Le blog d'Eskimon

Sommaire

Ebook Arduino !

Rejoignez moi en live sur Twitch pour apprendre le tuto Arduino autrement ! (plus d'infos)

1. Arduino

Arduino et Ethernet : serveur

Eskimon , le ven. 16 janvier 2015 (28 commentaires)

arduino tuto

Sommaire

Dans ce chapitre, l'Arduino sera maintenant responsable de l'envoi des données vers le monde extérieur. On dit qu'elle agit en serveur. Ce sera donc un outil externe (logiciel particulier, navigateur etc) qui viendra l'interroger et à ce moment-là, elle renverra les informations demandées. On pourra aussi, via un site externe, lui envoyer des ordres pour faire des actions.

Sommaire

Préparer l'Arduino

L'utilisation de l'Arduino en mode serveur est sûrement plus courante que celle en client. Cette partie devrait donc particulièrement vous intéresser. Deux grands rôles peuvent être accomplis :

- L'envoi de données *à la demande* (l'utilisateur vient demander les données quand il les veut) ;
- La réception d'ordre pour effectuer des actions.

Ces deux rôles ne sont pas exclusifs, ils peuvent tout à fait cohabiter. Mais dans un premier temps, reparlons un peu de ce qu'est un serveur.

Nous l'avons vu dans le premier chapitre, un serveur est chargé de réceptionner du trafic, l'interpréter puis agir en conséquence. Pour cela, il possède un **port** particulier qui lui est dédié. Chaque octet arrivant sur ce port lui est donc destiné. On dit que le serveur **écoute** sur un **port** .

C'est donc à partir de cela que nous allons pouvoir mettre en place notre serveur !

Comme pour le client, il va falloir commencer par les options du shield (MAC, IP...) afin que ce dernier puisse se connecter à votre box/routeur. On retrouve donc un setup similaire au chapitre précédent :

```
// Ces deux bibliothèques sont indispensables pour le shield
#include <SPI.h>
#include <Ethernet.h>

// L'adresse MAC du shield
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xA5, 0x7E };
// L'adresse IP que prendra le shield
IPAddress ip(192,168,0,143);

void setup()
{
  // On démarre la voie série pour déboguer
  Serial.begin(9600);

  char erreur = 0;
  // On démarre le shield Ethernet SANS adresse IP (donc donnée via DHCP)
  erreur = Ethernet.begin(mac);

  if (erreur == 0) {
    Serial.println("Parametrage avec ip fixe...");
    // si une erreur a eu lieu cela signifie que l'attribution DHCP
    // ne fonctionne pas. On initialise donc en forçant une IP
    Ethernet.begin(mac, ip);
  }
  Serial.println("Init...");
  // Donne une seconde au shield pour s'initialiser
  delay(1000);
  Serial.print("Pret !");
}
```

Bien. Au chapitre précédent cependant nous avions des variables concernant le *client* . Ici, de manière similaire nous aurons donc des variables concernant le **serveur** . La première et unique nouvelle chose sera donc une variable de type `EthernetServer` qui prendra un paramètre : le port d'écoute. J'ai choisi 4200 de manière un peu aléatoire, car je sais qu'il n'est pas utilisé sur mon réseau. Une liste des ports les plus souvent utilisés peut être trouvée sur Wikipédia .

```
// Initialise notre serveur
```

```
// Ce dernier écoutera sur le port 4200
EthernetServer serveur(4200);
```

Puis, à la fin de notre setup, il faudra démarrer le serveur avec la commande suivante :

```
serveur.begin();
```

En résumé, on aura donc le code suivant pour l'initialisation :

```
// Ces deux bibliothèques sont indispensables pour le shield
#include <SPI.h>
#include <Ethernet.h>

// L'adresse MAC du shield
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xA5, 0x7E };
// L'adresse IP que prendra le shield
IPAddress ip(192,168,0,143);

// Initialise notre serveur
// Ce dernier écoutera sur le port 4200
EthernetServer serveur(4200);

void setup()
{
  // On démarre la voie série pour déboguer
  Serial.begin(9600);

  char erreur = 0;
  // On démarre le shield Ethernet SANS adresse IP (donc donnée via DHCP)
  erreur = Ethernet.begin(mac);

  if (erreur == 0) {
    Serial.println("Parametrage avec ip fixe...");
    // si une erreur a eu lieu cela signifie que l'attribution DHCP
    // ne fonctionne pas. On initialise donc en forçant une IP
    Ethernet.begin(mac, ip);
  }
  Serial.println("Init...");
  // Donne une seconde au shield pour s'initialiser
  delay(1000);
  // On lance le serveur
  serveur.begin();
  Serial.print("Pret !");
}
```

Et voilà, votre serveur Arduino est en train de surveiller ce qui se passe sur le réseau !

