrons pas en détail mais vous pouvez bien sûr retrouver l'ensemble des codes dans le dépôt GitHub du livre. La fonction `loop()` du sketch Arduino est répétée toutes les 10 minutes grâce à une fonction `delay()`. En effet, étant donné que les grandeurs que l'on mesure dans ce projet n'évoluent que lentement en général, il n'est pas utile de recourir à un envoi fréquent de données.

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/iot-book

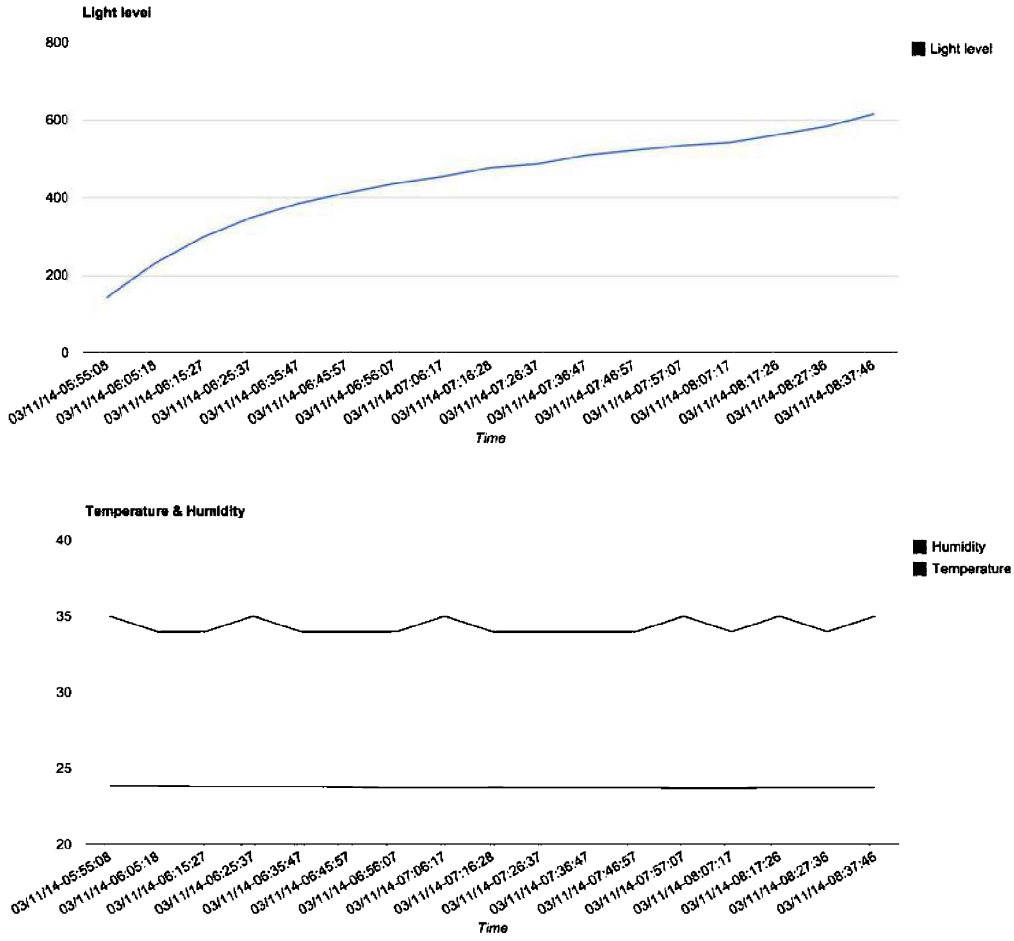
Testons enfin notre projet. Vérifiez que le contenu des fichiers a été mis à jour selon vos données et que le code a bien été transféré vers la carte Arduino Yun. Ensuite, ouvrez une nouvelle fois la feuille de calcul Google Sheets dans votre navigateur et patientez quelques instants. Les premières valeurs devraient alors rapidement apparaître sur la feuille de calcul :

	A	B	C	D	E	F
1	**Time**	**Humidity**	**Light level**	**Pressure**	**Temperature**	**Altitude**
2	03/11/14-08:47:56	34	653	1002.54	23.77	92.43

Je me suis servi des fonctions d'insertion de graphique de Google Sheets pour faire apparaître au fur et à mesure mes données sous forme de graphique. J'ai utilisé par exemple le type de diagramme basique appelé « Courbe » pour le niveau d'éclairage.

Voici les résultats obtenus durant la matinée après plusieurs heures. On remarque une progression lente de l'intensité de la lumière.

J'ai fait apparaître l'évolution de la température et de l'humidité sous forme de graphique sur la même période.



Le gros avantage de cette fonctionnalité est de pouvoir suivre l'évolution sur une certaine durée. De plus, vous y avez accès depuis n'importe où. Vous devez seulement vous munir de votre nom d'utilisateur Google et de votre mot de passe.

Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Assurez-vous d'abord que votre carte Arduino Yun est correctement configurée et qu'elle peut se connecter à Internet.
- Vérifiez aussi que votre feuille de calcul Google Sheets a bien été mise en place.
- Enfin, assurez-vous d'avoir correctement saisi vos identifiants Temboo et Google à l'intérieur des fichiers Arduino après les avoir téléchargés à partir du dépôt GitHub du livre.

Pour aller plus loin

Résumons ce que nous avons vu dans ce chapitre. Vous venez d'apprendre à envoyer des données vers Google Sheets depuis votre carte Arduino Yun et à les rendre par la même occasion accessibles depuis n'importe quel ordinateur connecté à Internet. Nous avons vu également comment afficher ces données sous forme de graphique, rendant ainsi possible leur suivi en direct depuis un navigateur web.

Vous pouvez partir de ce projet pour réaliser une multitude de montages plus complexes. Un premier montage consisterait à ajouter d'autres capteurs au projet, tels qu'un détecteur de vitesse du vent ou de fumée. Il vous est possible, bien entendu, d'utiliser plusieurs cartes Arduino Yun dans différentes pièces de votre logement.

15 Installer une caméra de surveillance sans fil

Connaissez-vous ces caméras de surveillance sans fil qui sont vendues dans le commerce ? Ces appareils peuvent être installés en intérieur ou en extérieur et reliés à votre réseau WiFi. En règle générale, ces caméras se déclenchent automatiquement lorsque quelque chose bouge devant l'objectif. Cependant, elles réagissent en fonction de ce qui a été programmé par le constructeur, ce qui limite leurs fonctionnalités.

Nous verrons dans ce chapitre comment mettre au point une version « *Do It Yourself* » de ce type d'équipement. Nous allons connecter l'élément de base du projet, la carte Arduino Yun, à une webcam USB ainsi qu'à un détecteur de mouvement PIR afin de créer une application de l'Internet des objets.

Cette application est l'équivalent moderne d'une fonctionnalité courante des caméras de surveillance : prendre des clichés lorsque des mouvements sont détectés. Ce projet stockera notamment des images prises par la caméra sur une carte SD branchée sur la carte Arduino. Puisque ce livre traite entre autres de l'Internet des objets, nous verrons aussi comment stocker ces images et les transférer vers un endroit sécurisé. Pour ce projet, nous avons choisi Dropbox.

MATÉRIEL ET LOGICIEL REQUIS

Assurez-vous d'abord que vous disposez des bons composants matériels. Commencez par vous équiper d'une carte **Arduino Yun** et d'un détecteur de mouvement PIR.

En ce qui concerne la **caméra USB**, choisissez un modèle compatible avec le protocole UVC, ce qui est le cas de la plupart des caméras vendues actuellement dans le commerce. J'ai opté pour le modèle C270 de la marque Logitech, doté d'une résolution 720p.

> Le lien suivant vous dirigera vers une liste de caméras compatibles :
> http://en.wikipedia.org/wiki/List_of_USB_video_class_devices

Procurez-vous également une **carte microSD** afin de stocker localement les images sur votre carte Arduino, ainsi que des fils de raccordement mâle/femelle pour le branchement du détecteur de mouvement PIR.

LISTE DES COMPOSANTS

- **Arduino Yun** (http://www.francerobotique.com/contr%C3%B4leurs-interfaces/333-carte-arduino-yun.html)

☞

☞

- Webcam USB (http://www.darty.com/nav/achat/informatique/webcam_haut_parleur_casque/webcam/logitech_c270hd.html)
- Détecteur de mouvement PIR (http://snootlab.com/adafruit/285-capteur-de-presence-pir.html)
- Carte MicroSD (http://www.darty.com/nav/achat/accessoires/accessoire_appareil_photo-carte_memoire/carte/sandisk_msd_ultra_8gb_new.html)
- Fils de raccordement mâle/femelle (http://snootlab.com/cables/23-kit-10-cordons-6-m-f.html)

Avant de pouvoir mettre au point des applications ingénieuses à l'aide de notre matériel, vous devez créer un compte sur chacun des sites que nous allons utiliser. Commencez par créer un compte Temboo (si vous n'en avez pas encore). Temboo jouera le rôle d'interface entre la carte Arduino et Dropbox.

Rendez-vous à l'adresse suivante : https://www.temboo.com/

Suivez les instructions. Prenez note de vos identifiants, du nom de votre application et de celui de votre clé API. Ces informations vous seront demandées prochainement.

Il vous reste à télécharger le kit de développement logiciel (SDK) Temboo pour le langage Python. Nous nous en servirons pour transférer des images vers Dropbox depuis la carte Arduino Yun.

Vous pouvez l'obtenir ici : https://www.temboo.com/python

Une fois le fichier téléchargé, il vous faut extraire le dossier « temboo » vers la racine de votre carte microSD.

Vous aurez également besoin d'un compte Dropbox.

Pour créer un compte, rendez-vous à cette adresse : https://www.dropbox.com/fr/

Vous pouvez désormais développer une application permettant à la carte Arduino de synchroniser des images vers votre dossier Dropbox.

Pour cela, suivez ce lien :
https://www.dropbox.com/login?cont=https%3A%2F%2Fwww.dropbox.com%2Fdevelopers%2Fapps

Cliquez ensuite sur « Create app » et choisissez le type d'application que vous souhaitez créer (sélectionnez ici « Dropbox API app ») :

Puis, cliquez sur « Files and datastores ».

What type of data does your app need to store on Dropbox?

(•) Files and datastores

( ) Datastores only

Après cette étape, nommez votre application et créez-la. Il est nécessaire de préciser que votre application n'est autorisée à accéder qu'aux fichiers de son propre dossier.

Vous devez maintenant récupérer toutes les clés pour votre application Dropbox. Vous allez devoir les saisir dans la partie du chapitre consacrée à la configuration logicielle. Notez les données des champs suivants qui apparaissent à l'écran : « App Key » et « App Secret ».

Il vous faut également récupérer les données suivantes : « Token Key » et « Token Secret ».

> Pour cela, rendez-vous d'abord sur la page « InitialiseOAuth Choreo » du site Temboo : https://temboo.com/library/Library/Dropbox/OAuth/InitializeOAuth/

Les données « App Key » et « App Secret » vous seront alors demandées. Cette opération générera d'autres informations comme l'attribut « callback ID » ainsi qu'un jeton temporaire « Token Secret ». Vous serez redirigé vers Dropbox pour finaliser la procédure d'identification. Pour finir, accédez à la page « FinalizeOAuth » pour mettre un terme à la procédure. Vous devrez entrer les données suivantes : « App Key », « App Secret », « callback ID » et le jeton temporaire « Token Secret » :

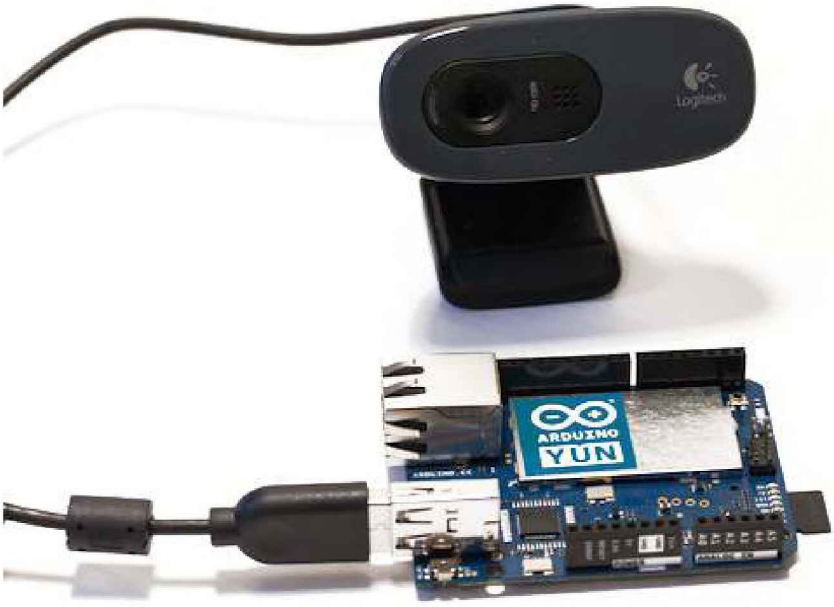
> https://temboo.com/library/Library/Dropbox/OAuth/FinalizeOAuth/

Vous obtiendrez vos identifiants « Token Key » et « Token Secret » définitifs. Prenez-en note, vous en aurez besoin prochainement.

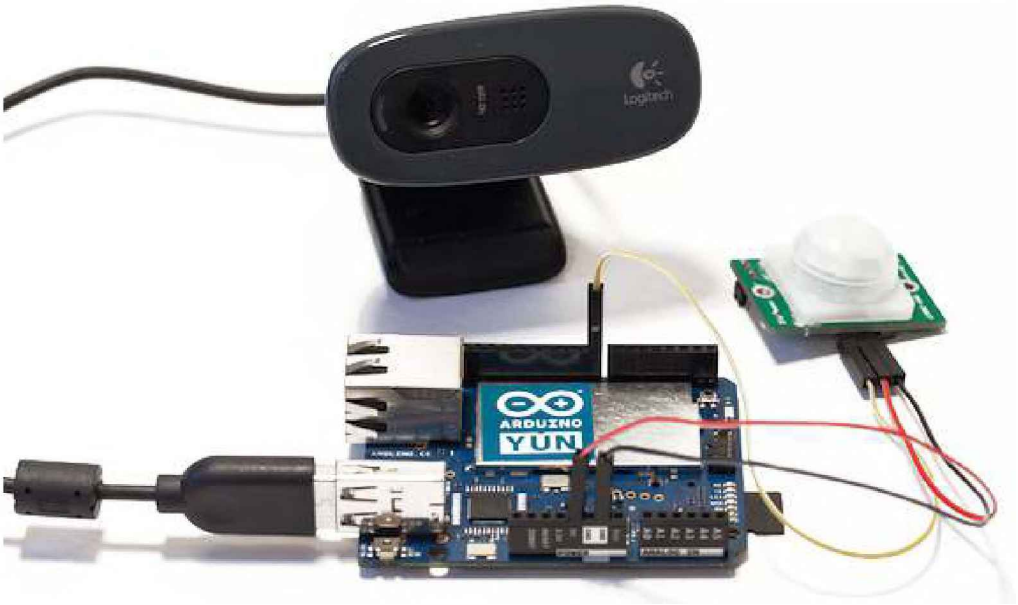
CONNECTER UNE CAMÉRA USB À LA CARTE ARDUINO YUN

Insérez premièrement la carte microSD dans l'emplacement de la carte Arduino Yun :

Raccordez ensuite la caméra au port USB de la carte Yun :



Enfin, connectez le détecteur de mouvement à la carte Yun. Il vous faut connecter respectivement les broches du connecteur à la carte Arduino : VCC à la broche 5 V, GND à la broche GND et SIG à la broche n° 8 :



Il ne vous reste plus qu'à connecter le projet à votre ordinateur *via* le port micro USB.

TESTER LA CAMÉRA

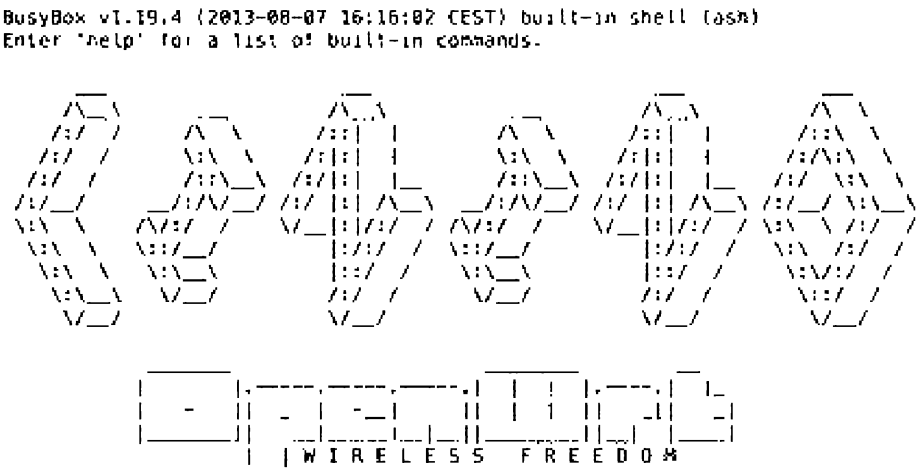
Avant de nous lancer dans la partie logicielle, il est nécessaire d'installer d'autres programmes sur la carte Arduino. Nous allons aussi tester la caméra USB pour vérifier le bon fonctionnement des drivers. Notez que pour le reste du projet, votre carte Arduino doit être activée et connectée au réseau local.

> Si vous utilisez la carte Arduino Yun pour la première fois, voici un lien (en anglais) qui peut vous aider : http://arduino.cc/en/Guide/ArduinoYun

Commençons par les drivers UVC pour la caméra. Pour les installer, vous devrez connecter votre carte Arduino *via* SSH. Ouvrez un terminal et saisissez :

```
ssh root@yourYunName.local
```

Inscrivez à présent le nom donné à votre carte Arduino lors de sa première utilisation. Il vous sera également demandé d'entrer votre mot de passe. Si la connexion est établie, vous obtiendrez une image d'art ASCII :



Nous pouvons à présent installer les paquets (*packages*) demandés. Mettons à jour le gestionnaire de paquets :

```
opkg update
```

Installons ensuite les drivers UVC :

```
opkg install kmod-video-uvc
```

Puis le paquet « python-openssl » :

```
opkg install python-openssl
```

Nous utiliserons aussi le programme `fswebcam` pour capturer des images du terminal :

```
opkg install fswebcam
```

Testons à présent la webcam. La carte SD se trouvant dans l'emplacement de la carte Arduino prévu à cet effet, parcourez son dossier à l'aide de cette commande :

```
cd /mnt/sda1
```

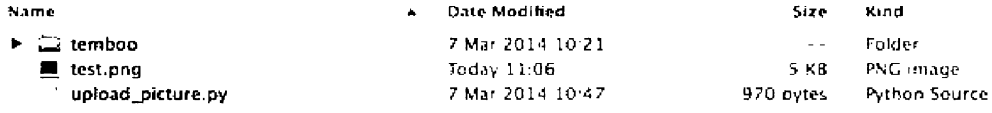
Pour tester la caméra et capturer une image, entrez simplement :

```
fswebcam test.png
```

Des informations apparaissent à l'écran incluant quelques messages d'erreur. N'en tenez pas compte puisque seules ces lignes nous intéressent :

```
— Opening /dev/video0...
Trying source module v4l2...
/dev/video0 opened.
```

Vérifiez que l'image a été correctement capturée en ouvrant son fichier sur votre ordinateur après avoir retiré la carte SD de la carte Arduino. L'image devrait apparaître à la racine de la carte SD :



Ouvrez-la pour contrôler sa qualité. Si vous êtes satisfait du rendu de l'image, vous pouvez passer à la section suivante et créer des applications intéressantes grâce au projet.

Si vous rencontrez des problèmes à ce stade, la première chose à faire est de vous assurer que la caméra est bien connectée à votre carte Arduino. Vérifiez ensuite la compatibilité de votre caméra avec le protocole UVC. La carte SD que vous utilisez doit être correctement formatée. Pour le savoir, tentez d'y accéder depuis votre ordinateur avant d'aborder le projet.

CAPTURER DES IMAGES À DISTANCE

Notre but est premièrement de programmer la capture d'images à chaque fois qu'un mouvement est détecté par le détecteur de mouvement PIR. Deuxièmement, nous stockerons ces images localement sur la carte SD avant de les envoyer sur Dropbox. Écrivons pour cela un code en deux parties.

La première n'est autre qu'un script Python chargé d'établir la connexion à Dropbox, de capturer des images et de les sauvegarder sur la carte SD, puis de les transférer sur Dropbox. Python facilite la communication avec Dropbox.

La seconde partie contient le sketch Arduino qui effectuera des requêtes auprès du script Python afin de capturer des images *via* la bibliothèque « Bridge » de la carte Arduino.

Programmons d'abord le script Python.