Répondre et servir des données

Maintenant que le serveur est prêt et attend qu'on lui parle, on va pouvoir coder la partie "communication avec le demandeur". La première étape va être de vérifier si un client attend une réponse de la part de notre serveur. On va donc retrouver notre objet EthernetClient vu au chapitre précédent et une fonction du serveur que l'on aurait presque pu deviner : `available()`

```
// Regarde si un client est connecté et attend une réponse
EthernetClient client = serveur.available();
```

Ensuite les choix sont simples. Soit un client (donc une application externe) est connecté avec l'Arduino et veut interagir, soit il n'y a personne et donc on ne fait… rien (ou autre chose). Pour cela, on va simplement regarder si client vaut autre chose que zéro. Si c'est le cas, alors on traite les données.

```
if (client) {
  // Quelqu'un est connecté !
}
```

Maintenant, on va faire au plus simple. On va considérer que l'on renvoie toujours les mêmes informations : la valeur de l'entrée analogique 0 et la variable millis(), quelle que soit la requête du client. On va alors se retrouver dans le cas similaire au chapitre précédent ou il faudra simplement construire une requête pour renvoyer des données. Comme j'aime les choses simples, j'ai décidé de ne pas renvoyer une page web (trop verbeux), mais juste du texte au format JSON qui est simple à lire et à fabriquer.

Le HTML demande **BEAUCOUP** trop de contenu texte à envoyer pour afficher une information utile. Soyons clairs, un système embarqué comme Arduino n'est **pas fait pour afficher des pages web** , il faut pouvoir renvoyer des informations de manière **simple et concise** .

Il faudra comme pour le client commencer par renvoyer un header. Le nôtre sera simple et possèdera seulement deux informations : le protocole utilisé avec le code de retour (encore http 1.1 et 200 pour dire que tout s'est bien passé) ainsi que le type mime du contenu renvoyé (en l'occurrence "application/json"). Si vous renvoyez de l'html, ce serait "text/html" par exemple.

```
// Tout d'abord le code de réponse 200 = réussite
client.println("HTTP/1.1 200 OK");
// Puis le type mime du contenu renvoyé, du json
client.println("Content-Type: application/json");
// Et c'est tout !
// On envoie une ligne vide pour signaler la fin du header
client.println();
```

Une fois le header envoyé, on construit et envoie notre réponse json. C'est assez simple, il suffit de bien former le texte en envoyant les données dans le bon ordre avec les bons marqueurs.

```
// Puis on commence notre JSON par une accolade ouvrante
client.println("{");
// On envoi la première clé : "uptime"
client.print("\t\"uptime (ms)\": ");
// Puis la valeur de l'uptime
client.print(millis());
//Une petite virgule pour séparer les deux clés
client.println(",");
// Et on envoie la seconde nommée "analog 0"
client.print("\t\"analog 0\": ");
client.println(analogRead(A0));
```

```
// Et enfin on termine notre JSON par une accolade fermante
client.println("}");
```
On a presque fini !

Une fois la réponse envoyée, on va faire une toute petite pause pour laisser le temps aux données de partir et enfin on fera le canal de communication avec le client.

```
// Donne le temps au client de prendre les données
delay(10);
// Ferme la connexion avec le client
client.stop();
```
Et voilà !

Il est maintenant temps de tester. Branchez votre Arduino, connectez-la au réseau et allez sur la page ip:port que vous avez paramétrée avec votre navigateur (en l'occurrence http://192.168.0.143:4200/ pour moi). Si tout se passe bien, aux valeurs près vous obtiendrez quelque chose de similaire à ceci :

```
{
    "uptime (ms)": 18155,
    "analog 0": 386
}
```
Génial, non ?

Voici le code complet pour faire tout cela :

```
// Ces deux bibliothèques sont indispensables pour le shield
#include <SPI.h>
#include <Ethernet.h>

// L'adresse MAC du shield
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xA5, 0x7E };
// L'adresse IP que prendra le shield
IPAddress ip(192,168,0,143);

// Initialise notre serveur
// Ce dernier écoutera sur le port 4200
EthernetServer serveur(4200);

void setup()
{
  // On démarre la voie série pour déboguer
  Serial.begin(9600);

  char erreur = 0;
  // On démarre le shield Ethernet SANS adresse IP (donc donnée via DHCP)
  erreur = Ethernet.begin(mac);

  if (erreur == 0) {
    Serial.println("Parametrage avec ip fixe...");
    // si une erreur a eu lieu cela signifie que l'attribution DHCP
    // ne fonctionne pas. On initialise donc en forçant une IP
    Ethernet.begin(mac, ip);
  }
```

```
  Serial.println("Init...");
  // Donne une seconde au shield pour s'initialiser
  delay(1000);
  // On lance le serveur
  serveur.begin();
  Serial.print("Pret !");
}

void loop()
{
  // Regarde si un client est connecté et attend une réponse
  EthernetClient client = serveur.available();
  if (client) {
    // Quelqu'un est connecté !
    Serial.print("On envoi !");
    // On fait notre en-tête
    // Tout d'abord le code de réponse 200 = réussite
    client.println("HTTP/1.1 200 OK");
    // Puis le type mime du contenu renvoyé, du json
    client.println("Content-Type: application/json");
    // Et c'est tout !
    // On envoie une ligne vide pour signaler la fin du header
    client.println();

    // Puis on commence notre JSON par une accolade ouvrante
    client.println("{");
    // On envoie la première clé : "uptime"
    client.print("\t\"uptime (ms)\": ");
    // Puis la valeur de l'uptime
    client.print(millis());
    //Une petite virgule pour séparer les deux clés
    client.println(",");
    // Et on envoie la seconde nommée "analog 0"
    client.print("\t\"analog 0\": ");
    client.println(analogRead(A0));
    // Et enfin on termine notre JSON par une accolade fermante
    client.println("}");
    // Donne le temps au client de prendre les données
    delay(10);
    // Ferme la connexion avec le client
    client.stop();
  }
}
```

S'il faut encore vous convaincre, sachez que cet exemple affiche environ 50 caractères, donc 50 octets (plus le header) à envoyer. La même chose en HTML proprement formaté aurait demandé au grand minimum le double.

Agir sur une requête plus précise

Nous savons maintenant envoyer des données vers notre shield, mais il pourrait être sympa de pouvoir en interpréter et ainsi interagir avec le shield. Voyons voir comment, mais avant tout un petit avertissement :

On ne va pas se mentir, faire cela demande un peu de bidouille car il faut être à l'aise avec la manipulation de chaîne de caractères. Ce type de code est difficilement généralisable ce qui