Vous trouverez ci-dessous la version complète du code de cette partie :

```
# Import correct libraries
import base64
import sys
from temboo.core.session import TembooSession
from temboo.Library.Dropbox.FilesAndMetadata import UploadFile

print str(sys.argv[1])

# Encode image
with open(str(sys.argv[1]), "rb") as image_file:
encoded_string = base64.b64encode(image_file.read())

# Declare Temboo session and Choreo to upload files
session = TembooSession('yourSession', 'yourApp', 'yourKey')
uploadFileChoreo = UploadFile(session)

# Get an InputSet object for the choreo
uploadFileInputs = uploadFileChoreo.new_input_set()

# Set inputs
uploadFileInputs.set_AppSecret("yourAppSecret")
uploadFileInputs.set_AccessToken("yourAccessToken")
uploadFileInputs.set_FileName(str(sys.argv[1]))
uploadFileInputs.set_AccessTokenSecret("yourTokenSecret")
uploadFileInputs.set_AppKey("yourAppKey")
uploadFileInputs.set_FileContents(encoded_string)
uploadFileInputs.set_Root("sandbox")

# Execute choreo
uploadFileResults =
uploadFileChoreo.execute_with_results(uploadFileInputs)
```

Regardons ce script de plus près. Premièrement, on inclut les bibliothèques du kit de développement logiciel (SDK) Temboo pour Python :

```
from temboo.core.session import TembooSession
from temboo.Library.Dropbox.FilesAndMetadata import UploadFile
```

Le script Python prendra aussi comme paramètre le nom de l'image que nous souhaitons envoyer vers Dropbox :

```
withopen(str(sys.argv[1]), "rb") as image_file:
encoded_string = base64.b64encode(image_file.read())
```

Vous avez auparavant obtenu des identifiants Temboo. C'est maintenant qu'ils vont nous être utiles :

```
session = TembooSession('yourTembooName', 'yourTembooApp'
                                        , 'yourTembooKey')
```

À présent, il nous est possible de créer la bibliothèque Dropbox chargée de transférer des fichiers (les bibliothèques s'appellent **Choreo** sur Temboo) :

```
uploadFileChoreo = UploadFile(session)
uploadFileInputs = uploadFileChoreo.new_input_set()
```

Saisissez maintenant les données liées à votre compte Dropbox : votre « App Key » et votre « App Secret » ainsi que vos « Access Token » et « Access Token Secret » :

```
uploadFileInputs.set_AppSecret("appSecret")
uploadFileInputs.set_AccessToken("accessToken")
uploadFileInputs.set_FileName(str(sys.argv[1]))
uploadFileInputs.set_AccessTokenSecret("accessTokenSecret")
uploadFileInputs.set_AppKey("appKey")
uploadFileInputs.set_FileContents(encoded_string)
uploadFileInputs.set_Root("sandbox")
```

Pour finir, transférez le fichier vers Dropbox :

```
uploadFileResults =
uploadFileChoreo.execute_with_results(uploadFileInputs)
```

Sauvegardez ce code à l'intérieur d'un fichier nommé « `upload_picture.py` ».

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/iot-book

Intéressons-nous maintenant au sketch Arduino.
Voici la version complète du code pour cette partie :

```
// Sketch to upload pictures to Dropbox when motion is detected
#include <Bridge.h>
#include <Process.h>

// Picture process
Process picture;

// Filename
String filename;

// Pin
int pir_pin = 8;

// Path
String path = "/mnt/sda1/";

void setup() {

// Bridge
  Bridge.begin();

// Set pin mode
  pinMode(pir_pin,INPUT);
}
```

```
void loop(void)
{

if (digitalRead(pir_pin) == true) {

// Generate filename with timestamp
    filename = "";
    picture.runShellCommand("date +%s");
while(picture.running());

while (picture.available()>0) {
char c = picture.read();
      filename += c;
    }
    filename.trim();
    filename += ".png";

// Take picture
    picture.runShellCommand("fswebcam " + path + filename
                            + " -r 1280x720");
while(picture.running());

// Upload to Dropbox
    picture.runShellCommand("python " + path + "upload_picture.py "
                           + path + filename);
while(picture.running());
  }
}
```

On inclut d'abord les bibliothèques requises :

```
#include <Bridge.h>
#include <Process.h>
```

On déclare ensuite une procédure visant à recueillir des fonctions sur la machine Linux de la carte Arduino (comme le programme `fswebcam` par exemple) :

```
Process picture;
```

Il nous faut nommer chaque image que le projet produira. Ce nom sera stocké au sein d'une séquence de caractères (`String`) :

```
String filename;
```

Il nous faut également définir la broche sur laquelle le détecteur de mouvement PIR est connecté :

```
int pir_pin = 8;
```

Indiquons le chemin d'accès de la carte SD branchée sur la carte Yun :

```
String path = "/mnt/sda1/";
```

Nous devons effectuer des requêtes de fonctions sur la machine Linux de la carte Yun. Il est donc nécessaire de lancer la bibliothèque « Bridge » :

```
Bridge.begin();
```

Dans la fonction `loop()`, nous allons voir si le détecteur de mouvement PIR a décelé quelque chose :

```
if (digitalRead(pir_pin) == true) {
```

Si c'est le cas, on donne un nom de fichier unique à l'image, incluant la date à laquelle elle a été prise :

```
filename = "";
picture.runShellCommand("date +%s");
while(picture.running());

while (picture.available()>0) {
char c = picture.read();
  filename += c;
}
filename.trim();
filename += ".png";
```

On continue en effectuant une requête vers la machine Linux de la carte Arduino pour capturer une image avec le programme `fswebcam`. Notez que nous fournirons ici un paramètre supplémentaire à la commande `-r` qui contrôle la résolution. Utilisez la résolution maximale de la caméra, la résolution 720p :

```
picture.runShellCommand("fswebcam " + path + filename
                                    + " -r 1280x720");
while(picture.running());
```

Effectuons désormais une seconde requête vers la machine Linux en impliquant cette fois le script Python avec le nom de l'image en tant que paramètre, ce qui transférera l'image vers Dropbox :

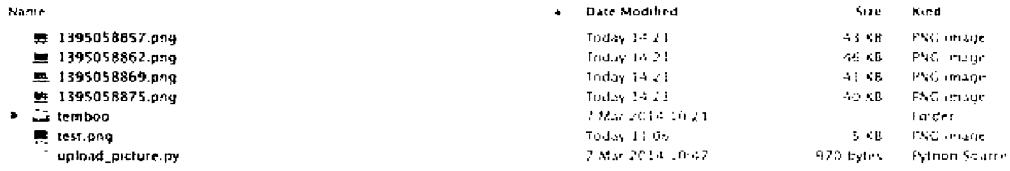
```
picture.runShellCommand("python " + path + "upload_picture.py "
                                  + path + filename);
while(picture.running());
```

Il nous faut maintenant voir si le projet fonctionne.

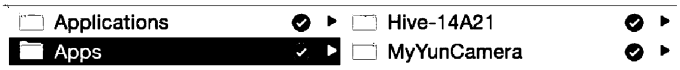
Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/iot-book

Placez tout d'abord le fichier Python sur la racine de la carte SD et réinsérez cette dernière dans l'emplacement de la carte Yun. Après cela, chargez le sketch Arduino sur la carte. Essayez maintenant de déclencher le détecteur de mouvement en passant votre main devant par exemple. La webcam devrait s'enclencher peu de temps après (dans mon cas, le voyant passe au vert lorsqu'elle est active).

Afin de vérifier le bon fonctionnement du projet, inspectez le contenu de votre carte SD après avoir patienté quelques instants :



Les mêmes images devraient apparaître dans votre dossier Dropbox. Elles sont normalement stockées dans le dossier « Apps » :



Si vous n'obtenez pas ce résultat à ce stade, vérifiez les points suivants :

- Vérifiez en premier lieu que votre carte Arduino Yun est correctement configurée et qu'elle est connectée à Internet.
- Assurez-vous ensuite que votre application Dropbox a bien été configurée et que vous avez obtenu les clés API requises pour votre compte Dropbox.
- Pour terminer, vérifiez l'exactitude des identifiants Temboo et Dropbox que vous avez entrés à l'intérieur des fichiers Arduino.

POUR ALLER PLUS LOIN

Grâce à ce chapitre, vous avez appris à connecter une caméra USB à votre carte Arduino Yun. Nous avons créé une caméra de surveillance qui envoie des images sur Dropbox de manière automatique lorsque des mouvements sont détectés.

Il est possible, à partir de ce projet, par exemple d'ignorer la partie dédiée au détecteur de mouvement et de mettre au point une caméra qui prendra des clichés instantanés à intervalles réguliers avant de les poster sur Dropbox. Avec ce type de projet, vous pouvez aussi créer des vidéos en accéléré : il vous suffit de rassembler les images de votre dossier Dropbox et de les copier dans un logiciel vidéo doté de la fonction « accéléré ». Ce projet peut être complété par l'ajout de cartes Arduino Yun et de caméras supplémentaires pour obtenir au final un système complet de surveillance de votre domicile.

16 Organiser un arrosage automatique en fonction de la météo

Ce chapitre est consacré à l'usage de la domotique en extérieur. Nous ferons interagir un capteur d'humidité et de température du sol avec Arduino et une puce WiFi. L'objectif ici est de mettre à disposition sur le cloud et de manière instantanée des mesures prises en extérieur.

Le service Carriots se chargera de traiter les données avant de les afficher de manière simple et pratique sur une page web. Vous pourrez recevoir une alerte mail ou SMS instantanément lorsque l'humidité du sol passe sous une valeur donnée.

Si vous ne possédez ni jardin, ni plantes, ce n'est pas un problème. Ce projet peut servir à tout type de montage impliquant des relevés de mesure à distance.

MATÉRIEL ET LOGICIEL REQUIS

Dans le cadre de ce projet, je me suis servi, une fois de plus, d'une carte **Arduino Uno**. Équipez-vous ensuite d'une carte dotée d'une puce CC3000. Je vous conseille une nouvelle fois d'utiliser le module WiFi CC3000.

Afin de mesurer le niveau d'humidité et la température du sol, vous avez besoin d'un équipement adapté. J'ai pour ma part choisi un **capteur d'humidité et de température** Adafruit, équipé du circuit intégré SHT10 de la marque Sensirion. Cet élément s'adapte parfaitement à la plateforme Arduino. En effet, il possède sa propre bibliothèque ce qui facilite la communication avec notre projet.

Pour finir, vous aurez besoin d'une **plaque d'essai** et de quelques fils de raccordement pour effectuer les différents branchements.

LISTE DES COMPOSANTS

- Carte Arduino Uno (http://snootlab.com/arduino/142-arduino-uno-rev3.html)
- Capteur DHT11 avec résistance de 4,7 kΩ (http://snootlab.com/adafruit/255-capteur-de-temperateur-et-d-humidite-dht11-extras-.html)
- Photorésistance (http://snootlab.com/composants/97-photoresistance.html)
- Résistance de 10 kΩ (http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html)

☞

☞

- Plaque d'essai (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- Fils de raccordement (http://snootlab.com/cables/20-kit-10-cordons-6-m-m.html)

Côté logiciel, téléchargez la dernière version du logiciel Arduino IDE. Vous aurez besoin de la bibliothèque pour la puce WiFi CC3000. Vous aurez aussi besoin de la bibliothèque Sensirion.

Téléchargez le logiciel Arduino IDE à cette adresse :
http://arduino.cc/en/Main/Software# toc3
Téléchargez la bibliothèque pour la puce WiFi CC3000 à cette adresse :
https://github.com/adafruit/Adafruit_CC3000_Library
Téléchargez la bibliothèque Sensirion à cette adresse :
http://playground.arduino.cc/Code/Sensirion

Veillez à placer ces bibliothèques sous **/libraries**, situé dans le dossier racine Arduino. Vous pouvez retrouver le code de ce projet dans le dépôt GitHub.

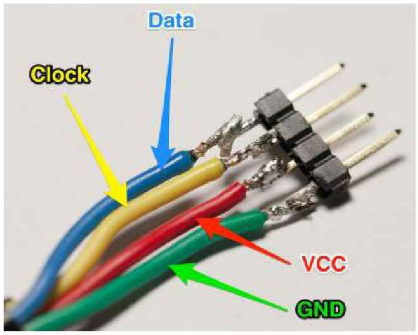
CONFIGURER LE MATÉRIEL

Il vous faut à présent connecter les broches de la puce CC3000 à la carte Arduino Uno. Vous trouverez dans les chapitres précédents les instructions concernant la connexion de la carte Arduino Uno au module CC3000.

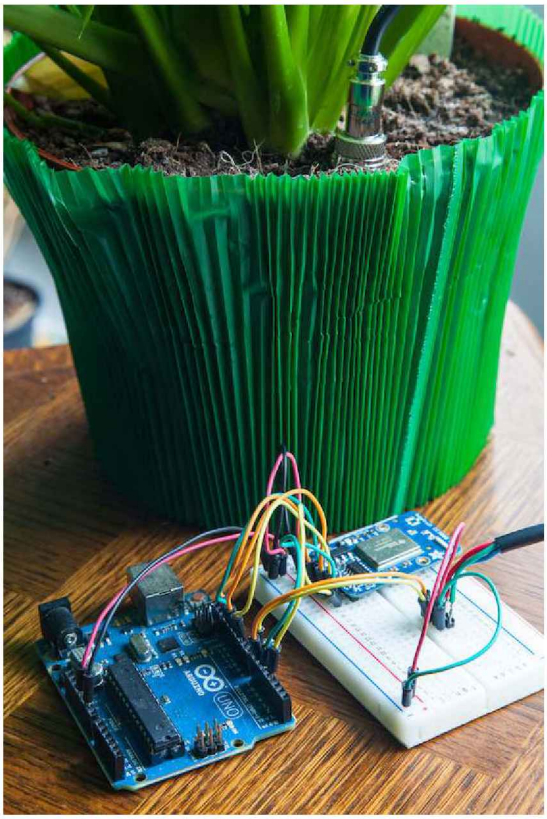
Pour résumer, il vous faut connecter la broche IRQ du module à la broche n° 3 de la carte Arduino, puis la broche VBAT à la broche n° 5 et enfin la broche CS à la broche n° 10. Créons à présent une interaction entre les broches de l'interface SPI et la carte Arduino : MOSI, MISO, et CLK doivent être branchées respectivement aux broches n° 11, 12 et 13.

En ce qui concerne les broches d'alimentation électrique : les broches VIN et GND doivent être respectivement connectées aux broches 5 V et GND.

L'ajout du capteur d'humidité est assez simple. Ses fils de raccordement ne sont reliés à aucune broche. Je les ai donc soudés à un connecteur 4 broches. Il vous suffit de connecter respectivement les broches GND, VCC, CLOCK et DATA du capteur aux broches GND, 5 V, n° 7 et n° 6 de la carte Arduino. Reportez-vous à l'image suivante pour un aperçu des câbles du capteur :



Et voilà un aperçu du projet assemblé :



TESTER LE CAPTEUR D'HUMIDITÉ ET DE TEMPÉRATURE DU SOL

Testons le capteur séparément avant de connecter le projet à Internet. Nous nous assurerons ainsi que les câblages sont bons. Pour tester le projet dans des conditions réelles, enterrez le capteur dans un sol humide :

Pour le test du capteur, il est nécessaire d'importer au préalable la bibliothèque Sensirion dans le sketch Arduino :

```
#include <Sensirion.h>
```

Créons ensuite une instance pour ce capteur :

```
Sensirion soilSensor = Sensirion(dataPin, clockPin);
```

À présent, on programme le relevé de température et d'humidité du sol dans la fonction `loop()` du sketch :

```
soilSensor.measure(&temperature, &humidity, &dewpoint);
```

Voici le sketch Arduino complet pour le test du capteur :

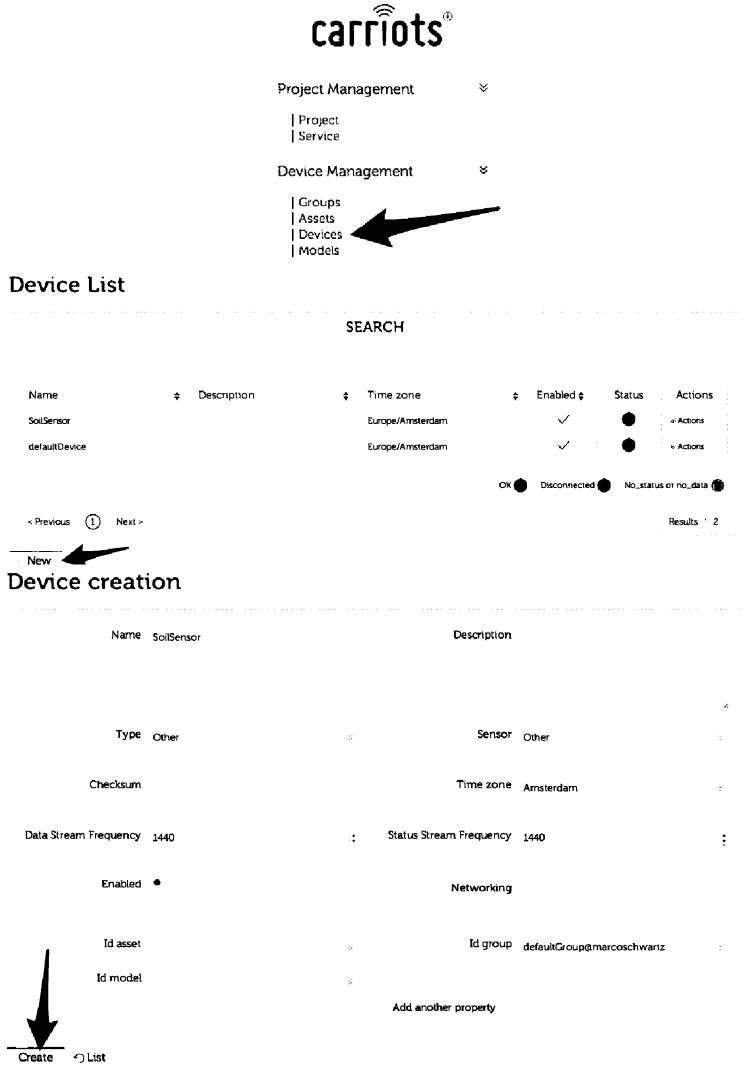
```
// Include Sensirion library
#include <Sensirion.h>

// Sensor pins
constuint8_t dataPin  =  6;
constuint8_t clockPin =  7;

// Variables for the temperature & humidity sensor
float temperature;
float humidity;
float dewpoint;

// Create sensor instance
Sensirion soilSensor = Sensirion(dataPin, clockPin);

void setup()
{
  Serial.begin(115200);
}

void loop()
{
// Make a measurement
  soilSensor.measure(&temperature, &humidity, &dewpoint);

// Print results
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" C, Humidity: ");
  Serial.print(humidity);
  Serial.print(" %");
  Serial.println("");

// Wait 100 ms before next measurement
  delay(100);
}
```

Chargez à présent le code sur la carte Arduino et ouvrez le moniteur série. Vous devriez y voir apparaître les mesures du capteur.

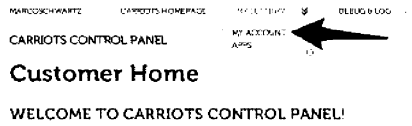
CONFIGURER VOTRE COMPTE CARRIOTS

Pour pouvoir transférer des données vers Carriots, vous devrez d'abord créer un compte.

Après cette étape, attribuez un nom à ce que Carriots appelle un « *device* », chargé de recevoir les données de notre projet.



Il est nécessaire pour la suite de connaître votre clé API. Vous la trouverez sous le menu « My Account » :



TRANSFÉRER DES DONNÉES VERS LE CLOUD

Vous êtes maintenant prêt à envoyer des données vers le service cloud de Carriots. Nous devons écrire le sketch qui sera transféré sur la carte Arduino Uno.

Ce service cloud est légèrement différent des autres. En effet, vous serez amené tout d'abord à définir la clé API dans le sketch mais aussi le « device » auquel vous voulez envoyer des données :

```
#define WEBSITE  "api.carriots.com"
#define API_KEY "yourAPIkey"
#define DEVICE  "yourDeviceName@yourUserName"
```

Dans la fonction `loop()` du sketch, il vous faut formater les données mesurées par la carte Arduino en fonction des instructions fournies par le site Carriots. Les données envoyées par notre sketch devraient ressembler à cela :

```
{
"protocol":"v2",
"at":"now",
"device":"yourDevice@yourUserName",
"data":{
"Temperature":"21.05",
"Humidity":"58.50"
        },
"checksum":""
}
```

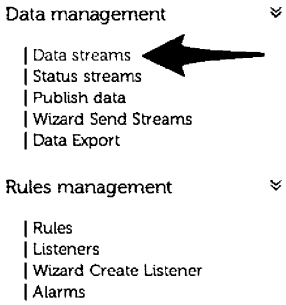
À la fin de chaque boucle, ces données sont envoyées avec un en-tête HTTP. On lit la réponse du serveur avant de l'afficher sur le moniteur série. Si le serveur accepte les données, le moniteur affichera « OK ». Enfin, on programme l'interruption de la connexion ainsi qu'un délai de 10 secondes avant d'effectuer un nouveau relevé de mesures.

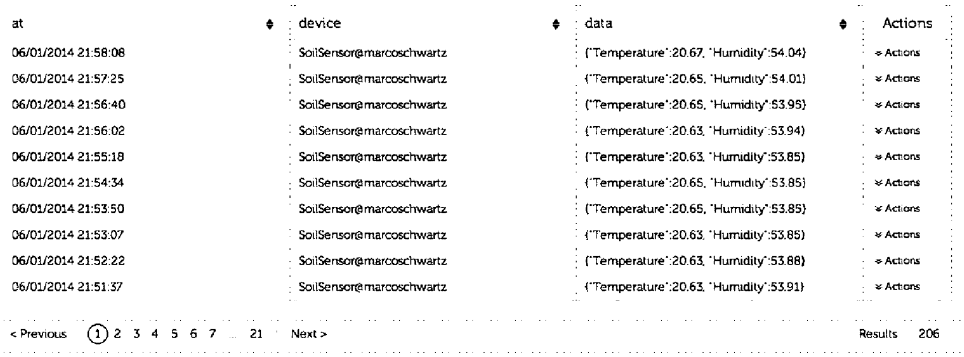
Pour voir si des données ont été reçues sur Carriots, il vous suffit de vous rendre dans la section « Data streams » après vous être identifié.

Vous devriez voir apparaître des données pour votre « device ».

Vous n'êtes pas au bout de vos surprises : Carriots peut également vous aider à afficher vos données sous forme de graphique et à les incorporer à votre site Internet, si vous le souhaitez. Pour tenter l'expérience, suivez le lien « Wizard Widget Graphs » dans le menu de gauche.

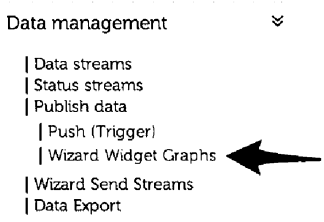
Vous accédez à un menu vous permettant de choisir progressivement un type de graphique et les données concernées. Pour finir, vous aurez accès au code que vous pourrez insérer dans une page web afin d'afficher automatiquement le graphique. J'ai aussi ajouté un exemple d'une telle page dans le dépôt GitHub du projet.





at	device	data	Actions
06/01/2014 21:58:08	SoilSensor@marcoschwartz	{"Temperature":20.67, "Humidity":54.04}	Actions
06/01/2014 21:57:25	SoilSensor@marcoschwartz	{"Temperature":20.65, "Humidity":54.01}	Actions
06/01/2014 21:56:40	SoilSensor@marcoschwartz	{"Temperature":20.65, "Humidity":53.95}	Actions
06/01/2014 21:56:02	SoilSensor@marcoschwartz	{"Temperature":20.63, "Humidity":53.94}	Actions
06/01/2014 21:55:18	SoilSensor@marcoschwartz	{"Temperature":20.63, "Humidity":53.85}	Actions
06/01/2014 21:54:34	SoilSensor@marcoschwartz	{"Temperature":20.65, "Humidity":53.85}	Actions
06/01/2014 21:53:50	SoilSensor@marcoschwartz	{"Temperature":20.65, "Humidity":53.85}	Actions
06/01/2014 21:53:07	SoilSensor@marcoschwartz	{"Temperature":20.63, "Humidity":53.85}	Actions
06/01/2014 21:52:22	SoilSensor@marcoschwartz	{"Temperature":20.63, "Humidity":53.88}	Actions
06/01/2014 21:51:37	SoilSensor@marcoschwartz	{"Temperature":20.63, "Humidity":53.91}	Actions

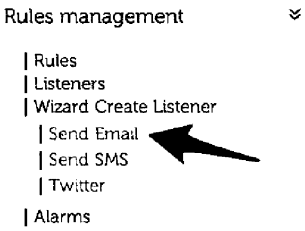
< Previous 1 2 3 4 5 6 7 ... 21 Next > Results 206



DÉCLENCHER UNE ALERTE E-MAIL AUTOMATIQUE

Pour finir, nous allons provoquer une alerte automatique pour l'une des variables. Nous souhaitons par exemple savoir lorsqu'une plante a besoin d'eau, être ainsi alerté si son niveau d'humidité tombe en dessous d'une valeur limite choisie (en fonction de l'espèce de la plante par exemple). Sous Carriots, nous allons devoir créer un « listener » chargé de vous envoyer un e-mail de manière automatique si, dans notre cas, les données franchissent une valeur limite. Un assistant Carriots est prévu à cet effet.

Vous serez invité à saisir votre adresse e-mail et à configurer le « listener ». J'ai testé le mien en programmant (à l'aide d'une condition) l'envoi d'un e-mail dans le cas où la température atteindrait les 23 degrés Celsius.



If expression
```
1 context.data.Temperature>23
```

Then expression
```
1 import com.carriots.sdk.utils.Email;
2
3 def email = new Email();
4 email.to="marcolivier.schwartz@gmail.com";
5 email.subject="Carriots Listener sends email";
6 email.message="Context Data: "+context.data;
7
8 email.send();
```

J'ai ensuite dirigé de la chaleur vers mon capteur. J'ai aussitôt reçu un e-mail lorsque la valeur limite de 23 degrés a été atteinte :

no-reply (13) **Carriots Listener sends email** - Context Data: [Temperature:24.09, Humidity:55.41]

Il est possible d'ajouter d'autres « listeners » qui s'appliqueront à des variables différentes et de programmer l'envoi d'un SMS vers votre mobile plutôt qu'un e-mail.


Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/intro-book

POUR ALLER PLUS LOIN

N'oubliez pas que vous pouvez adapter ce projet à des applications autres que le jardinage. Vous pouvez connecter plusieurs capteurs au service Carriots à l'intérieur et l'extérieur de votre habitation et créer des conditions plus complexes pour programmer l'envoi d'alertes spécifiques basées sur les données enregistrées.

Partie **IV**

Concevoir des circuits imprimés pour des installations personnalisées

17 Construire son propre système Arduino

Peut-être avez-vous déjà eu le projet de réaliser une installation telle qu'un détecteur de mouvement autonome construit autour d'Arduino. Or dans ce cas, il serait peu judicieux d'utiliser une carte Arduino officielle comme le modèle Uno.

Pourquoi ? Tout simplement parce que plusieurs éléments comme le port USB, les LED, le bouton de réinitialisation ou encore les connecteurs vous seraient inutiles. Or ces composants prennent de la place et ils consommeraient de l'énergie pour rien.

Ce chapitre est consacré à la réalisation intégrale d'une carte Arduino sur une plaque d'essai. Nous utiliserons le moins de composants possible. Grâce à ce chapitre, vous serez en mesure de personnaliser ce schéma de base pour l'adapter à vos besoins, et vous vous servirez de ces acquis pour construire vos propres cartes basées sur Arduino.

MATÉRIEL ET LOGICIEL REQUIS

Plusieurs composants sont nécessaires à la réalisation de votre système Arduino sur une plaque d'essai.

L'élément clé du projet est évidemment le **microcontrôleur**, chargé d'exécuter vos sketches Arduino. Un grand nombre de microcontrôleurs peuvent faire l'affaire mais nous utiliserons, par souci de simplicité, la même puce Atmel ATmega328 que celle de la carte Arduino Uno.

Équipez-vous d'une **puce programmée** avec le « bootloader » Arduino. Un *bootloader* est un code « gravé » dans la mémoire morte (ROM) d'un microcontrôleur. De telles puces déjà programmées avec le bootloader Arduino sont aujourd'hui faciles à se procurer et je vous conseille d'en acheter une. Vous serez ainsi en mesure d'utiliser le projet que nous allons créer dans ce chapitre avec le logiciel Arduino IDE sans effectuer de manipulations supplémentaires.

Il vous faudra également vous munir de plusieurs composants autour de la puce : deux **condensateurs** de 10 µF, deux **condensateurs** de 22 pF, une **résistance** de 10 kΩ, deux **résistances** de 220 Ω (ou 330 Ω), une **LED** rouge, une LED verte et un **oscillateur** à quartz 16 MHz.

Vous aurez également besoin d'un **convertisseur FTDI** pour programmer la puce Arduino directement sur la plaque d'essai. Si vous utilisez une autre carte Arduino pour programmer le microcontrôleur, comme le modèle Uno, passez alors à la partie suivante.

Enfin, munissez-vous d'une **plaque d'essai** ainsi que de **fils de raccordement** mâle/mâle.

LISTE DES COMPOSANTS

- **Atmel ATmega328** (http://boutique.semageek.com/fr/48-atmega328-avec-bootloader-arduino-duemilanove.html)
- **Deux condensateurs de 10 µF** (http://www.gotronic.fr/art-condensateur-radial-10uf-25v-110.htm)
- **Deux condensateurs de 22 pF** (http://www.gotronic.fr/art-condensateur-ceramique-22-pf-8683.htm)
- **Résistance de 10 kΩ** (http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html)
- **Deux résistances de 220 Ω ou 330 Ω** (http://www.gotronic.fr/art-5-resistances-1-2w-330-8486-2653.htm)
- **LED rouge** (http://www.gotronic.fr/cat-leds-5mm-standards-477.htm)
- **LED verte** (http://www.gotronic.fr/cat-leds-5mm-standards-477.htm)
- **Oscillateur à quartz 16 MHz** (http://www.gotronic.fr/art-quartz-bas-profil-16-0000-mhz-17101.htm)
- **Module FTDI** (http://www.gotronic.fr/art-module-ftdi-basic-5v-20177.htm)
- **Plaque d'essai** (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- **Fils de raccordement (nappe)** (http://snootlab.com/adafruit/845-nappe-20-cordons-7-cm-3-m-m-fr.html)

Côté **logiciel**, équipez-vous bien entendu de **Arduino IDE**.

Procurez-vous également les drivers FTDI pour le module du même nom. Il est cependant possible qu'ils soient d'ores et déjà installés sur votre ordinateur.

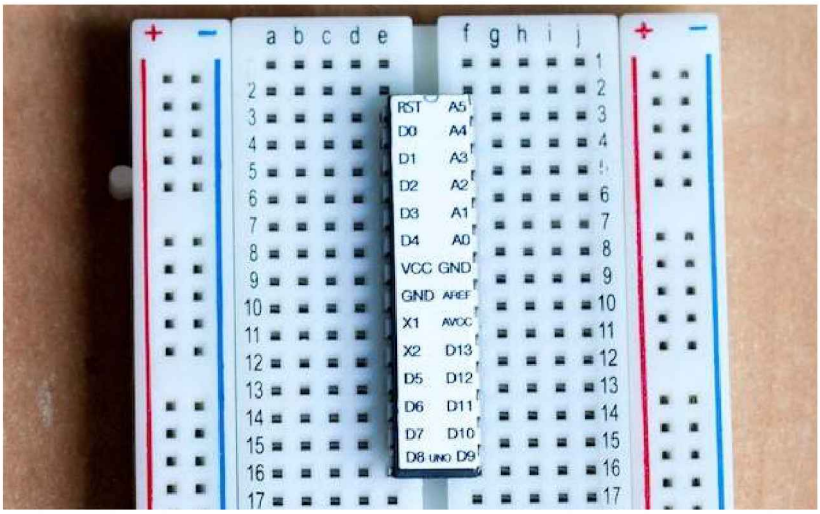
> Sinon, ils sont téléchargeables à cette adresse :
> http://www.ftdichip.com/FTDrivers.htm

J'ai personnellement opté pour les drivers « Virtual COM Port (VCP) ».

CONFIGURER LE MATÉRIEL

Ce montage est assez complexe. Soyez donc bien attentif aux instructions données. Afin de laisser de la place au module FTDI, raccordez le microcontrôleur sur la partie supérieure de la plaque d'essai.

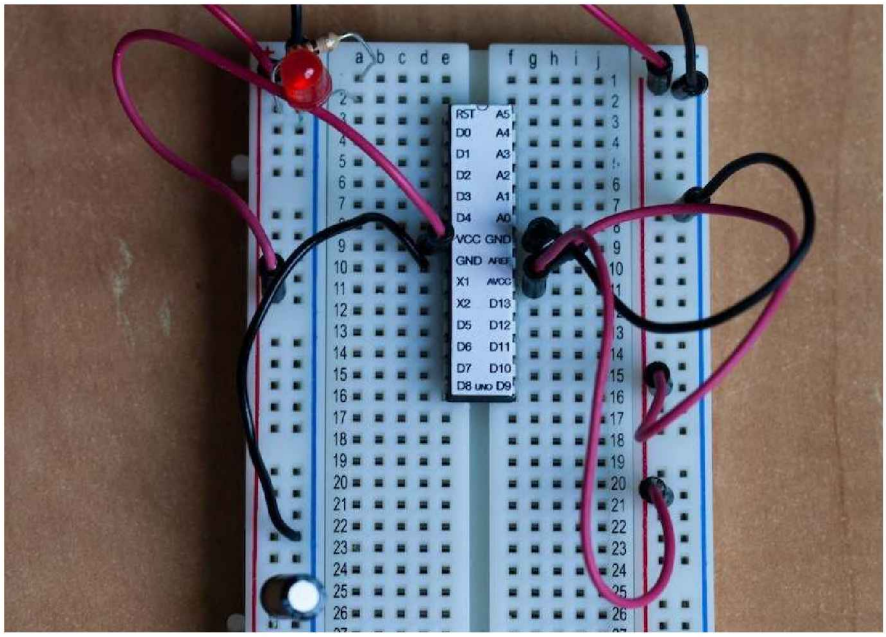
Vous devriez aboutir à un résultat similaire à celui-ci :



Occupons-nous maintenant de l'alimentation. Il vous faut connecter les lignes d'alimentation ensemble sur la plaque d'essai : la masse et l'alimentation électrique positive. Connectez ensuite les broches VCC, AREF et AVCC à la ligne d'alimentation électrique positive et les deux broches GND aux lignes de masse.

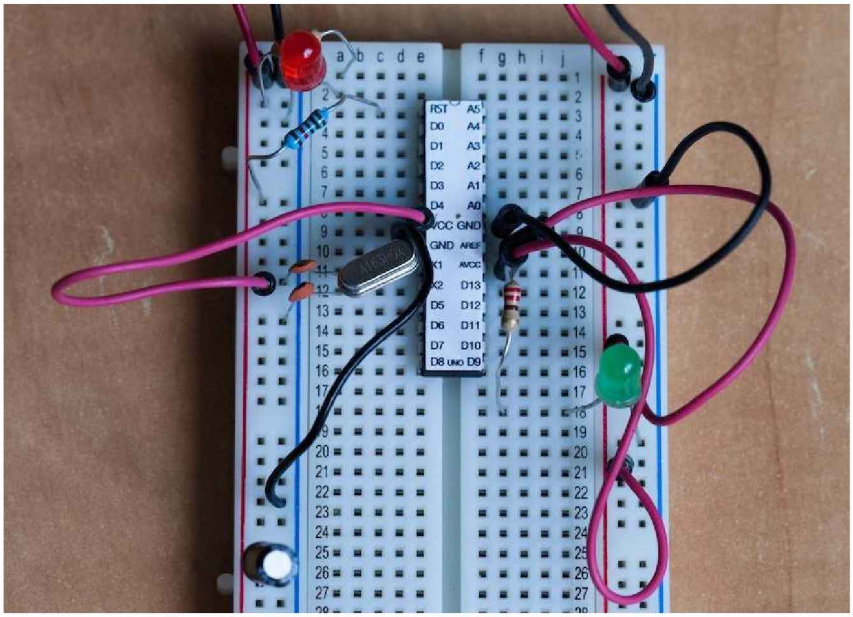
J'ai également ajouté une résistance de 220 Ω en série avec une LED rouge entre les deux lignes d'alimentation afin de vérifier si le courant passe.

Pour terminer, ajoutez un condensateur de 10 µF entre les deux lignes d'alimentation pour stabiliser la tension appliquée au microcontrôleur. Voici un aperçu du résultat :



Penchons-nous à présent sur le microcontrôleur. Vous devez connecter une résistance de 10 kΩ entre la broche de remise à zéro (*reset*) et la ligne d'alimentation électrique positive pour éviter que la puce ne se réinitialise. Insérez à présent l'oscillateur à quartz entre les broches X1 et X2. Enfin, connectez chacune de ces broches à la ligne de masse *via* les deux condensateurs de 22 pF.

Il est tout à fait possible de connecter en série une LED verte avec une résistance de 220 Ω sur la broche n° 13 du microcontrôleur puisque cette dernière est connectée à une LED sur la carte Arduino Uno. Cela vous permettra de voir rapidement si le projet fonctionne :

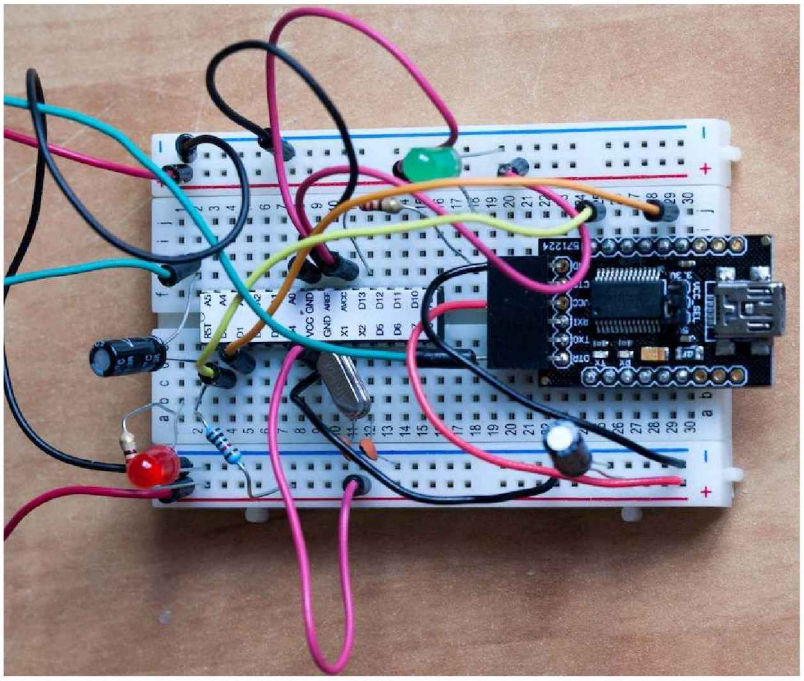


Si vous utilisez une autre carte (une Arduino Uno par exemple) pour programmer votre puce, vous pouvez vous arrêter ici. Votre projet fonctionnera en connectant seulement une source d'alimentation (quelques piles peuvent suffire) aux lignes d'alimentation. Pour la tension soumise au microcontrôleur, choisissez une valeur comprise entre 1,8 et 5,5 V.

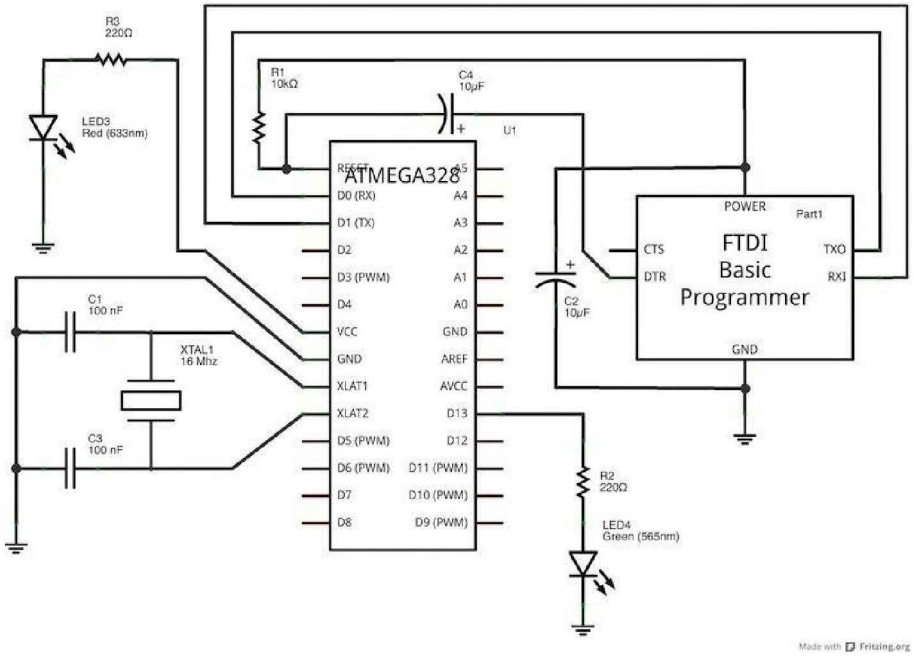
En revanche, pour programmer la puce sans la retirer de la plaque d'essai, vous devrez accomplir les manœuvres suivantes. Installez tout d'abord le module FTDI sur la partie inférieure de la plaque d'essai. Connectez ensuite les broches VCC et GND du module FTDI (le projet sera alimenté en USB) aux lignes d'alimentation correspondantes.

Connectez ensuite respectivement les broches TX et RX du module aux broches RX et TX du microcontrôleur (broches 1 et 2). Connectez enfin la broche DTR du module à la broche de remise à zéro (reset) du microcontrôleur *via* un condensateur de 10 µF.

Vous devriez aboutir à ce résultat :



Voici le schéma complet du projet qui vous facilitera sans doute la tâche :



TESTER LE PROJET

Testons maintenant le montage réalisé. Si vous voulez programmer votre microcontrôleur sur la plaque d'essai à l'aide du module FTDI, il vous suffit de relier la carte à votre ordinateur *via* un câble USB. Si les lignes d'alimentation ont été correctement raccordées, vous devriez voir la LED s'allumer. Vous en déduirez alors que le courant circule bien dans votre projet.

Ouvrez à présent le logiciel Arduino IDE et sélectionnez le port série concerné (commençant par « usbserial » dans mon cas). J'ai sélectionné ensuite la carte Arduino Uno. Selon la puce que vous utilisez, vous serez peut-être amené à sélectionner « [Optiboot] Arduino Duemilanove » dans l'onglet « Board ». Afin de vérifier brièvement si le projet est opérationnel, choisissez le sketch « Blink » inclus par défaut dans Arduino IDE.