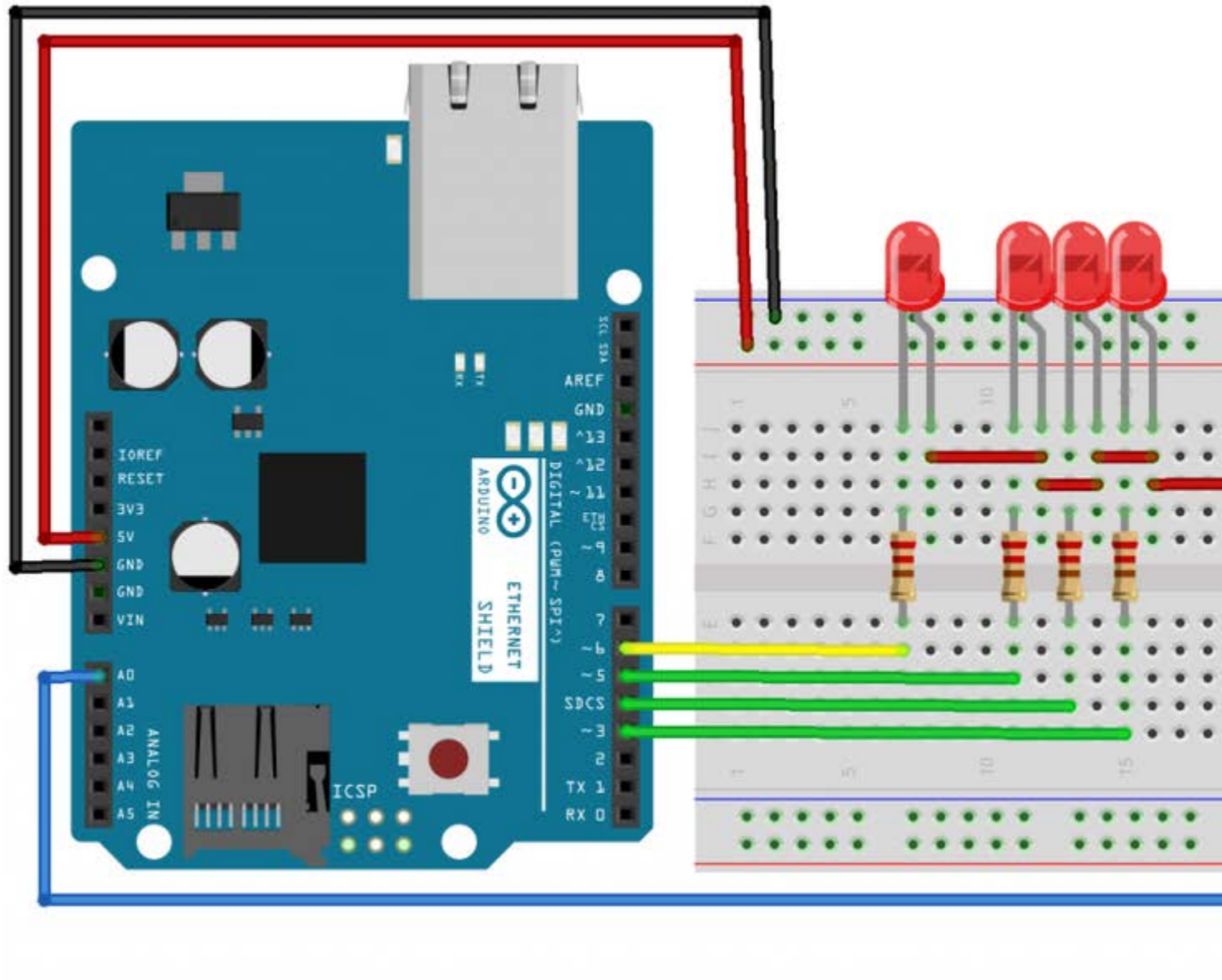
signifie qu'il faut bien souvent développer pour son cas précis. Ce qui sera exposé ici sera donc un "exemple d'application pour faire comprendre".

Notre objectif va être simple, on doit être capable de recevoir et interpréter les choses suivantes :