Chargez le sketch et vous devriez maintenant voir la LED verte clignoter. Félicitations, vous avez réussi à monter votre propre système Arduino du début jusqu'à la fin.

POUR ALLER PLUS LOIN

Vous êtes désormais en mesure de réaliser intégralement vos propres systèmes Arduino. Nous avons retenu le nombre minimum de composants pour construire ce système Arduino sur une plaque d'essai. Nous l'avons ensuite testé à l'aide d'un simple sketch Arduino.

Il vous est d'ores et déjà possible d'ajouter des fonctionnalités au projet ou des capteurs comme un détecteur de mouvement PIR. Nous verrons par la suite comment créer nos propres circuits imprimés (PCB) basés sur Arduino. Il est donc essentiel pour la suite de savoir comment un système Arduino fonctionne.

18 Optimiser Arduino pour des projets à basse consommation

Pour la plupart des projets Arduino disponibles sur Internet, l'alimentation n'est pas un problème étant donné que la carte Arduino est presque toujours alimentée par un ordinateur via un câble USB.

Vous pouvez toutefois créer un système autonome avec une pile comme source d'alimentation. Admettons que vous vouliez alimenter un détecteur de mouvement sans fil à l'aide d'un jeu de piles.

La première idée qui nous vient à l'esprit consisterait à relier une carte Arduino (Uno R3 par exemple) directement à une pile. Simple, non ? Cela pourrait fonctionner mais certains composants tels que les régulateurs de tension consomment de l'énergie en permanence. La pile se déchargera donc très rapidement. Il nous faut donc trouver une solution mieux adaptée.

Nous allons réaliser un montage similaire à celui du chapitre précédent. Nous nous intéresserons aux techniques permettant à notre projet de consommer moins d'énergie afin d'augmenter son autonomie et le rendre autonome plusieurs années.

MATÉRIEL ET LOGICIEL REQUIS

Plusieurs composants sont nécessaires. L'élément clé du projet est évidemment le **microcontrôleur**, chargé d'exécuter vos sketches Arduino, comme la puce Atmel ATmega328.

Équipez-vous d'une **puce programmée** avec ce que l'on appelle le « bootloader » Arduino. Je vous conseille d'acheter directement une de ces puces déjà programmées.

Nous avons précédemment utilisé un **module FTDI** pour programmer la puce Arduino directement sur la plaque d'essai. Mais pour ce projet, aucune source d'alimentation externe ne sera branchée à la plaque d'essai. J'utiliserai donc uniquement une carte **Arduino Uno** pour programmer le microcontrôleur.

Une **pile** sera nécessaire à l'alimentation de la carte Arduino. Vous lierez les deux éléments sans intermédiaire. Vous n'aurez pas besoin d'utiliser de régulateurs de tension. J'ai choisi un jeu de deux piles AA (1.5 V) avec porte-piles pour alimenter le microcontrôleur avec une tension proche de 3 V comme indiqué dans la documentation de la puce ATmega328.

Il vous faudra également vous munir des composants listés dans l'encadré suivant. Enfin, équipez-vous d'une **plaque d'essai** et de **fils de raccordement** mâle/mâle.

LISTE DES COMPOSANTS

- **Atmel ATmega328** (http://boutique.semageek.com/fr/48-atmega328-avec-bootloader-arduino-duemilanove.html)
- **Condensateur de 10 µF** (http://www.gotronic.fr/art-condensateur-radial-10uf-25v-110.htm)
- **Deux condensateurs de 22 pF** (http://www.gotronic.fr/art-condensateur-ceramique-22-pf-8683.htm)
- **Résistance de 10 kΩ** (http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html)
- **Une résistance de 220 Ω ou 330 Ω** (http://www.gotronic.fr/art-5-resistances-1-2w-330-8486-2653.htm)
- **LED** (http://www.gotronic.fr/cat-leds-5mm-standards-477.htm)
- **Oscillateur à quartz 16 MHz** (http://www.gotronic.fr/art-quartz-bas-profil-16-0000-mhz-17101.htm)
- **Plaque d'essai** (http://snootlab.com/breadboard/349-breadboard-400-points-blanc-demie-fr.html)
- **Fils de raccordement (nappe)** (http://snootlab.com/adafruit/845-nappe-20-cordons-7-cm-3-m-m-fr.html)

Côté **logiciel**, j'ai testé plusieurs bibliothèques destinées à réduire au minimum la consommation d'énergie. Mon choix s'est porté finalement sur la bibliothèque **JeeLib**.

Téléchargez cette bibliothèque à l'adresse suivante : https://github.com/jcw/jeelib

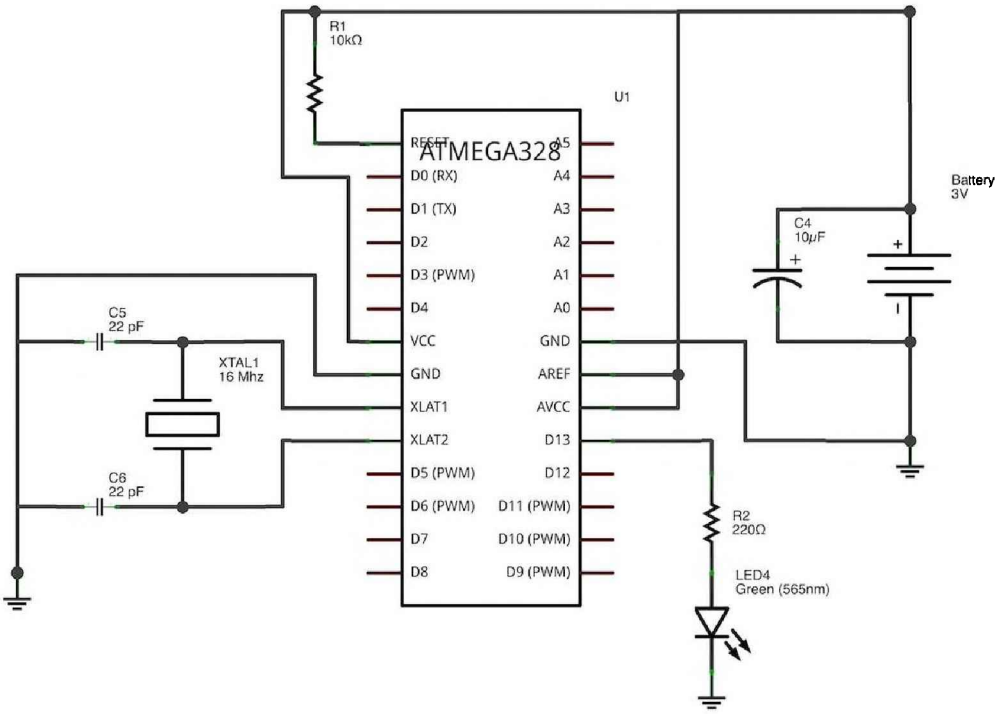
Pour l'installer, placez son dossier dans **Arduino/libraries**.

CONFIGURER LE MATÉRIEL

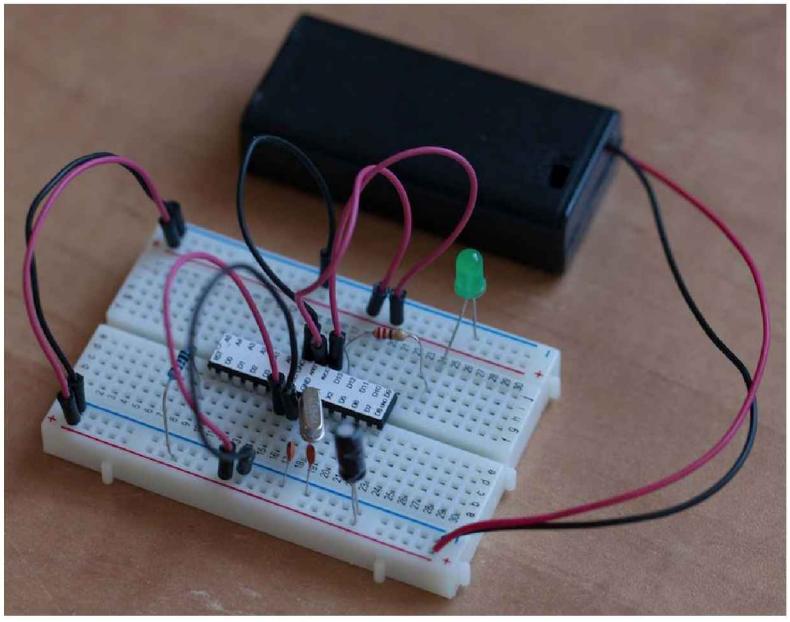
À l'image du projet précédent, la réalisation de ce montage est assez complexe. Lisez donc attentivement les instructions données. Pour vous aider, voici ci-après le schéma complet du projet.

Placez d'abord le microcontrôleur au centre de la plaque d'essai. En ce qui concerne l'alimentation : connectez les lignes d'alimentation de chaque côté, puis la ligne d'alimentation négative aux deux broches du microcontrôleur et enfin la ligne d'alimentation positive aux broches VCC, AVCC et AREF. Placez maintenant le condensateur de 10 µF entre les deux lignes d'alimentation. Enfin, complétez le système avec la pile.

Poursuivez le montage avec l'insertion de l'oscillateur à quartz entre les broches X1 et X2 et placez les deux condensateurs de 22 pF entre les broches et la masse. Il faut ensuite connecter la broche RST à la ligne d'alimentation positive à l'aide d'une résistance de 10 Ω. Pour vérifier l'état de fonctionnement du système, connectez en série la LED verte d'une part avec une résistance de 220 Ω à l'entrée numérique n° 13 de la carte Arduino et de l'autre part à la masse.



Voici à quoi le montage devrait ressembler :



TESTER LE PROJET

Préparons maintenant le sketch qui permettra de tester les fonctions basse consommation de la bibliothèque **JeeLib**. Le sketch dont je me suis servi est le suivant :

```
#include <JeeLib.h>

// LED pin
int led_pin = 13;

// Setup the watchdog
ISR(WDT_vect) { Sleepy::watchdogEvent(); }

void setup() {

// Set LED pin to output
  pinMode(led_pin, OUTPUT);
}

void loop() {

// Turn the LED on and sleep for 5 seconds
  digitalWrite(led_pin, HIGH);
  Sleepy::loseSomeTime(5000);

// Turn the LED off and sleep for 5 seconds
  digitalWrite(led_pin, LOW);
  Sleepy::loseSomeTime(5000);
}
```

Voyons maintenant ce sketch en détail. Il vous suffit en fait d'introduire la bibliothèque **JeeLib** en saisissant :

```
#include <JeeLib.h>
```

Initialisons le chien de garde avec :

```
ISR(WDT_vect) { Sleepy::watchdogEvent(); }
```

Vous pouvez mettre en veille la carte Arduino pendant un laps de temps donné :

```
Sleepy::loseSomeTime(5000);
```


Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/pcb-book

Chargez le sketch à l'aide du logiciel Arduino et replacez la puce sur la plaque d'essai. Votre carte Arduino doit réagir de la même façon qu'au chapitre précédent (avec des intervalles de 5 secondes). Il s'agit cependant d'un scénario légèrement différent puisqu'ici lorsque la LED est inactive, la puce Arduino consomme peu d'énergie, ce qui n'est pas le cas lorsque l'on a recours à la fonction `delay()`.

À présent, évaluons la consommation de courant réelle du projet. Pour ce faire, installez un multimètre entre l'une des lignes d'alimentation. J'ai connecté la broche positive de la pile à une broche du multimètre et l'autre broche de ce dernier à la ligne d'alimentation rouge de la plaque d'essai.

Afin d'obtenir des valeurs de référence, j'ai également testé le projet avec le sketch « Blink » qui fait figure d'exemple sur Arduino IDE. Voici les résultats obtenus :

- LED inactive, sans bibliothèque JeeLib : 6,7 mA
- LED active, sans bibliothèque JeeLib : 8,8 mA
- LED inactive, avec bibliothèque JeeLib : 43 uA
- LED active, avec bibliothèque JeeLib : 2,2 mA

On remarque grâce à ces résultats que notre système Arduino installé sur la plaque d'essai consomme 6.7 mA au repos avec l'exemple Arduino Blink. *Via* la fonction veille, la consommation de courant ne dépasse pas les 43 uA, soit 150 fois moins.

Calculons à présent l'impact sur un projet réel, comme un capteur de température. Partons du principe qu'il faut 500 ms pour prendre une mesure avec une consommation de courant de 2,5 mA. Le système est ensuite mis en veille pendant 10 secondes avant que la boucle soit relancée. La consommation « moyenne » de courant est donc de 0,16 mA sur une boucle complète. Des piles d'une puissance de 2 500 mAh pourront ainsi alimenter le système pendant environ 2 ans.

Évidemment, cette estimation est approximative car d'autres facteurs peuvent intervenir et fausser ces chiffres. Vous pouvez adapter ce concept à n'importe quel système dont la période d'activité est minime en comparaison avec la période de veille. Ainsi, votre système Arduino aura une autonomie de plusieurs années.

POUR ALLER PLUS LOIN

Nous avons étudié dans ce chapitre la manière d'optimiser un système Arduino afin qu'il consomme un minimum d'énergie. Ce principe est essentiel pour créer un système alimenté par une pile ou un panneau solaire.

Vous pouvez partir de ce projet pour mettre au point votre propre système Arduino basse consommation. Associez au projet de ce chapitre un détecteur de mouvement basique et un petit haut-parleur pour mettre au point un système d'alarme alimenté par un jeu de piles qui fonctionnera pendant des mois, voire des années.

19 Concevoir une carte d'extension Arduino

Maintenant que nous en savons plus sur la manière dont un système Arduino fonctionne, il est possible de concevoir notre propre système. Commençons par un système simple : une carte d'extension Arduino. Les cartes d'extension offrent des fonctions supplémentaires aux cartes Arduino et les rendent ainsi plus polyvalentes. Elles ajoutent des composants à la carte Arduino.

La mise au point de votre propre carte d'extension comporte des avantages par rapport à la réalisation de projets sur plaque d'essai. L'avantage principal est de pouvoir connecter votre carte d'extension sur votre carte Arduino, c'est-à-dire sans vous encombrer de câbles.

Ce chapitre vous aidera à construire votre propre carte d'extension Arduino adaptée à la domotique. Nous verrons toutes les étapes pas à pas, du début jusqu'à la fin.

Je survolerai de manière générale l'ensemble des étapes de la conception et apporterai des détails sur ces certains points. L'objet de ce chapitre est de vous donner les clés pour concevoir une carte d'extension issue de votre imagination et non de vous apprendre à réaliser le modèle qui est pris ici pour exemple.

MATÉRIEL ET LOGICIEL REQUIS

Suivant les éléments qui composeront votre carte d'extension, vous devrez acquérir les différents composants nécessaires au circuit imprimé une fois que vous l'aurez reçu.

LISTE DES COMPOSANTS

- Relais 5 V (http://www.selectronic.fr/relais-de-signal-1rt-5v-5a-120vac-ou-1a-30vdc.html)
- Capteur de courant ACS712 (http://www.digikey.fr/product-detail/fr/ACS712ELCTR-30A-T/620-1191-1-ND/1284608)
- Capteur DHT11 avec résistance de 4,7 kΩ (http://snootlab.com/adafruit/255-capteur-de-temperateur-et-d-humidite-dht11-extras-.html)
- Photorésistance (http://www.gotronic.fr/art-photoresistance-ldr04-2150.htm)
- Deux résistances de 10 kΩ (http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html)
- Diode redresseur (http://www.amazon.fr/Diode-redresseur-1A-50V-IN4001/dp/B004QOPOY8)

☞

☞

- Condensateur de 0,1 nF (http://www.digikey.fr/product-detail/fr/K101K10X7RF5UH5/BC2657CT-ND/2356871)
- Condensateur polarisé de 1 uF (http://www.digikey.fr/product-detail/fr/EEA-GA1H1R0H/P15831CT-ND/3831198)
- Barrette de connexion (trois broches nécessaires au projet) (http://www.audiophonics.fr/barette-secable-broches-1x36-ecartement-254mm-p-7047.html)
- Borne à vis trois pôles (http://www.lextronic.fr/P4719-bornier-a-vis-3-plots-pour-ci-pas-5-mm.html)
- Kit de connecteurs pour Arduino (http://snootlab.com/lang-fr/connectors/192-kit-connecteurs-empilables-arduino-1.html)

Côté logiciel, procurez-vous la dernière version du programme de conception assistée par ordinateur de circuits imprimés **Eagle**.

Vous pourrez l'obtenir ici : http://www.cadsoftusa.com/download-eagle/

Vous aurez également besoin des bibliothèques **Eagle Sparkfun**.

Téléchargez les à cette adresse : https://github.com/sparkfun/SparkFun-Eagle-Libraries

Pour installer la bibliothèque **Eagle**, déposez-la dans **/lbr**, situé sur votre disque dur dans le dossier racine **Eagle**.

CONCEVOIR LA CARTE D'EXTENSION

Lorsque vous voulez créer une nouvelle carte d'extension, la première étape consiste à définir ses caractéristiques. Dans ce cas, je voulais une carte d'extension comportant un certain nombre de composants régulièrement utilisés en domotique. J'ai donc décidé d'y inclure :

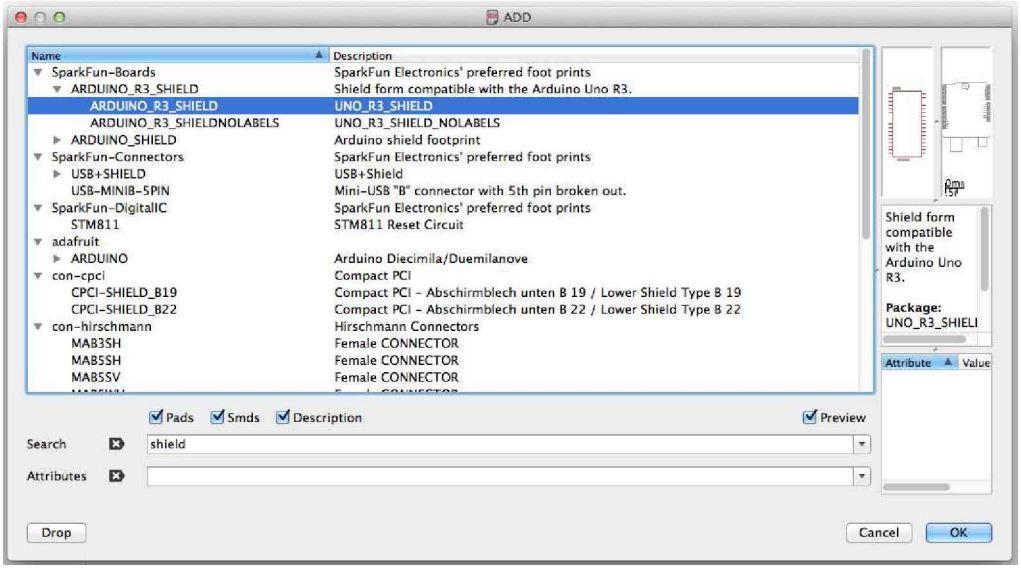
- un relais accompagné d'un capteur de courant pour contrôler une lampe et mesurer simultanément sa consommation énergétique ;
- un capteur de température et d'humidité ;
- un capteur d'éclairage ;
- la possibilité de connecter un détecteur de mouvement et de contact si besoin.

Faisons en sorte également que la carte d'extension soit compatible avec d'autres cartes comme celles vendues dans le commerce par exemple.

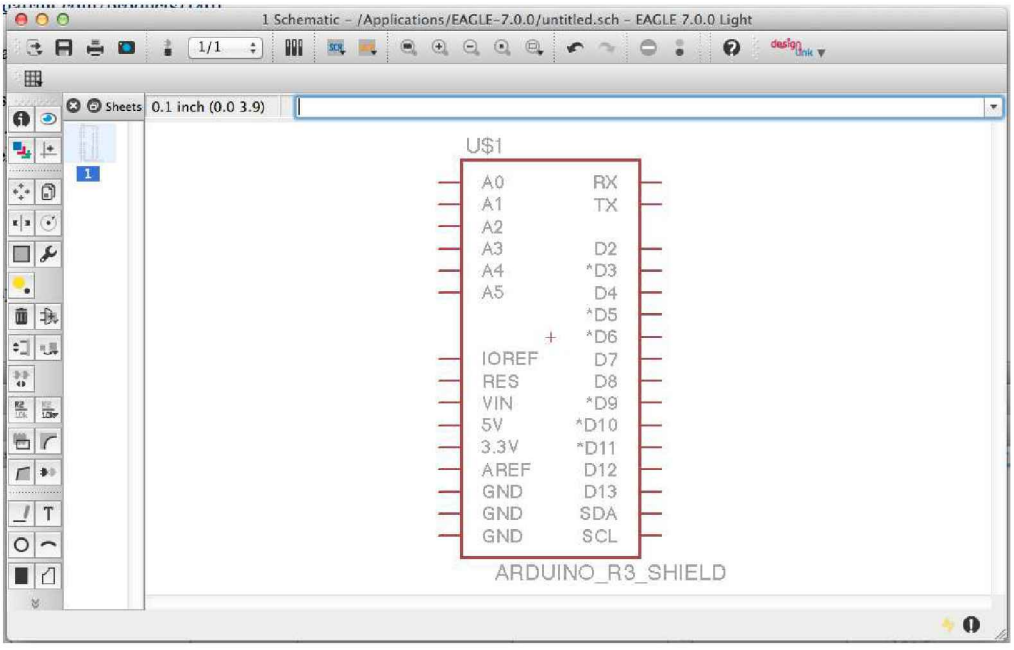
De mon côté, je souhaitais rendre ma carte compatible avec la carte d'extension CC3000 WiFi de la marque Adafruit :
http://snootlab.com/adafruit/460-adafruit-cc3000-wifi-shield-avec-connecteur-ufl-pour-antenne-externe-fr.html

Pour cela, j'ai veillé à ne pas choisir les mêmes broches que celle de la carte d'extension WiFi que je voulais utiliser.

Commençons par la conception du schéma de la carte d'extension. Placez tout d'abord le composant principal qui définira les différentes broches de la carte. Par chance, la bibliothèque Sparkfun dispose de ce composant.

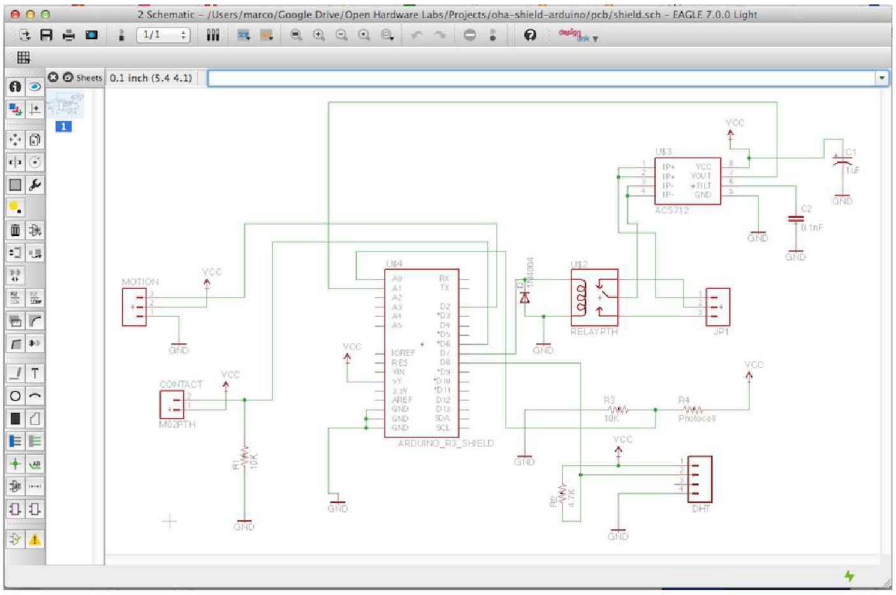


Vous pouvez désormais introduire la carte d'extension dans l'éditeur de schémas.



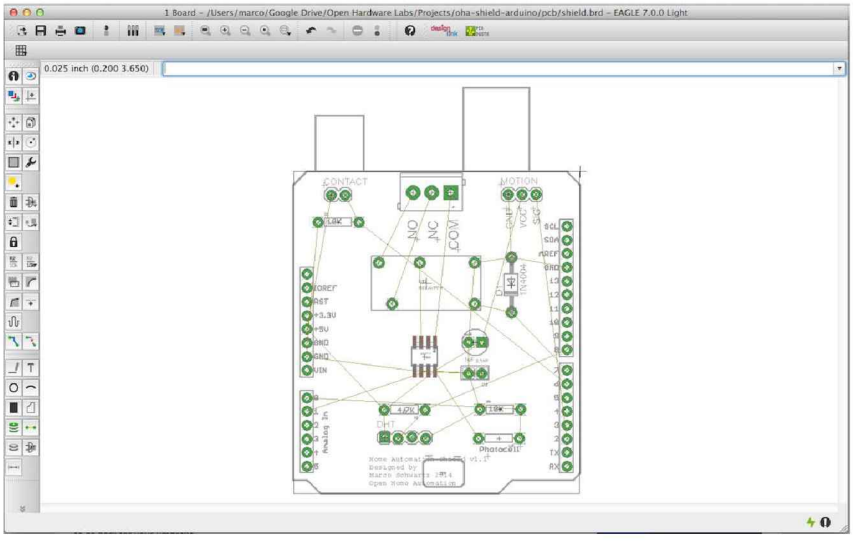
Faites de même avec les différents composants que vous souhaitez intégrer à la carte. Il vous faut maintenant les connecter ensemble. Utilisez le menu « Add » pour chaque composant que vous souhaitez installer ; leur nombre varie en fonction de votre projet.

Voici un aperçu du schéma complet :

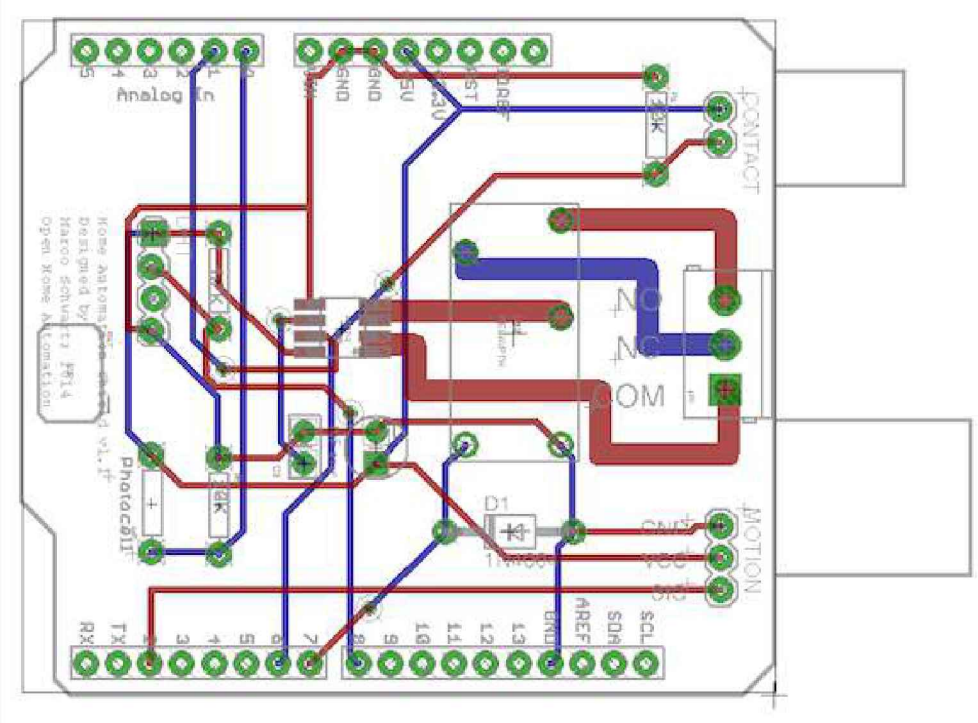


Intéressons-nous à présent au schéma de circuits imprimés (*layout*) de la carte d'extension. N'oubliez pas que celle-ci doit pouvoir fonctionner avec les cartes Arduino existantes. Vous devez donc faire face à une contrainte d'espace.

Voici une illustration du schéma de circuits imprimés comprenant l'ensemble des composants placés sur la carte :



Utilisez le routeur automatique du logiciel. Voici le résultat obtenu, une fois le routage achevé :



La phase de conception s'achève par une vérification de la conformité avec certaines règles. Celles-ci nous permettent d'assurer la fiabilité de notre circuit imprimé lors de la production et d'éviter des erreurs qui l'empêcheraient de fonctionner correctement. C'est ce que l'on appelle la vérification des règles de dessin (*Design Rule Checking*) qui peut être effectuée directement par Eagle.

> Je recommande d'utiliser les règles fournies par le détaillant de composants électroniques Sparkfun :
> http://www.sparkfun.com/tutorial/Eagle-DFM/SparkFun.dru

Téléchargez le fichier avant de le glisser dans le dossier **/dru** de Eagle. Sélectionnez Tools > DRC. C'est à cet endroit que vous chargerez le fichier. Cliquez ensuite sur « Check » pour lancer la vérification. Si des erreurs sont détectées, un descriptif de chacune d'elles apparaîtra. Il vous faudra répéter cette action jusqu'à la validation finale.

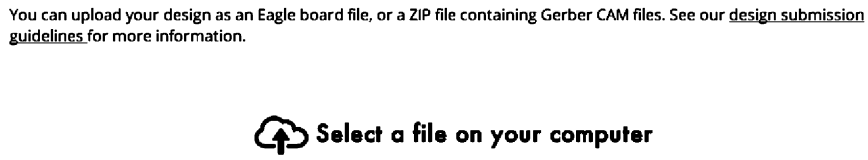
> Vous pouvez retrouver le code complet du chapitre dans le dossier correspondant du dépôt GitHub du livre : https://github.com/openhomeautomation/pcb-book

FABRIQUER LA CARTE

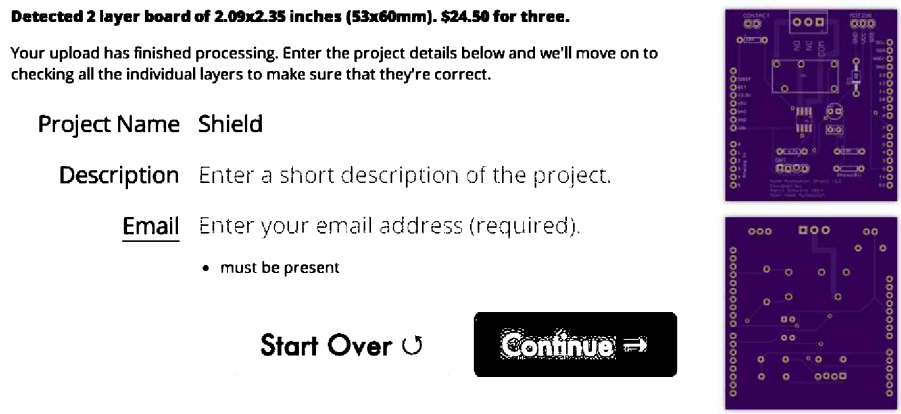
Il est temps de prévoir la fabrication de notre circuit imprimé. Pour cela, vous aurez le choix entre plusieurs services. Cependant, puisqu'il s'agit là de votre premier projet de circuit imprimé, je vous recommande de le confier au service OSH Park. Le processus de fabrication qui y est proposé est très simple.

| Rendez-vous d'abord à cette adresse : https://oshpark.com/

Il vous sera demandé de charger votre fichier .brd. OSH Park permet l'importation directe de fichiers Eagle. Il n'est donc pas nécessaire de les exporter dans un autre format :



Après quelques instants, OSH Park identifiera les différentes couches de votre circuit imprimé et générera un aperçu sur le site :

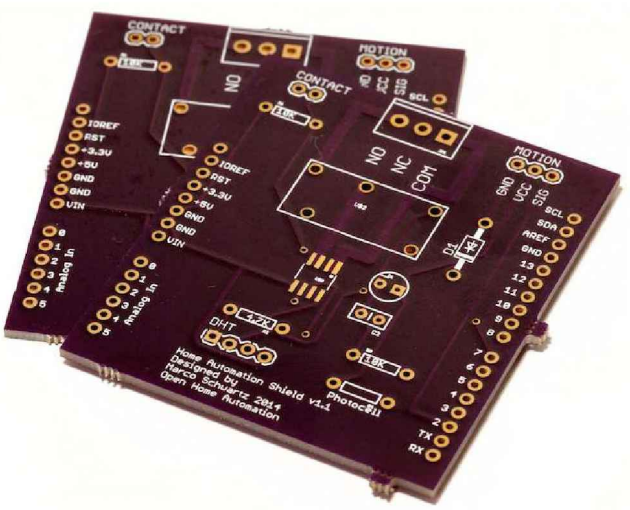


À ce stade, vous devrez entrer le nom de votre projet ainsi que vos informations personnelles. Puis, vous serez invité à régler votre commande afin de lancer la fabrication de votre circuit.

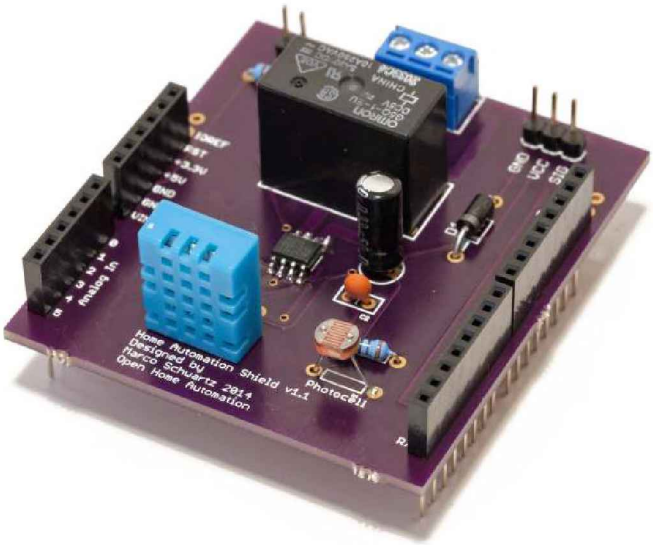
N'oubliez pas de commander en parallèle l'ensemble des composants choisis pour l'assemblage de votre carte d'extension. Il n'y a rien de plus frustrant que de devoir retarder l'assemblage d'un projet en raison de composants manquants.

RÉSULTAT FINAL

Dans mon cas, deux semaines après, le circuit imprimé était dans ma boîte aux lettres :



Je l'ai assemblé avec ses composants, et voici un aperçu du résultat :



Il était alors essentiel de tester la carte. J'ai donc utilisé un sketch Arduino basique afin de tester de manière automatique les différentes fonctionnalités de la carte d'extension. Cette méthode est pratique pour l'identification de bugs potentiels.

Ne négligez pas cette étape car il est fort probable qu'un voire deux bugs soient détectés sur votre carte. En fonction des résultats de ce test initial, il vous faudra peut-être de nouveau modifier le schéma.

Pour aller plus loin

Résumons ce que nous avons appris dans ce chapitre. Vous avez conçu votre propre carte d'extension Arduino pour une utilisation en domotique. Nous avons parcouru l'intégralité du processus, de la conception de la carte jusqu'à sa fabrication, avant son test final.

Vous pouvez partir de ce projet pour mettre au point vos propres cartes d'extension Arduino destinées à la domotique. Vous pouvez prendre le schéma de ce chapitre comme référence. D'autres capteurs peuvent être ajoutés à votre carte ainsi que des composants tels que des afficheurs LCD ou du matériel pour brancher un moteur à courant continu.

20 Concevoir une carte Arduino personnalisée avec Eagle

Dans ce chapitre, vous allez mettre en application l'ensemble des connaissances acquises jusqu'ici afin de réaliser votre propre carte Arduino. Nous nous intéresserons à la manière de définir les caractéristiques de la carte, de la concevoir et enfin de l'assembler. En fin de chapitre, nous procéderons au test de la carte produite par le fabricant.

À titre d'exemple, nous nous appuierons sur une version simplifiée de la carte Arduino Pro. Nous nous inspirerons de sa conception pour fabriquer ensuite notre propre carte.

Comme pour le chapitre précédent, je survolerai de manière générale l'ensemble des étapes de la conception et apporterai des détails sur certains points. L'objectif est de vous donner les clés pour concevoir une carte Arduino issue de votre imagination et non de vous apprendre à réaliser le modèle présenté ici comme exemple.

MATÉRIEL ET LOGICIEL REQUIS

Dans ce projet, nous n'aurons pas besoin d'acheter de composants car nous aurons recours à un service en ligne chargé de gérer la partie matérielle. Nous allons nous servir du service d'assemblage de circuits imprimés proposé par Seeedstudio qui produira nos cartes avant d'assembler les composants.

Cela implique toutefois de sélectionner tous les composants dans une liste définie.

| Voici un lien vers cette liste : http://www.seeedstudio.com/wiki/Open_parts_library

Côté logiciel, procurez-vous la dernière version du programme de conception assistée par ordinateur de circuits imprimés Eagle.

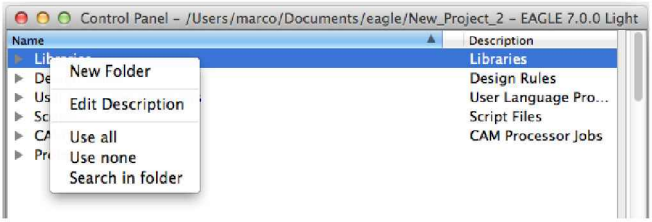
| Vous pouvez l'obtenir à cette adresse : http://www.cadsoftusa.com/download-eagle/

Il nous faudra également installer la bibliothèque Eagle « Open Parts Library » afin de nous assurer d'utiliser seulement des composants autorisés pour cette carte.

| http://www.seeedstudio.com/wiki/File:OPL_eagle_library.zip

Pour installer la bibliothèque, déposez-la dans **/lbr**, situé dans le dossier racine Eagle.

Afin de l'activer et de l'utiliser dans le cadre d'un nouveau projet, cliquez sur « Use all » au lancement du logiciel Eagle :



Téléchargez aussi le fichier CAM de Sparkfun pour Eagle. Ce fichier permettra d'exporter les fichiers Gerber à partir de Eagle avant de les envoyer vers le site du fabricant.

| Vous pouvez l'obtenir ici : http://www.sparkfun.com/tutorial/Eagle-DFM/sfe-gerb274x.cam

Pour l'installer, déposez-le dans **/cam**, situé dans le dossier racine Eagle de votre disque dur.

CONCEVOIR LA CARTE

Passons maintenant à la partie conception de la carte. Comme indiqué précédemment, nous nous inspirerons de la conception de la carte Arduino Pro :

| http://arduino.cc/en/Main/ArduinoBoardPro

Le modèle Arduino Pro est une carte qui comporte uniquement les composants indispensables de la carte Uno. Je souhaitais obtenir une carte similaire au départ mais avec certaines modifications :

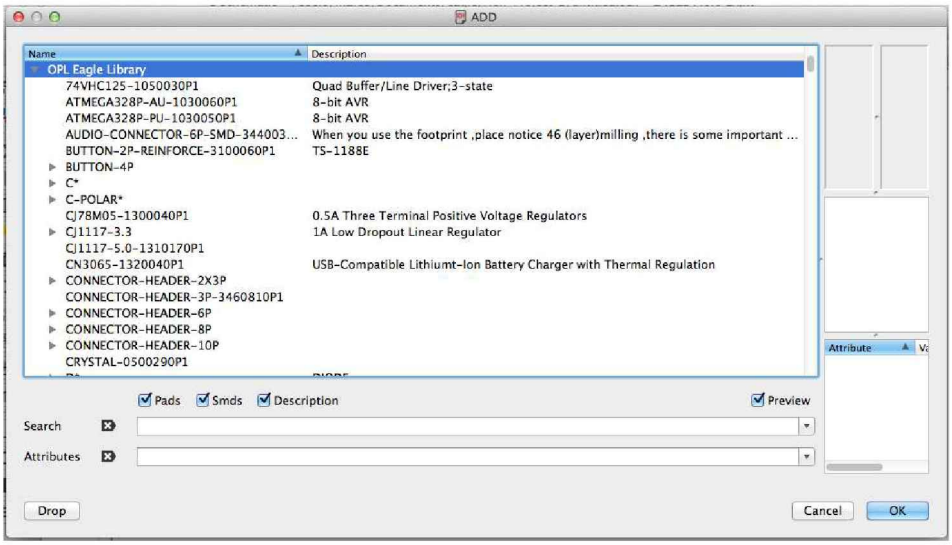
- Mon objectif était de pouvoir la programmer directement en la branchant *via* un câble micro USB, c'est-à-dire sans faire intervenir de composants externes.
- Il était important pour moi d'équiper la carte d'un mode basse consommation, soit de pouvoir déconnecter la partie programmation de la partie microcontrôleur afin d'économiser de l'énergie.
- Mon but était aussi de réduire au maximum la taille de la carte tout en assurant sa compatibilité avec les cartes d'extension Arduino existantes.

Enfin, la carte devait être en mesure de tester des projets de domotique basés sur Arduino, impliquant par exemple des détecteurs de mouvement sans fil basse consommation.

| Souvenez-vous que chacun des composants utilisés dans le projet doit être issu de la bibliothèque « Open Parts Library » :
| http://www.seeedstudio.com/wiki/Open_parts_library

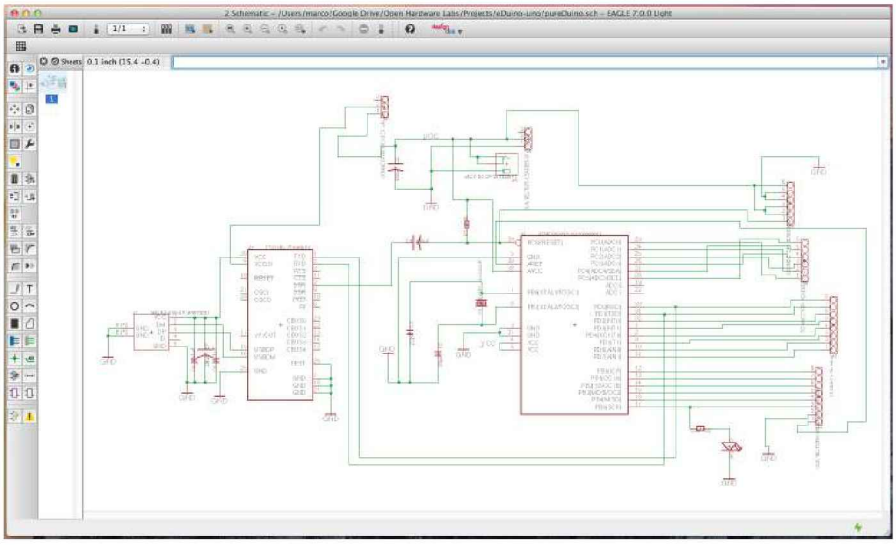
Confectionnons d'abord le schéma de la carte à l'aide de composants exclusivement tirés de la bibliothèque mentionnée ci-dessus.