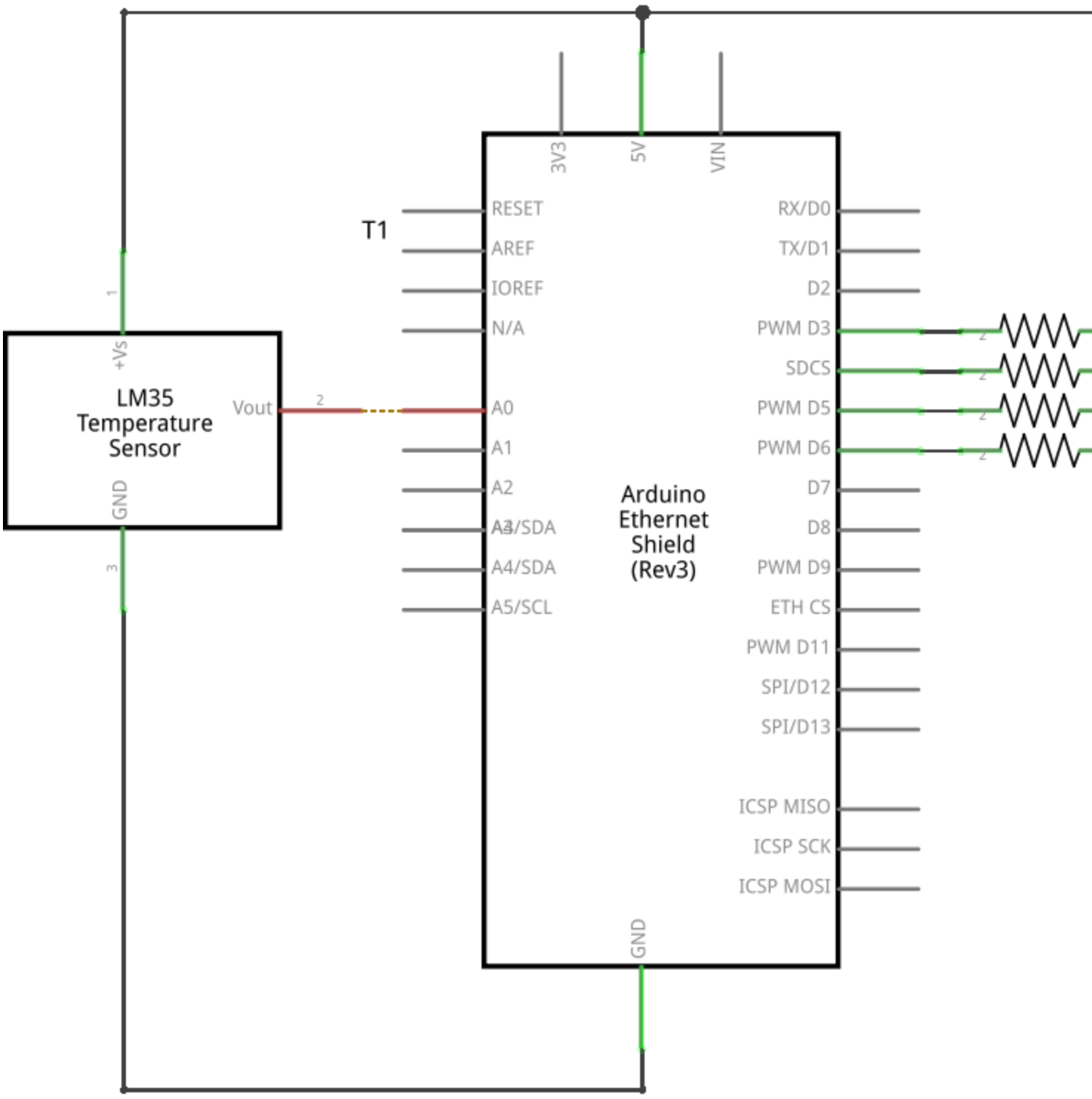
- Un premier paramètre sera "b" signifiant "broches". Il indiquera l'état des broches 3, 4, 5. Si une broche est dans la liste, alors elle est à 1, sinon 0
- Un second paramètre sera "p" et indiquera le rapport cyclique (entier entre 0 et 255) d'une pwm sur la broche 6.
- Enfin, dans tous les cas on renvoie un json possédant les informations suivantes :
    - L'uptime de l'Arduino (millis)
    - L'état des broches 3, 4 et 5
    - La valeur de la PWM de la broche 6
    - La lecture analogique de la broche A0

Démarrons par la fin et avec les choses simples, une fonction pour renvoyer le json.

En cadeau voici le schéma !



Ethernet, montage



Ethernet, schéma

Répondre à la requête

On l'a vu plus tôt, répondre à une requête n'est pas très compliqué et construire un json non plus. Je vais donc simplement vous poster le code de la fonction `repondre()` avec des commentaires en espérant que cela suffise. Rien de nouveau par rapport aux choses vues ci-dessus.

```
void repondre(EthernetClient client) {
  // La fonction prend un client en argument

  Serial.println("\nRepondre"); // debug
  // On fait notre en-tête
  // Tout d'abord le code de réponse 200 = réussite
  client.println("HTTP/1.1 200 OK");
  // Puis le type mime du contenu renvoyé, du json
  client.println("Content-Type: application/json");
  // Autorise le cross origin
  client.println("Access-Control-Allow-Origin: *");
  // Et c'est tout !
  // On envoi une ligne vide pour signaler la fin du header
  client.println();

  // Puis on commence notre JSON par une accolade ouvrante
  client.println("{");
  // On envoie la première clé : "uptime"
  client.print("\t\"uptime\": ");
  // Puis la valeur de l'uptime
  client.print(millis());
  //Une petite virgule pour séparer les deux clés
  client.println(",");
  // Et on envoie la seconde nommée "analog 0"
  client.print("\t\"A0\": ");
  client.print(analogRead(A0));
  client.println(",");
  // Puis la valeur de la PWM sur la broche 6
  client.print("\t\"pwm\": ");
  client.print(pwm, DEC);
  client.println(",");
  // Dernières valeurs, les broches (elles-mêmes dans un tableau)
  client.println("\t\"broches\": {");
  // La broche 3
  client.print("\t\t\"3\": ");
  client.print(digitalRead(3));
  client.println(",");
  // La broche 4
  client.print("\t\t\"4\": ");
  client.print(digitalRead(4));
  client.println(",");
  // La broche 5
  client.print("\t\t\"5\": ");
  client.println(digitalRead(5));
  client.println("\t}");
  // Et enfin on termine notre JSON par une accolade fermante
  client.println("}");
}
```