Cliquez sur « Add » afin d'insérer un nouveau composant au schéma et parcourez la bibliothèque.



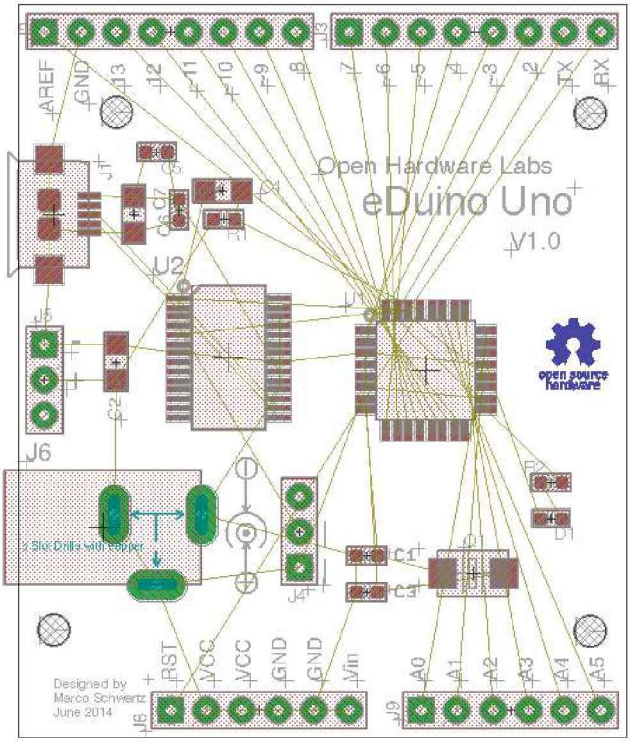
Effectuez cette étape pour tous les composants et connectez-les ensemble.

Dans ce chapitre, je souhaite me concentrer sur le processus de conception de la carte et ne pas m'attarder sur l'ajout de chacun des composants. La conception du schéma dépend vraiment de la carte que vous souhaitez obtenir. Cette capture d'écran illustre le schéma complet :



Traitons à présent le schéma de circuits imprimés de la carte. L'objectif ici est d'agencer les différents composants de sorte à réduire le plus possible la surface du circuit imprimé. En revanche, la compatibilité avec les cartes d'extension Arduino existantes était dans mon cas un facteur important. J'ai dû ainsi gérer le peu d'espace disponible pour insérer les connecteurs permettant le raccord aux cartes d'extension.

Voici une capture d'écran du schéma de circuits imprimés (circuit non routé) :



Enfin, pour router ma carte, j'ai exécuté le routeur automatique du logiciel Eagle. La capture d'écran de la page suivante illustre le schéma de circuits imprimés final de ma carte.

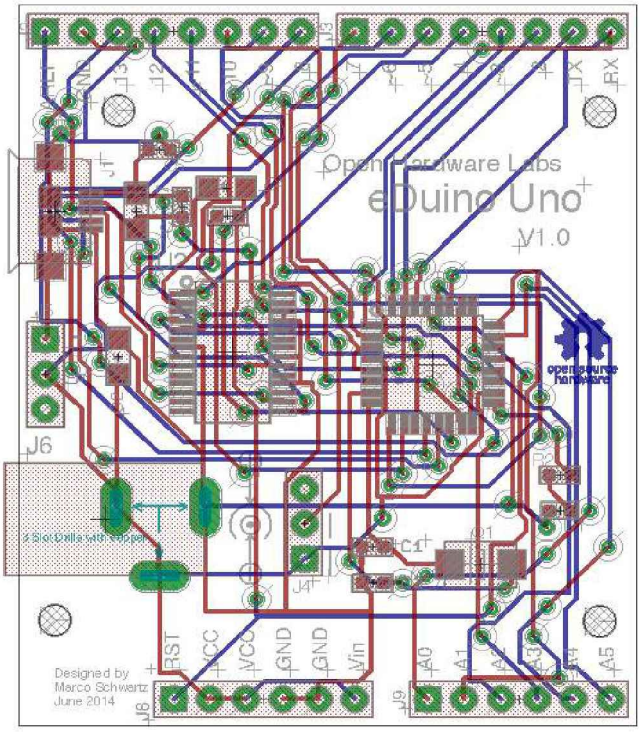
Vous pouvez retrouver le schéma complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/pcb-book

Pour mettre un terme à la phase de conception de la carte, vérifions sa conformité avec certaines règles veillant à assurer la fiabilité de notre circuit imprimé.

On procède alors à ce que l'on appelle la vérification des règles de dessin (*Design Rule Checking*), qui peut être effectuée automatiquement par Eagle.

Je recommande d'utiliser les règles fournies par le détaillant de composants électroniques Sparkfun : http://www.sparkfun.com/tutorial/Eagle-DFM/SparkFun.dru

Téléchargez le fichier avant de le glisser dans le dossier **/dru** de Eagle. Sélectionnez ensuite Tools > DRC. C'est à cet endroit que vous chargerez le fichier que vous venez de télécharger. Après ça, cliquez sur « Check » pour lancer la vérification. Si des erreurs sont détectées, leur descriptif apparaîtra. Il vous faudra répéter cette action jusqu'à la validation finale.



FABRIQUER LA CARTE

Une fois la partie conception de la carte achevée, concentrons-nous sur sa fabrication.

Exportez tout d'abord le fichier dans un format compatible avec un service de fabrication de PCB. La plupart des fabricants de circuits imprimés travaillent à partir de fichiers au format Gerber. Par chance, Eagle permet d'exporter très facilement votre schéma dans ce format.

Premièrement, passez en mode schéma et sélectionnez **File > CAM Processor**. Cette action ouvrira une nouvelle fenêtre où vous pourrez exporter vos fichiers vers le format Gerber. Certaines sociétés vous épargnent cette opération car elles ont construit des modèles pour le processeur « CAM Processor » de Eagle.

> Je vous conseille de télécharger les paramètres CAM de Sparkfun :
> http://www.sparkfun.com/tutorial/Eagle-DFM/sfe-gerb274x.cam

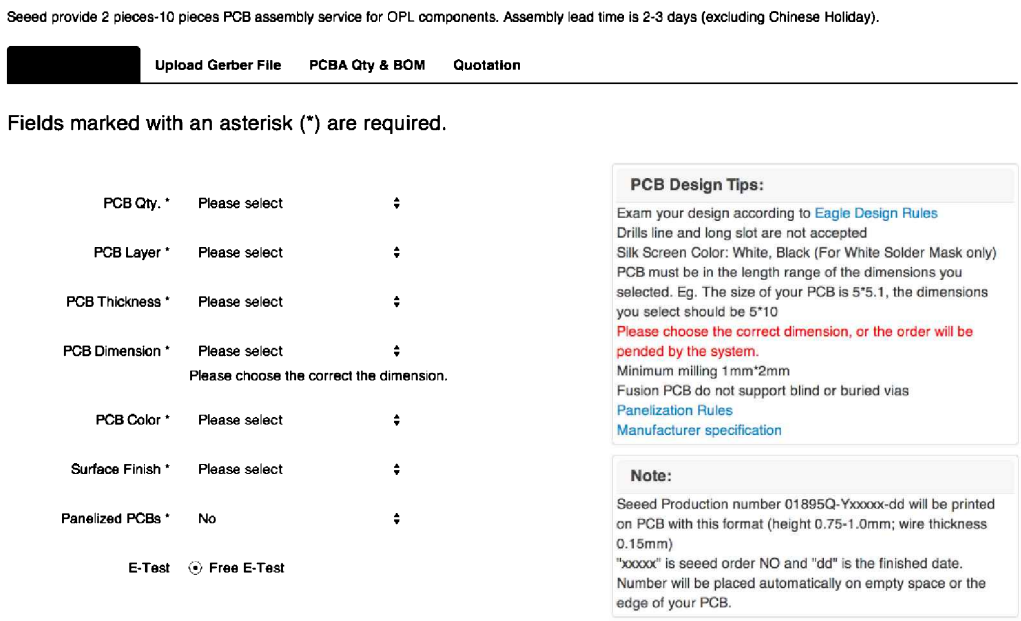
Sélectionnez ensuite **File > Open > Job** pour ouvrir le fichier que vous venez de télécharger. Tout est déjà configuré pour vous, il vous reste à cliquer sur « Process Job » pour lancer l'exportation des fichiers.

Vous retrouverez l'ensemble des fichiers créés dans le dossier où figure votre schéma. Stockez-les enfin dans un fichier Zip. C'est bien ce fichier compressé que nous enverrons au fabricant.

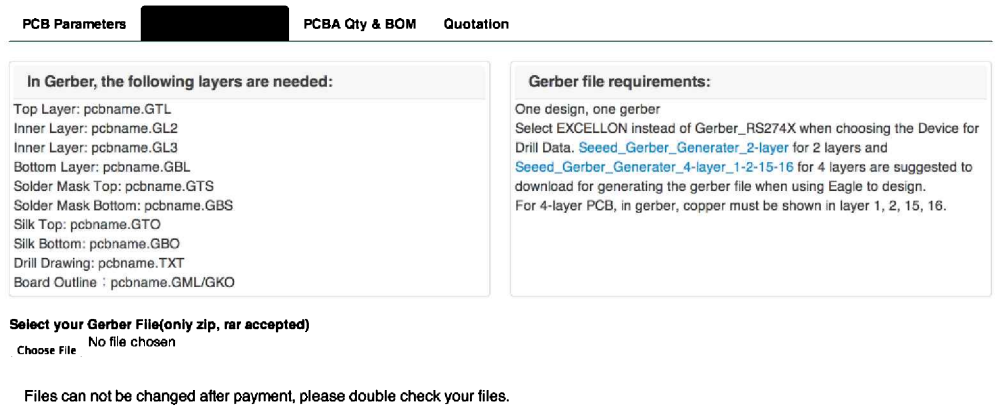
Comme mentionné auparavant, nous allons recourir au service d'assemblage de circuits imprimés de Seeedstudio pour la fabrication de nos cartes.

Suivez ce lien pour commencer :
http://www.seeedstudio.com/service/index.php?r=site/pcbService&type=pcba

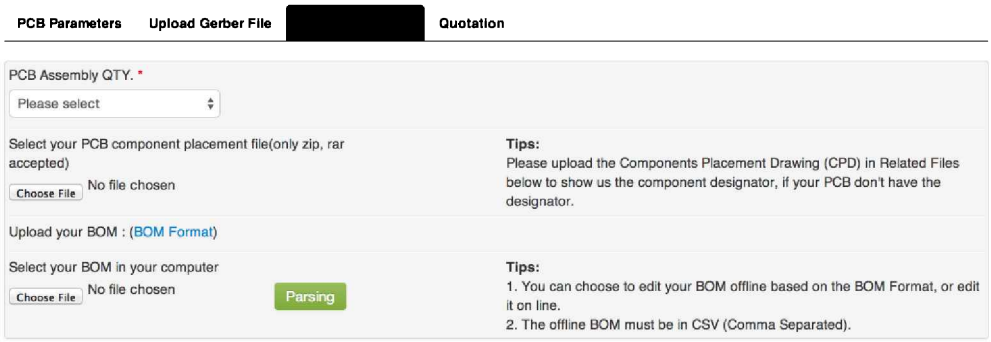
Certaines informations relatives à votre projet vous seront alors demandées :



Soyez vigilant car vous n'aurez plus la possibilité de les modifier ultérieurement. Chargez ensuite votre fichier Zip contenant vos fichiers au format Gerber :



L'étape suivante consiste à créer ce que l'on appelle la nomenclature de votre projet, appelée « BOM » :



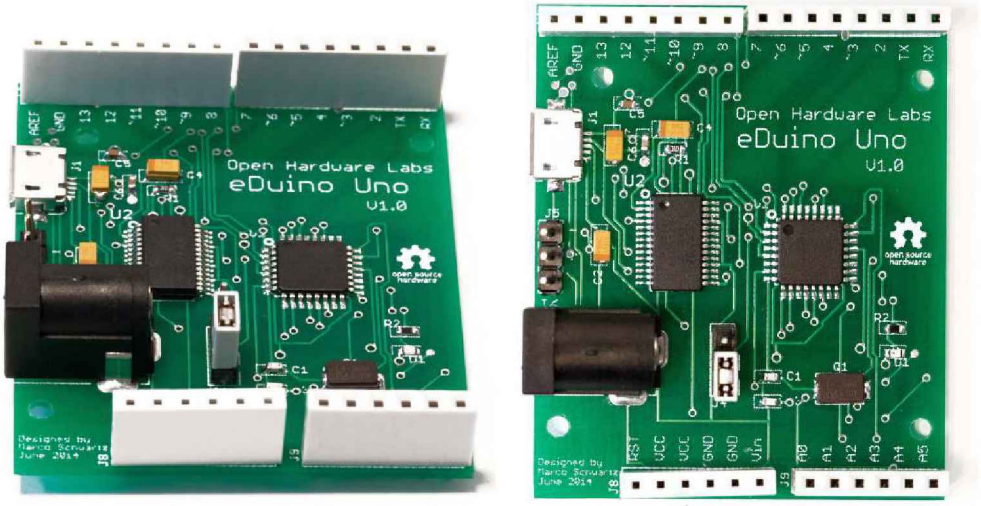
Cette nomenclature renferme toutes les informations propres aux composants utilisés dans votre schéma. Vous pouvez la télécharger depuis le logiciel Eagle.

Notez qu'il n'est pas rare de devoir appliquer des modifications après le chargement, c'est pourquoi je vous recommande d'utiliser le service en ligne de SeeedStudio prévu à cet effet :

Il ne vous reste plus qu'à entrer votre adresse de livraison et à régler la commande.

RÉSULTAT FINAL

Vous devriez recevoir votre circuit imprimé après quelques semaines. Voici un aperçu de ma carte, vue de côté et vue de dessus.



Il est temps maintenant d'effectuer quelques tests. Une fois de plus, la nature des tests dépend de votre projet. Dans notre exemple, j'ai simplement exécuté les sketches Arduino que j'utilise habituellement pour effectuer des tests. J'ai choisi de procéder ainsi car cette carte est très complète. Elle s'utilise comme une carte Arduino Uno.

Pour aller plus loin

Dans ce chapitre, nous avons suivi ensemble toutes les étapes permettant d'obtenir une carte Arduino personnalisée, de la conception à la fabrication. Vous avez appris à manier le logiciel Eagle afin de concevoir une carte dans son intégralité en vous inspirant de projets impliquant du matériel libre. Vous avez pu observer la manière dont on exporte le schéma de la carte vers un format pris en charge par la plupart des fabricants. Nous avons fait appel à un service d'assemblage et de fabrication de circuits imprimés, ce qui nous évite de souder nous-mêmes les composants sur les cartes.

Vous pouvez partir de ce projet pour mettre au point vos propres cartes destinées à des applications en domotique basées sur Arduino. Libre à vous de modifier le schéma de la carte prise comme exemple afin de réaliser votre propre système. Il vous est possible de retirer certains composants tels que les barrettes de connexion et de réduire la taille de la carte. De cette manière, vous pouvez créer une carte basse consommation pour la domotique. Vous avez également la possibilité de créer votre propre système en intégralité.

Partie V

Construire ses propres boîtiers en impression 3D

21 Imprimer un boîtier simple pour Arduino

Ce chapitre vous explique comment imprimer votre premier modèle d'objet 3D pour un projet de domotique. Après avoir défini les caractéristiques de notre boîtier, nous partirons à la recherche du modèle prêt à imprimer le mieux adapté à vos besoins.

Dans le cas où vous ne posséderiez pas d'imprimante 3D chez vous, nous ferons appel à un service d'impression 3D en ligne. À titre d'exemple, je vous montrerai comment je suis parvenu à intégrer une carte Arduino et des composants au sein du modèle de boîtier.

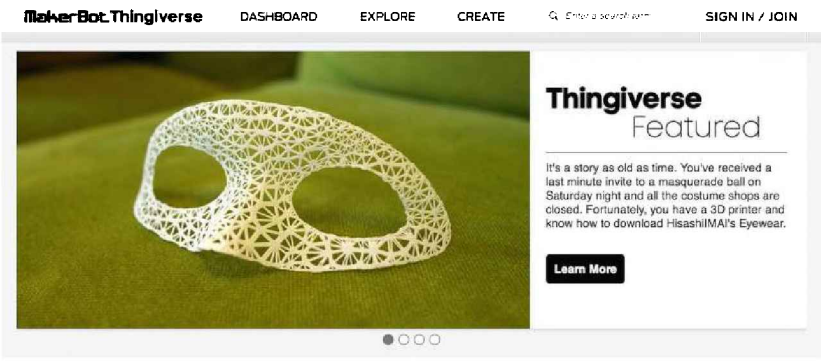
S'APPROPRIER UN MODÈLE EXISTANT

Il est important en premier lieu de définir vos attentes, votre « cahier des charges », par rapport à ce projet. Ensuite, nous verrons comment trouver des modèles existants, prêts à être imprimés.

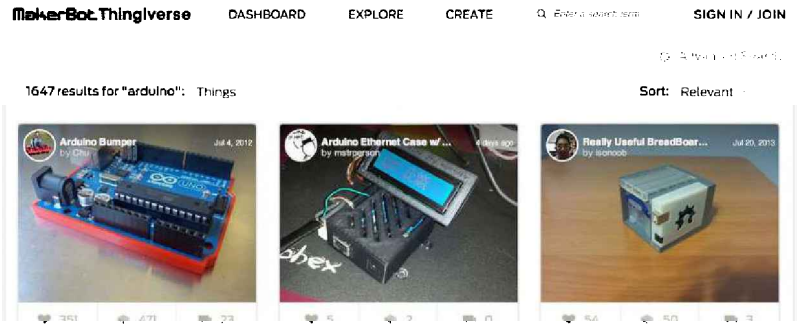
J'essaierai de faire au plus simple pour ce chapitre. Nous souhaitons obtenir un boîtier pouvant accueillir une carte Arduino (en laissant la prise USB apparente) mais aussi quelques câbles et composants.

Pour ce projet, nous allons partir d'un modèle 3D mis à disposition sur le web par son concepteur. Plusieurs sites Internet fournissent des modèles prêts à imprimer.

Voici l'adresse à laquelle je me rends le plus souvent : http://www.thingiverse.com/



Vous pouvez débuter votre recherche. Pour ce projet, j'ai simplement tapé « Arduino ». Voici les résultats obtenus :



Comme vous pouvez le voir, un grand nombre de modèles vous sont proposés. Cependant, certains d'entre eux ne sont pas spécifiquement conçus pour la carte Arduino et il vaut mieux les exclure. Vous pouvez voir les notes attribuées aux différents modèles, ainsi que le nombre d'utilisateurs qui les ont déjà imprimés. Selon moi, il y a sur ce site trois critères essentiels à prendre en compte pour choisir un modèle :

- Le nombre de votes favorables (> 100) et la présence de commentaires positifs.
- Le fait que le projet ait déjà été fabriqué par son auteur et par d'autres utilisateurs (images du résultat final à l'appui).
- Le projet est disponible au format STL (pour imprimantes 3D), open source, c'est-à-dire qu'il peut librement être modifié (comme le format OpenSCAD dont nous parlerons plus loin dans le livre).

En appliquant ces critères, mon choix s'est porté sur ce boîtier :

Vous pouvez télécharger son modèle à l'adresse suivante :
http://www.thingiverse.com/thing:20739

Toujours sur la même page, vous avez la possibilité de télécharger les fichiers STL et de passer à l'étape suivante : la fabrication de votre objet.


Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre :
https://github.com/openhomeautomation/3d-printing-book

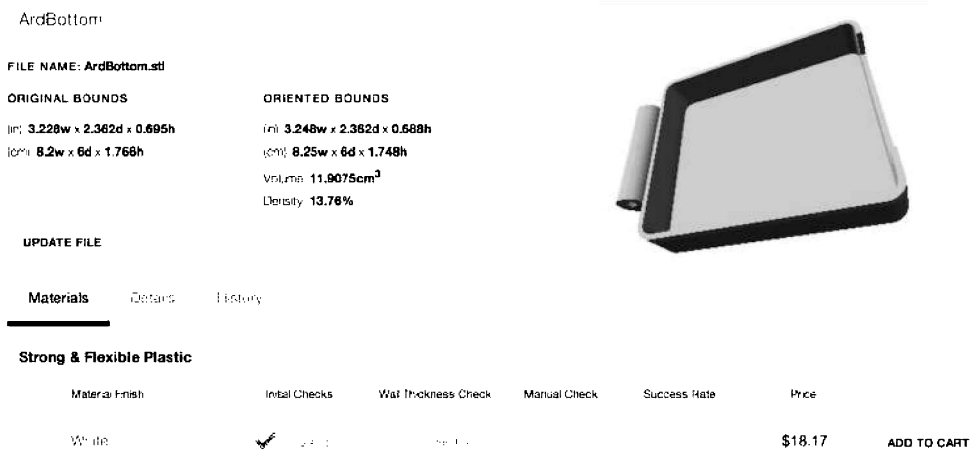
IMPRIMER VOTRE BOÎTIER VIA UN SERVICE EN LIGNE

Voyons à présent la procédure à suivre pour imprimer le modèle que vous venez de sélectionner *via* un service d'impression 3D en ligne. Passez à l'étape suivante si vous possédez votre propre imprimante 3D ou si vous faites partie d'un hackerspace ou d'un Fab Lab.