Lire la requête

Lorsque l'on reçoit une requête, il faut la traiter. Avez-vous essayé de faire un `Serial.print()` des caractères reçus lors des demandes ? Vous pouvez remarquer que la

première ligne est toujours "GET /... HTTP/1.1". Avec `GET` qui est "l'action", `/...` l'url demandée et `HTTP/1.1` le protocole utilisé, on a tout ce qu'il faut pour (ré)agir ! Par exemple, si on reçoit la requête `GET /?b=3,4&p=42 HTTP/1.1` , on aura "juste" à traiter la demande pour extraire le numéro des broches à allumer (3 et 4 mais pas 5) et la valeur du rapport cyclique à appliquer pour la PWM (42). Voyons comment faire.

Préparation

Tout d'abord, on va réserver un tableau de caractères pour le traitement des données. Dans notre cas, 100 caractères devraient largement faire l'affaire. On va aussi déclarer quelques variables pour stocker l'état des broches à changer.

```
char *url = (char *)malloc(100); // L'url reçue à stocker
//char url[100]; // équivalent à la ligne du dessus mais qui ne semble pas vouloir fonctionner
char index = 0; // index indiquant où l'on est rendu dans la chaine
boolean etats[3] = {LOW, LOW, LOW}; // L'état des 3 sorties
unsigned char pwm = 0; // La valeur de la pwm
```

Une fois cela fait, nous allons devoir passer à la récupération de la première chaîne envoyée par le client.

Récuperer l'URL

C'est parti, faisons notre loop ! On va commencer comme avant, en allant lire la présence d'un client.

```
void loop() {
  // Regarde si un client est connecté et attend une réponse
  EthernetClient client = serveur.available();
  if (client) {
    url = ""; // on remet à zéro notre chaîne tampon
    index = 0;
    // traitement
  }
}
```

Si un client est présent, on va regarder s'il a des données à nous donner tant qu'il est connecté (car s'il ne se déconnecte pas, pas la peine de perdre du temps avec lui !).

```
void loop() {
  // Regarde si un client est connecté et attend une réponse
  EthernetClient client = serveur.available();
  if (client) { // Un client est là ?
    url = ""; // on remet à zéro notre chaîne tampon
    index = 0;
    while(client.connected()) { // Tant que le client est connecté
      if(client.available()) { // A-t-il des choses à dire ?
        // traitement des infos du client
      }
    }
  }
}
```

Maintenant que l'on sait que le client est là et nous parle, on va l'écouter en lisant les caractères reçus.

```
void loop() {
  // Regarde si un client est connecté et attend une réponse
  EthernetClient client = serveur.available();
  if (client) { // Un client est là ?
    url = ""; // on remet à zéro notre chaîne tampon
    index = 0;
    while(client.connected()) { // Tant que le client est connecté
      if(client.available()) { // A-t-il des choses à dire ?
        // traitement des infos du client
        char carlu = client.read(); //on lit ce qu'il raconte
        if(carlu != '\n') { // On est en fin de chaîne ?
          // non ! alors on stocke le caractère
          url[index] = carlu;
          index++;
        } else {
          // on a fini de lire ce qui nous intéresse
          // on marque la fin de l'url (caractère de fin de chaîne)
          url[index] = '';
          // + TRAITEMENT
          // on quitte le while
          break;
        }
      }
    }
    // Donne le temps au client de prendre les données
    delay(10);
    // Ferme la connexion avec le client
    client.stop();
  }
}
```

Interpréter l'URL

Maintenant que nous avons la chaîne du client, il faut l'interpréter pour lire les valeurs des paramètres. Tout d'abord, on commencera par remettre les anciens paramètres à zéro. Ensuite, on va parcourir les caractères à la recherche de marqueur connu : b et p. Ce n'est pas ce qu'il y a de plus simple, mais vous allez voir avec un peu de méthode on y arrive ! Rappel : Nous cherchons à interpréter `GET /?b=3,4&p=42 HTTP/1.1`