Pour ce chapitre, comme pour l'ensemble de l'ouvrage, j'ai choisi le service en ligne Shapeways.

Pour plus d'informations sur les services proposés : http://www.shapeways.com/

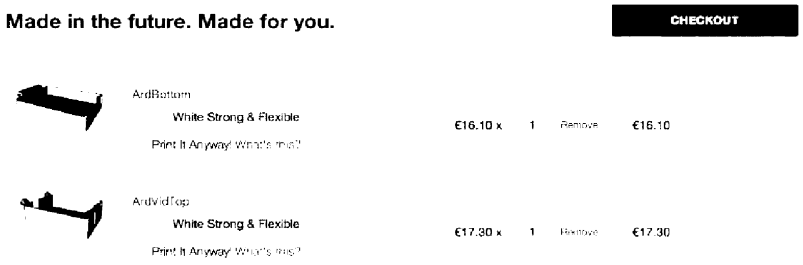
Libre à vous de choisir un autre service d'impression 3D pour la fabrication de votre objet. Pour vous aider, plusieurs adresses sont à votre disposition dans le chapitre *Ressources*. Créez un compte sur le site. Accédez ensuite à l'interface de chargement et sélectionnez votre premier fichier STL :



Vous devrez alors choisir le matériau qui composera votre objet (j'ai opté ici pour du plastique blanc). Ajoutez ensuite le modèle à votre panier.

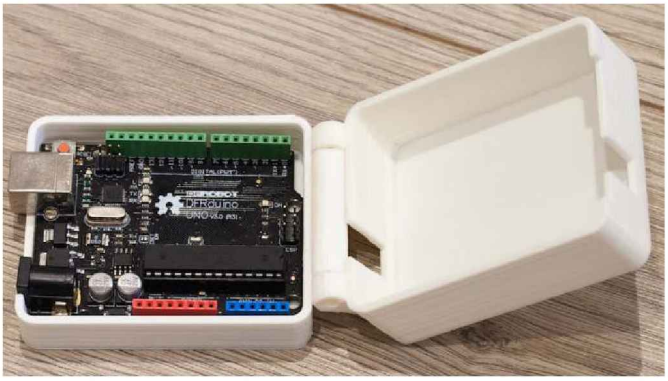
Répétez l'opération pour chacun des éléments qui le composent (deux éléments au total dans mon cas).

Finalisez ensuite la commande :



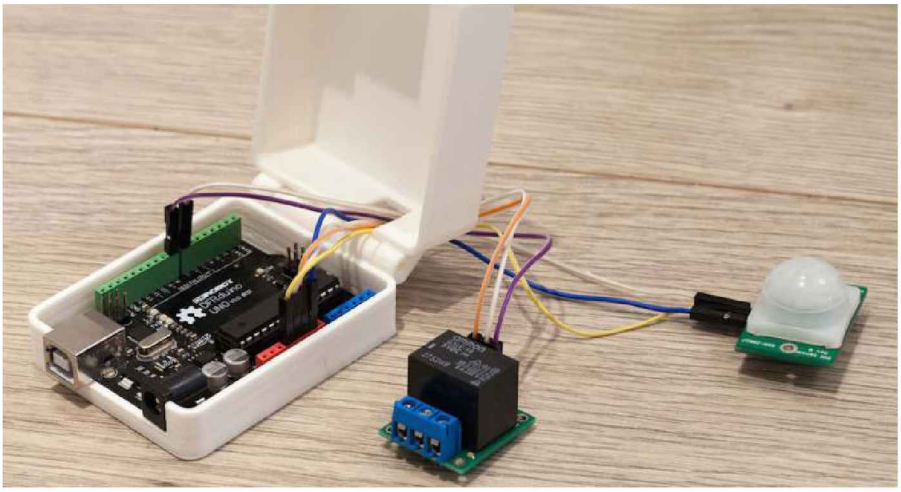
Vous devriez recevoir votre objet en pièces détachées sous une quinzaine de jours (selon ma propre expérience). Voici le contenu du colis que j'ai reçu :

Comme vous pouvez le remarquer, l'objet est une réplique fidèle du modèle téléchargé sur Thingiverse. Il nous faut maintenant tester l'objet en y insérant tout simplement une carte Arduino. Dans mon cas, le boîtier s'est parfaitement adapté à la carte :

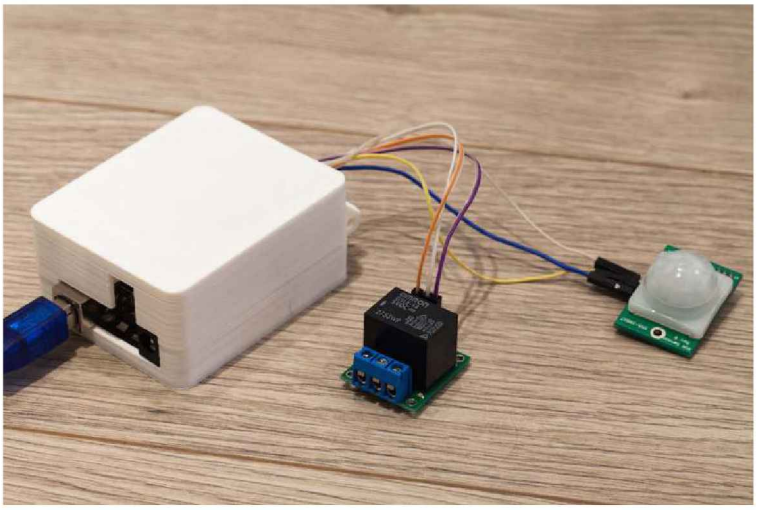


TESTER LE RÉSULTAT AVEC UN PROJET DE DOMOTIQUE

Pour finir, j'ai souhaité tester l'objet avec plusieurs composants fréquemment utilisés dans le domaine de la domotique : un détecteur de mouvement et un relais. J'ai donc laissé ma carte Arduino dans le boîtier, puis j'y ai branché les deux composants au moyen de fils de raccordement. J'ai également veillé à ce que les câbles passent par l'encoche latérale de la partie supérieure du boîtier :



Puis, j'ai rabattu la partie supérieure pour le refermer :



Vous pouvez constater que le résultat est bien plus élégant avec, plutôt que sans le boîtier.

POUR ALLER PLUS LOIN

Dans ce chapitre, nous avons vu comment choisir un modèle de boîtier sur la toile. Nous avons utilisé un service d'impression 3D en ligne pour fabriquer notre boîtier. Une fois reçu, nous y avons installé notre projet.

Vous pouvez consulter des sites web semblables à Thingiverse afin de trouver des modèles qui vous plaisent. Non seulement vous accéderez à des modèles adaptés à vos projets, mais vous trouverez aussi de l'inspiration pour créer vos propres réalisations par la suite.

22 Modifier un modèle existant

Dans ce chapitre, nous allons franchir une nouvelle étape en modifiant un modèle 3D existant. Définissons une nouvelle fois nos attentes et choisissons un modèle qui pourra être modifié avec le logiciel OpenSCAD.

Nous y apporterons des modifications en fonction des caractéristiques du système que nous souhaitons y insérer, puis nous le ferons imprimer *via* un service d'impression 3D.

La dernière étape consistera à le tester.

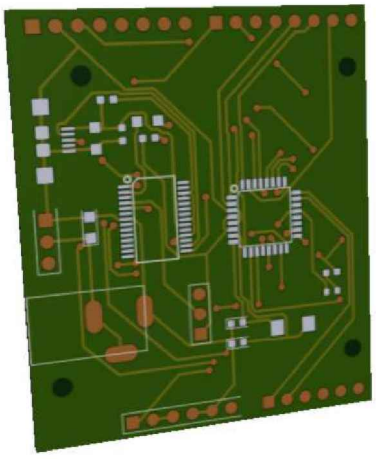
CONCEVOIR UNE CARTE ARDUINO ET CHOISIR UN BOÎTIER

Comme pour le chapitre précédent, il est nécessaire de définir notre besoin.

Dans mon cas, le but était d'obtenir un boîtier pour une carte compatible avec Arduino que j'avais récemment réalisée dans le cadre de projets de domotique basse consommation.

Étant donné que la taille de ma carte différait légèrement de celle d'un modèle Arduino Uno (même si elle disposait des mêmes propriétés), il me fallait un boîtier sur mesure.

Voici une représentation 3D de ma carte :



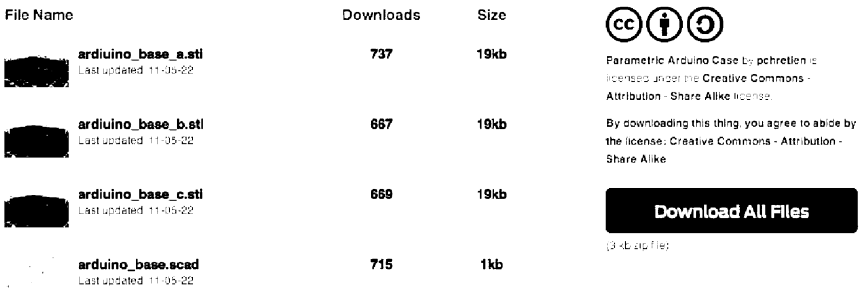
Vous êtes bien entendu libre de concevoir intégralement votre boîtier mais il faudra vous armer de patience. De plus, le risque de commettre une erreur est assez élevé. Je vous propose donc une alternative qui consiste à personnaliser un modèle existant.

Je vous suggère de parcourir une nouvelle fois le site Thingiverse pour trouver un boîtier adapté. Cette fois-ci, vous devrez trouver un modèle qui a été préalablement testé, produit et qui est facilement modifiable. Je vous conseille de télécharger un fichier compatible avec OpenSCAD (le logiciel que nous utiliserons ici).

Je vous propose de choisir ce boîtier de conception paramétrique :

Ce boîtier est disponible à l'adresse suivante : http://www.thingiverse.com/thing:8706

En dessous du descriptif de l'objet, vous remarquerez dans la section téléchargement (« Downloads ») que le modèle est disponible au format **.scad** (OpenSCAD) :



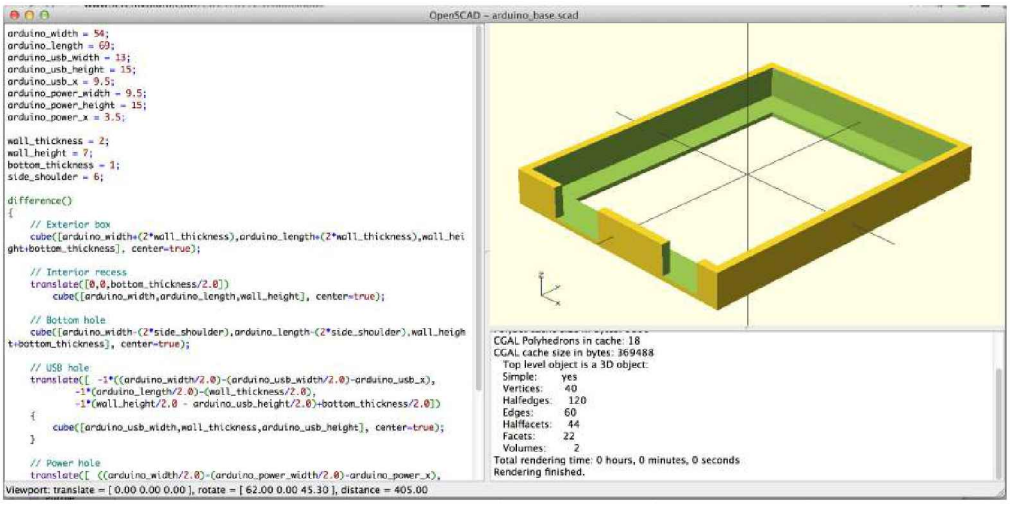
Le fichier étant désormais téléchargé, je vais maintenant pouvoir l'arranger à ma façon.

Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/3d-printing-book

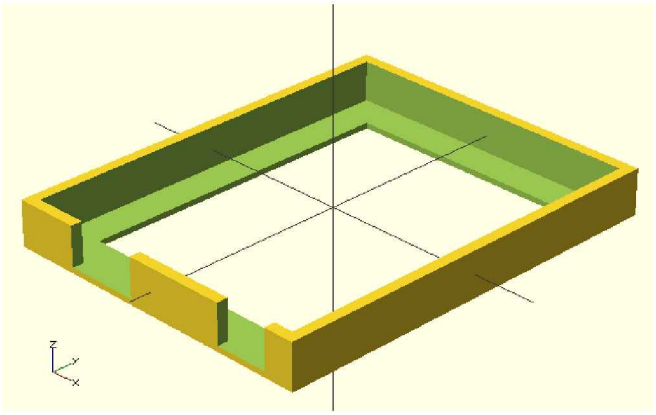
PERSONNALISER LE BOÎTIER

Il nous faut à présent opérer des modifications sur le fichier 3D afin de l'adapter aux dimensions du système que l'objet va accueillir. Ouvrez le fichier avec OpenSCAD.

Vous devriez obtenir ceci :



Deux sous-fenêtres apparaissent : celle de gauche permet de modifier le script du fichier. La sous-fenêtre de droite permet de visualiser l'objet en 3D. Voici le rendu visuel initial de l'objet.

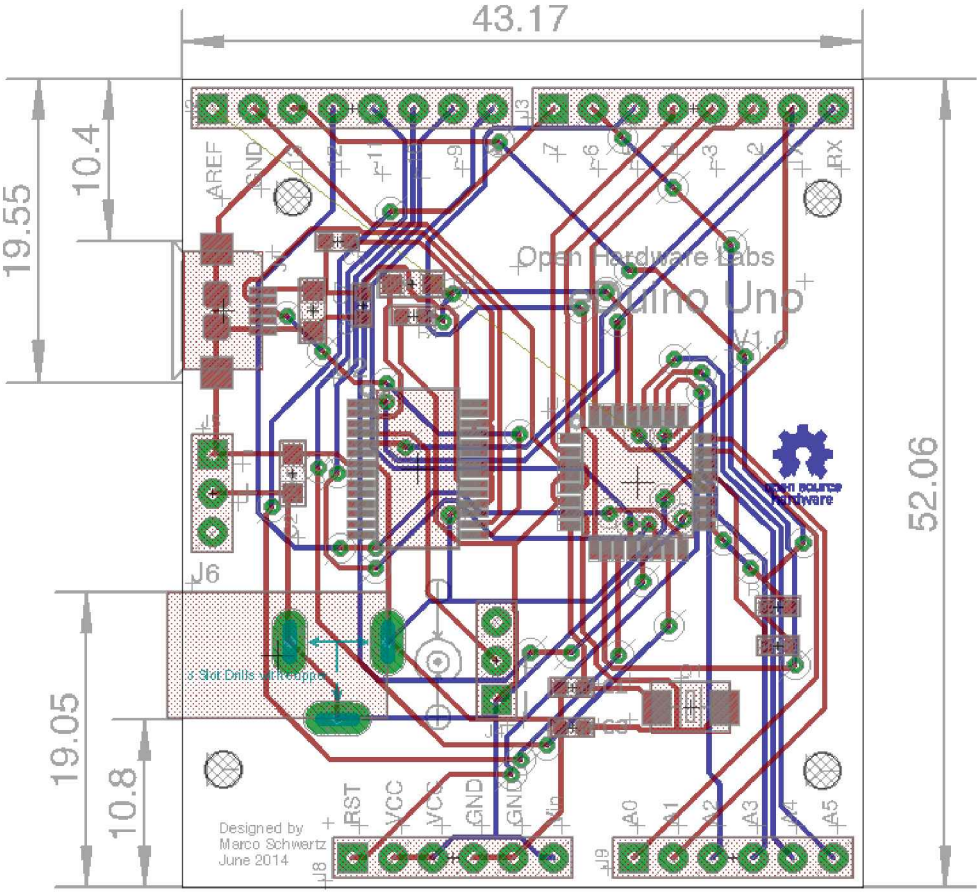


Il ne sera pas question ici de manipuler ou d'ajouter des lignes de code. Nous nous contenterons simplement de changer les variables du début de fichier qui définissent les dimensions du matériel que va accueillir l'objet :

```
arduino_width = 54;
arduino_length = 69;
arduino_usb_width = 13;
arduino_usb_height = 15;
arduino_usb_x = 9.5;
arduino_power_width = 9.5;
arduino_power_height = 15;
arduino_power_x = 3.5;
wall_thickness = 2;
wall_height = 7;
bottom_thickness = 1;
side_shoulder = 6;
```

Il nous faut donc prendre ces dimensions. Pour ma part, il a fallu modifier la taille de la carte, celle du port USB et du connecteur d'alimentation et leur emplacement.

Pour connaître les dimensions de ma carte, j'ai simplement ouvert son fichier sur le logiciel Eagle, puis j'ai pris les mesures correspondantes :

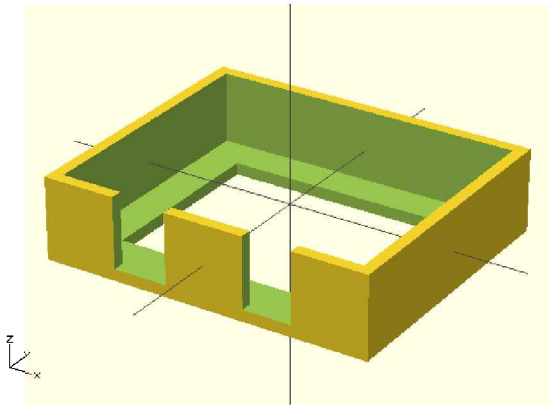


Une fois en possession des bonnes mesures de votre matériel, saisissez-les dans OpenSCAD.

Voici donc les dimensions mises à jour selon mon matériel :

```
arduino_width = 52.1;
arduino_length = 43.2;
arduino_usb_width = 19.55 - 10.4;
arduino_usb_height = 15;
arduino_usb_x = 10.4;
arduino_power_width = 19.05 - 10.8;
arduino_power_height = 16;
arduino_power_x = 10.8;
wall_thickness = 2;
wall_height = 12;
bottom_thickness = 2;
side_shoulder = 6;
```

Notez que je souhaitais épaissir la couche inférieure de mon objet et lui donner un peu de hauteur. Voici un aperçu du modèle modifié :



Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/3d-printing-book

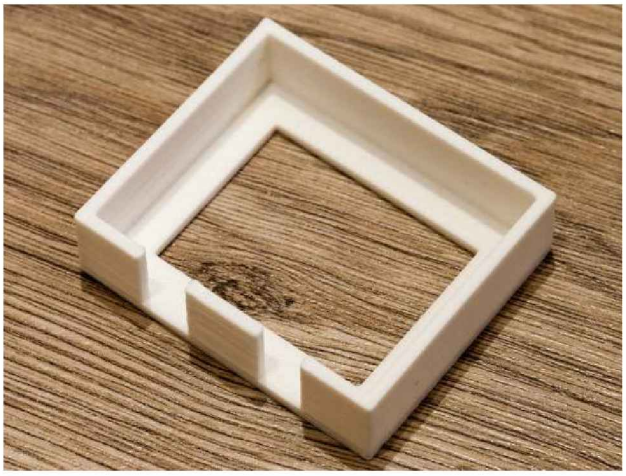
Exportez le fichier au format STL pour pouvoir l'imprimer par la suite. Pour cela, sélectionnez **File > Export > Export as STL**.

FABRIQUER LE BOÎTIER

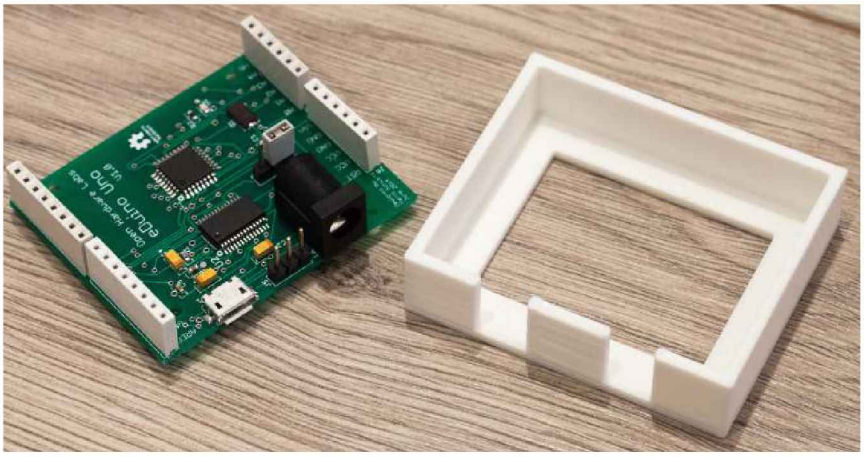
Une fois la partie modélisation achevée, il ne vous reste plus qu'à imprimer l'objet 3D. Pour cette opération, j'ai de nouveau fait confiance au site Shapeways.

Vous pouvez bien entendu utiliser votre imprimante 3D, si vous en possédez une.