PS : le code est "volontairement" un peu plus lourd, car je fais des tests évitant les problèmes si quelqu'un écrit une URL un peu farfelue (sans le "b" ou le "p").

```
boolean interpreter() {
  // On commence par mettre à zéro tous les états
  etats[0] = LOW;
  etats[1] = LOW;
  etats[2] = LOW;
  pwm = 0;

  // Puis maintenant on va chercher les caractères/marqueurs un par un.
  index = 0; // Index pour se promener dans la chaîne (commence à 4 pour enlever "GET "
  while(url[index-1] != 'b' && url[index] != '=') { // On commence par chercher le "b="
    index++; // Passe au caractère suivant
```

```
    if(index == 100) {
      // On est rendu trop loin !
      Serial.println("Oups, probleme dans la recherche de 'b='");
      return false;
    }
  }
  // Puis on lit jusqu'à trouver le '&' séparant les broches de pwm
  while(url[index] != '&') { // On cherche le '&'
    if(url[index] >= '3' && url[index] <= '5') {
      // On a trouvé un chiffre identifiant une broche
      char broche = url[index]-'0'; // On ramène ça au format décimal
      etats[broche-3] = HIGH; // Puis on met la broche dans un futur état haut
    }
    index++; // Passe au caractère suivant
    if(index == 100) {
      // On est rendu trop loin !
      Serial.println("Oups, probleme dans la lecture des broches");
      return false;
    }
    // NOTE : Les virgules séparatrices sont ignorées
  }
  // On a les broches, reste plus que la valeur de la PWM
  // On cherche le "p="
  while(url[index-1] != 'p' && url[index] != '=' && index<100) {
    index++; // Passe au caractère suivant
    if(index == 100) {
      // On est rendu trop loin !
      Serial.println("Oups, probleme dans la recherche de 'p='");
      return false;
    }
  }
  // Maintenant, on va fouiller jusqu'a trouver un espace
  while(url[index] != ' ') { // On cherche le ' ' final
    if(url[index] >= '0' && url[index] <= '9') {
      // On a trouve un chiffre !
      char val = url[index]-'0'; // On ramene ca au format decimal
      pwm = (pwm*10) + val; // On stocke dans la pwm
    }
    index++; // Passe au caractère suivant
    if(index == 100) {
      // On est rendu trop loin !
      Serial.println("Oups, probleme dans la lecture de la pwm");
      return false;
    }
    // NOTE : Les virgules séparatrices sont ignorées
  }
  // Rendu ici, on a trouvé toutes les informations utiles !
  return true;
}
```

Agir sur les broches

Lorsque toutes les valeurs sont reçues et interprétées, il ne reste plus qu'à les appliquer à nos broches. Vu ce que l'on vient de faire, c'est de loin le plus facile !

```
void action() {
  // On met à jour nos broches
  digitalWrite(3, etats[0]);
  digitalWrite(4, etats[1]);
  digitalWrite(5, etats[2]);
```

```
  // Et la PWM
  analogWrite(6, pwm);
}
```
On assemble !!

Il ne reste plus qu'à enchaîner toutes nos fonctions pour avoir un code complet !!

```
void loop() {
  // Regarde si un client est connecté et attend une réponse
  EthernetClient client = serveur.available();
  if (client) { // Un client est là ?
    Serial.println("Ping !");
    url = ""; // on remet à zéro notre chaîne tampon
    index = 0;
    while(client.connected()) { // Tant que le client est connecté
      if(client.available()) { // A-t-il des choses à dire ?
        // traitement des infos du client
        char carlu = client.read(); //on lit ce qu'il raconte
        if(carlu != '\n') { // On est en fin de chaîne ?
          // non ! alors on stocke le caractère
          Serial.print(carlu);
          url[index] = carlu;
          index++;
        } else {
          // on a fini de lire ce qui nous intéresse
          // on marque la fin de l'url (caractère de fin de chaîne)
          url[index] = '\0';
          boolean ok = interpreter(); // essaie d'interpréter la chaîne
          if(ok) {
            // tout s'est bien passé = on met à jour les broches
            action();
          }
          // et dans tout les cas on répond au client
          repondre(client);
          // on quitte le while
          break;
        }
      }
    }
    // Donne le temps au client de prendre les données
    delay(10);
    // Ferme la connexion avec le client
    client.stop();
    Serial.println("Pong !");
  }
}
```
Code complet