Voici un aperçu de l'objet reçu deux semaines plus tard :



Le voilà maintenant à côté de la carte :



POUR ALLER PLUS LOIN

Dans ce chapitre, nous avons vu quel type de modèle 3D existant il est conseillé de choisir en vue de le modifier par la suite. Vous avez appris à rectifier un modèle 3D suivant les caractéristiques prédéfinies de votre objet. Pour ce faire, vous avez simplement édité son code dans le logiciel OpenSCAD avant de l'imprimer en 3D.

Vous pouvez réaliser des projets plus évolués à partir des principes abordés ici. Parcourez par exemple la bibliothèque du site Thingiverse pour dénicher des modèles correspondant à vos besoins. N'oubliez pas qu'il est préférable de choisir des modèles pris en charge par OpenSCAD. Personnalisez ensuite ces modèles pour les adapter à vos projets de domotique.

23 Concevoir un boîtier pour des capteurs

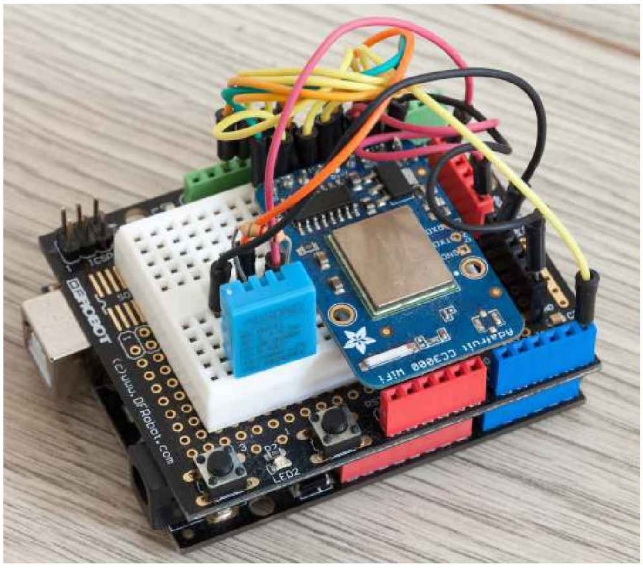
Dans ce dernier chapitre, outre la modification des dimensions d'un modèle, il sera question d'ajouter de nouveaux éléments.

Le boîtier que nous allons concevoir accueillera une installation domotique complète basée sur Arduino, impliquant un capteur de température et d'humidité doté d'une fonction WiFi. Le système Arduino comportera une carte Uno, une carte d'extension de prototypage (shield de prototypage), quelques composants et des fils de raccordement.

CHOISIR UN BOÎTIER ADAPTÉ À UN SYSTÈME ARDUINO

Comme toujours, commençons par définir nos besoins. Mon désir était de réaliser une installation domotique permettant de mesurer et de transférer en WiFi les données de température et d'humidité d'une pièce.

Voici un aperçu du montage terminé, vous pouvez constater que le rendu est peu esthétique.



Je souhaitais donc me procurer un boîtier adapté à cette installation afin de dissimuler le câblage du projet tout en laissant le port USB et le connecteur d'alimentation accessibles.

Le boîtier devait aussi disposer de quelques ouvertures nécessaires au bon fonctionnement du capteur.

J'ai pris les mesures de la carte Arduino, de la carte d'extension et de l'ensemble des composants qui allaient me servir lors de la modification du modèle de l'objet.

Étant donné que mon projet s'appuie sur une carte Arduino Uno, j'ai utilisé le même modèle que pour le chapitre précédent.

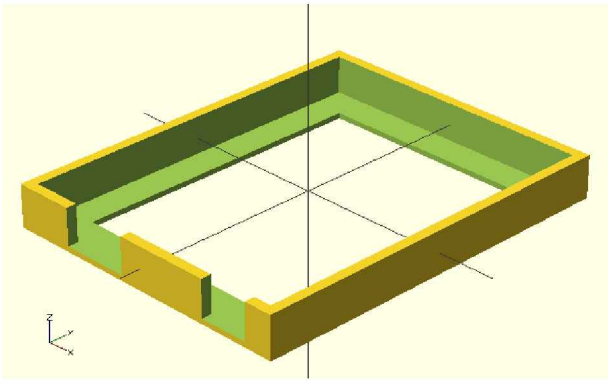
> Rappelez-vous que celui-ci est disponible à l'adresse suivante :
> http://www.thingiverse.com/thing:8706

J'ai ainsi téléchargé le fichier au format pris en charge par OpenSCAD afin de pouvoir le personnaliser par la suite.

> Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre :
> https://github.com/openhomeautomation/3d-printing-book

CONCEVOIR LE BOÎTIER

Ouvrez à présent le fichier sous OpenSCAD pour appliquer les modifications. Voici le modèle sous OpenSCAD :



Afin d'adapter notre modèle à nos besoins initiaux, trois opérations sont nécessaires :

- adapter les dimensions à celles de la carte d'extension et des autres composants que comportera le boîtier ;
- ajouter des ouvertures latérales au boîtier pour laisser l'air rentrer ;
- concevoir un couvercle.

Ajustons d'abord les dimensions de l'objet, notamment sa hauteur.

Il vous suffit de modifier les variables du début de fichier :

```
arduino_width = 57;
arduino_length = 69;
arduino_usb_width = 13;
arduino_usb_height = 50;
arduino_usb_x = 9.5;
arduino_power_width = 9.5;
arduino_power_height = 50;
arduino_power_x = 6.5;
wall_thickness = 2;
wall_height = 50;
bottom_thickness = 1;
side_shoulder = 6;
```

Les dimensions ont été légèrement augmentées afin de pouvoir ajouter une carte d'extension au-dessus de la carte Arduino.

La hauteur du boîtier a également été remaniée afin d'incorporer quelques composants et fils de raccordement au projet.

En ce qui concerne les ouvertures latérales d'aération, il nous faut dans un premier temps définir les dimensions appropriées :

```
side_hole_width = 5;
side_hole_height = 30;
```

Pour incorporer des ouvertures à l'objet, il est nécessaire de créer un « cube », de le positionner à l'emplacement de l'ouverture, puis de le soustraire à l'objet pour former ainsi un espace vide.

Répétez cette étape autant de fois que nécessaire.

Créez d'abord un cube de la longueur totale du boîtier et aux dimensions définies précédemment :

```
cube([500,side_hole_width,side_hole_height], center=true);
```

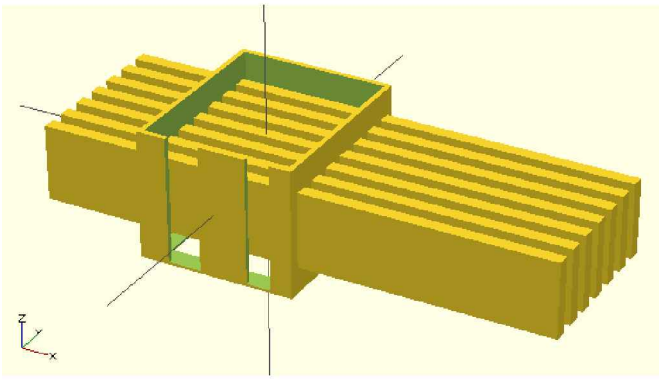
Effectuez ensuite une translation du cube afin de le positionner à l'emplacement prévu pour l'ouverture. Notez que nous avons déjà introduit un index appelé `i`, nous pouvons donc effectuer une translation du cube un peu plus tard :

```
translate([arduino_width/2.0,side_hole_width-arduino_length/2 +
i*10,0]){
  cube([500,side_hole_width,side_hole_height], center=true);
}
```

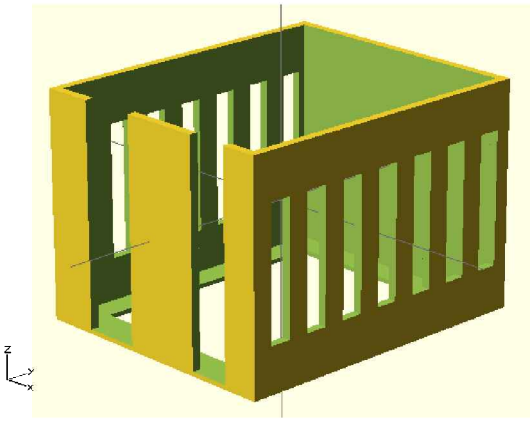
Après cette étape, on introduit la boucle `for` qui se charge de copier les cubes de manière automatique en les séparant d'un espace régulier :

```
for ( i = [0 : 1 : 6] ){
  translate([arduino_width/2.0,side_hole_width-arduino_length/2 +
i*10,0]){
    cube([500,side_hole_width,side_hole_height], center=true);
  }
}
```

Voici un aperçu de ce que l'on obtient à ce stade :



Ensuite, il nous faut simplement faire intervenir la fonction `difference()` pour soustraire les cubes au bloc principal. Voici un aperçu de la version finale du modèle :

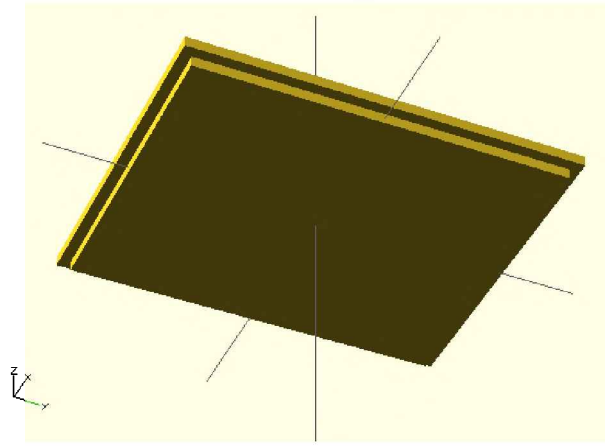


Occupons-nous maintenant de la conception du couvercle qui est assez simple car son seul rôle consiste à recouvrir la partie inférieure.

L'objet se compose de deux parties aux dimensions inégales, emboîtées l'une sur l'autre :

```
arduino_width = 57;
arduino_length = 69;
wall_thickness = 2;
cube([arduino_width,arduino_length,wall_thickness], center=true);
translate([0,0,wall_thickness]){
  cube([arduino_width+wall_thickness*2,
    arduino_length+wall_thickness*2,wall_thickness],
    center=true);
}
```

Voici un aperçu de la version finale du couvercle :



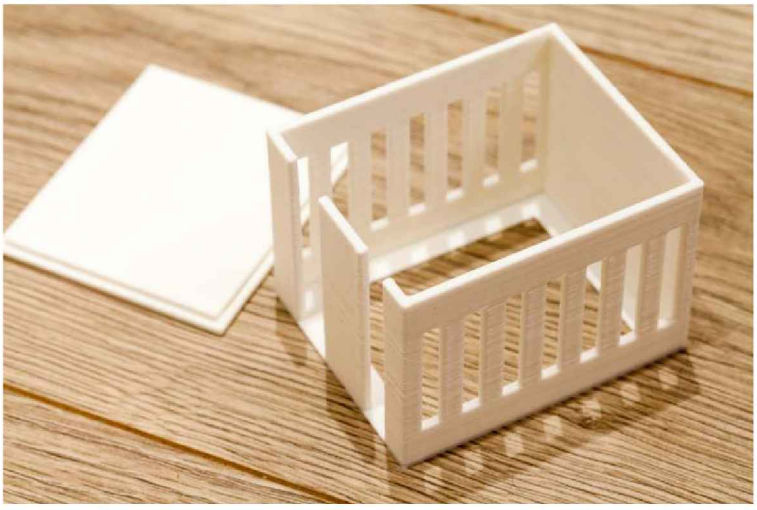
Vous pouvez retrouver le code complet du chapitre dans le dépôt GitHub du livre : https://github.com/openhomeautomation/3d-printing-book

FABRIQUER ET TESTER LE BOÎTIER

Nous pouvons désormais lancer la fabrication de notre modèle.

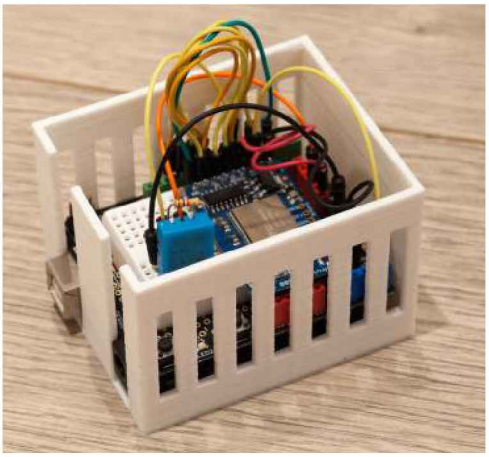
J'ai fait appel au site Shapeways pour la fabrication de l'objet mais vous pouvez l'imprimer chez vous si vous êtes équipé.

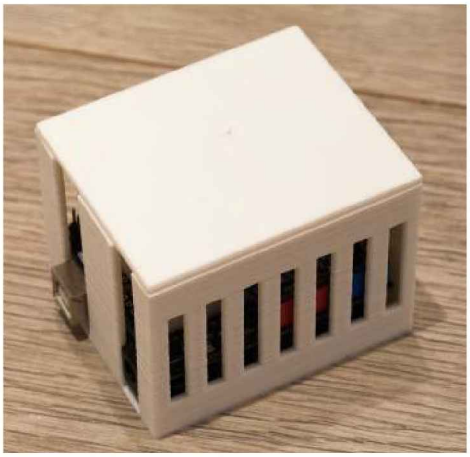
Voici un aperçu de l'objet une fois imprimé :



Je l'ai testé sans attendre avec l'installation domotique mentionnée plus haut.

Et voici un aperçu du boîtier (sans, puis avec son couvercle) contenant l'installation domotique :





Comme vous le voyez, l'utilisation du boîtier n'altère en rien les fonctions de l'installation domotique. Il améliore l'aspect du montage.

Pour aller plus loin

Dans ce chapitre nous avons vu comment choisir puis modifier un modèle 3D existant. Dans le fichier 3D, nous avons non seulement adapté les dimensions du modèle suivant nos besoins, mais nous avons également ajouté de nouveaux éléments tels que les ouvertures latérales. Nous avons ensuite conçu la partie supérieure du boîtier qui viendra recouvrir l'installation lorsque celle-ci sera en fonctionnement.

Je vous invite maintenant à mettre en forme vos propres concepts et idées, de la modélisation à la fabrication, en vous aidant du logiciel OpenSCAD (ou d'un logiciel équivalent). La conception de modèles 3D dédiés à la domotique est un travail passionnant. Je vous recommande fortement de vous lancer dans ce domaine à partir des connaissances acquises dans ce livre.

Conclusion

Vous disposez à présent de toutes les clés pour laisser libre cours à votre imagination et créer vos propres installations domotiques. La première question qui se pose est : par où commencer ?

Je vous conseille de commencer par de petits défis. Y a-t-il un projet que vous aviez commencé avant de lire ce livre mais auquel vous avez toujours dû renoncer, faute d'expérience ? Il est temps de vous lancer. Il n'est évidemment pas conseillé, par exemple, de commencer par la conception et la fabrication d'un système de sécurité complet pour votre domicile. Essayez plutôt de connecter un capteur à une carte Arduino qui sera chargé de détecter l'ouverture d'une porte par exemple. Ajoutez ensuite d'autres capteurs au projet. Si vous y parvenez, complétez le projet avec un afficheur LCD et quelques boutons afin d'obtenir une interface simple. Après cela, connectez-la à votre ordinateur à partir des compétences développées grâce au livre. Vous parviendrez assez rapidement à obtenir un système de sécurité complet.

Mon dernier conseil : amusez-vous. Si vous avez choisi de réaliser les projets du livre, il est très probable que vous projetiez de créer vos propres installations domotiques plutôt que d'acheter un système vendu dans le commerce avec comme ambition principale de tester de nouvelles choses et d'avoir le contrôle de votre installation. Plus vous prendrez de plaisir à mettre au point vos montages, plus vous avez de chances de réussir. Peu importe si votre premier montage n'est pas parfait : prenez le temps de le mettre à l'essai, de l'ajuster et de jouer avec. Vous gagnerez en confiance et prendrez du plaisir à la tâche ce qui vous poussera à vous lancer dans l'élaboration d'installations plus complexes.

Je terminerai en évoquant le futur de la domotique impliquant du matériel libre. Je remarque plusieurs tendances dont le domaine de la domotique pourrait pleinement profiter.

La première n'est autre que l'objet du dernier chapitre du livre, c'est-à-dire **l'impression 3D**. Cette technique permet le prototypage rapide d'objets de petite taille. Un certain nombre de designers ont d'ores déjà recours à l'impression 3D pour mettre au point leurs nouveaux produits plus rapidement. Cette technique est plus appropriée à la fabrication de petits objets que ne l'est le moulage par injection. En effet, elle est plus abordable et l'imprimante présente l'avantage de tenir sur un bureau.

Mais qu'apporte-t-elle au monde de la domotique ? Elle va selon moi révolutionner la domotique open source. Grâce aux projets du livre, vous pouvez mettre au point un système d'alarme basique à installer chez vous, possédant les mêmes fonctionnalités qu'un système vendu dans le commerce. Pour obtenir quelque chose de semblable, il vous manquera simplement la touche esthétique offerte par un boîtier. Si vous êtes amené à manipuler régulièrement votre système, alors il est inutile de le recouvrir. En revanche, pour une installation domotique qui durera dans le temps, cela en vaut la peine.

L'impression 3D est là pour ça selon moi. Il était impossible, il y a peu de temps encore, de concevoir soi-même un boîtier pour système d'alarme ou capteur. Il fallait pour cela construire un moule très onéreux et vous rendre dans une usine pour faire fabriquer les pièces de votre système. C'est désormais possible avec l'impression 3D . Les points de location d'imprimantes 3D se multiplient dans le monde entier. Vous pouvez donc vous approprier une machine pendant une durée limitée afin de fabriquer vos modèles. Vous pouvez également vous inscrire dans un « Fab Lab ». Vous y trouverez toute une gamme d'outils pour la fabrication de vos objets.

> Voici une liste de Fab Labs répertoriés en France, entre autres :
> http://fab.cba.mit.edu/about/labs/

Ce type de technologie permet d'offrir aux installations domotiques une esthétique professionnelle, digne des produits disponibles sur le marché.

Une autre tendance se dégage actuellement : la mise à disposition croissante d'objets connectés, plus connue sous le nom d'**Internet des objets**. La domotique a besoin d'un protocole commun pour standardiser l'ensemble des installations. Un tel projet semblerait pouvoir voir le jour dans les années à venir. Ceci aura également un impact sur le monde de la domotique impliquant des composants open source. Connecter un objet à Internet, comme un capteur par exemple, reste une initiative assez contraignante qui requiert l'utilisation d'une carte d'extension dédiée pour Arduino par exemple. Toutefois, un nombre croissant de cartes prévues pour fonctionner avec la plateforme Arduino bénéficie d'une connectivité WiFi intégrée.

Au moment d'écrire le livre, Arduino propose plusieurs cartes avec connectivité WiFi intégrée, comme le modèle Yun. Je pense qu'à terme il sera plus facile, grâce à ces cartes dites « connectées », de créer des installations domotiques pouvant interagir sans problème avec d'autres systèmes.

Ressources

Vous trouverez dans cette partie une liste de ressources concernant la domotique avec Arduino. Les références sont présentées par catégorie afin de vous faciliter l'accès à l'information.

Informations générales sur Arduino

- **Open Home Automation** : le site compagnon de ce livre, il contient de nombreux tutoriels de domotique dont la plupart utilisent Arduino.
  http://www.openhomeautomation.net/
- **Arduino** : le site de référence, à consulter en cas de doute sur une fonction. Les forums de ce site sont réputés, vous pourrez y trouver de l'aide pour vos projets.
  http://arduino.cc/
- **Instructables** : un annuaire de projets électroniques, dont de nombreux sont dédiés à la domotique avec Arduino.
  http://www.instructables.com/
- **Adafruit Learning System** : une plate-forme d'apprentissage en ligne avec une sélection d'articles de grande qualité pour des projets d'électronique en général. Nombre de ces projets utilisent la plateforme Arduino et certains portent sur la domotique.
  https://learn.adafruit.com

Composants

- **SnootLab** : un excellent site en français ou vous dénicherez la plupart des composants utilisés dans ce livre, notamment les cartes Arduino.
  http://snootlab.com/
- **GoTronic** : un autre site en français qui propose de nombreux composants.
  http://www.gotronic.fr/
- **Farnell** : un autre site semblable à GoTronic.
  http://fr.farnell.com/

Ouvrages sur Arduino

- *Programming Arduino: Getting Started With Sketches*, Simon Monk, TAB Books Inc, 2012.
  Cet ouvrage en anglaisest une excellente introduction à la plateforme Arduino.
- *Arduino Workshop: A Hands-On Introduction*, John Boxall, No Starch Press, 2013.
  Un ouvrage en anglais qui contient des projets simples, idéal pour s'initier à Arduino.
- *La boîte à outils Arduino, 120 techniques pour réussir vos projets*, Michael Margolis, 2e édition, Dunod, 2015.
  Cet ouvrage contient de nombreuses ressources sur l'Arduino.
- *Arduino, Maîtrisez sa programmation et ses cartes d'interface (shields)*, 2e édition, Christian Tavernier, EEA, Dunod, 2014.
- *Arduino : applications avancées, Claviers tactiles, télécommande par Internet, géo-localisation, applications sans fil...*, Christian Tavernier, EEA, Dunod, 2012.

Index