Contenu masqué, cliquez pour afficher

```
// Ces deux bibliothèques sont indispensables pour le shield
#include <SPI.h>
#include <Ethernet.h>

// L'adresse MAC du shield
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xA5, 0x7E };
// L'adresse IP que prendra le shield
IPAddress ip(192,168,0,143);
```

```
// Initialise notre serveur
// Ce dernier écoutera sur le port 4200
EthernetServer serveur(4200);

char *url = (char *)malloc(100); // L'url recu à stocker
//char url[100];
char index = 0; // index indiquant où l'on est rendu dans la chaîne
boolean etats[3] = {LOW, LOW, LOW}; // L'état des 3 sorties
unsigned char pwm = 0; // La valeur de la pwm

void setup()
{
  // On démarre la voie série pour déboguer
  Serial.begin(9600);

  // Configure et initialise les broches
  pinMode(3, OUTPUT); digitalWrite(3, LOW);
  pinMode(4, OUTPUT); digitalWrite(4, LOW);
  pinMode(5, OUTPUT); digitalWrite(5, LOW);
  pinMode(6, OUTPUT); analogWrite(6, 0);

  char erreur = 0;
  // On démarre le shield Ethernet SANS adresse ip (donc donnée via DHCP)
  erreur = Ethernet.begin(mac);

  if (erreur == 0) {
    Serial.println("Parametrage avec ip fixe...");
    // si une erreur a eu lieu cela signifie que l'attribution DHCP
    // ne fonctionne pas. On initialise donc en forçant une IP
    Ethernet.begin(mac, ip);
  }
  Serial.println("Init...");
  // Donne une seconde au shield pour s'initialiser
  delay(1000);
  // On lance le serveur
  serveur.begin();
  Serial.println("Pret !");
}

void loop() {
  // Regarde si un client est connecté et attend une réponse
  EthernetClient client = serveur.available();
  if (client) { // Un client est là ?
    Serial.println("Ping !");
    url = ""; // on remet à zéro notre chaîne tampon
    index = 0;
    while(client.connected()) { // Tant que le client est connecté
      if(client.available()) { // A-t-il des choses à dire ?
        // traitement des infos du client
        char carlu = client.read(); //on lit ce qu'il raconte
        if(carlu != '\n') { // On est en fin de chaîne ?
          // non ! alors on stocke le caractère
          Serial.print(carlu);
          url[index] = carlu;
          index++;
        } else {
          // on a fini de lire ce qui nous intéresse
          // on marque la fin de l'url (caractère de fin de chaîne)
          url[index] = '\0';
          boolean ok = interpreter(); // essaie d'interpréter la chaîne
          if(ok) {
```

```
            // tout s'est bien passé = on met à jour les broches
            action();
          }
          // et dans tout les cas on répond au client
          repondre(client);
          // on quitte le while
          break;
        }
      }
    }
    // Donne le temps au client de prendre les données
    delay(10);
    // Ferme la connexion avec le client
    client.stop();
    Serial.println("Pong !");
  }
}

void rafraichir() {
  // Rafraichit l'etat des broches / PWM
  digitalWrite(3, etats[0]);
  digitalWrite(4, etats[1]);
  digitalWrite(5, etats[2]);
  analogWrite(6, pwm);
}

void repondre(EthernetClient client) {
  // La fonction prend un client en argument

  Serial.println("\nRepondre"); // debug
  // On fait notre en-tête
  // Tout d'abord le code de réponse 200 = réussite
  client.println("HTTP/1.1 200 OK");
  // Puis le type mime du contenu renvoyé, du json
  client.println("Content-Type: application/json");
  // Autorise le cross origin
  client.println("Access-Control-Allow-Origin: *");
  // Et c'est tout !
  // On envoi une ligne vide pour signaler la fin du header
  client.println();

  // Puis on commence notre JSON par une accolade ouvrante
  client.println("{");
  // On envoie la première clé : "uptime"
  client.print("\t\"uptime\": ");
  // Puis la valeur de l'uptime
  client.print(millis());
  //Une petite virgule pour séparer les deux clés
  client.println(",");
  // Et on envoie la seconde nommée "analog 0"
  client.print("\t\"A0\": ");
  client.print(analogRead(A0));
  client.println(",");
  // Puis la valeur de la PWM sur la broche 6
  client.print("\t\"pwm\": ");
  client.print(pwm, DEC);
  client.println(",");
  // Dernières valeurs, les broches (elles mêmes dans un tableau)
  client.println("\t\"broches\": {");
  // La broche 3
  client.print("\t\t\"3\": ");
```

```
  client.print(digitalRead(3));
  client.println(",");
  // La broche 4
  client.print("\t\t\"4\": ");
  client.print(digitalRead(4));
  client.println(",");
  // La broche 5
  client.print("\t\t\"5\": ");
  client.println(digitalRead(5));
  client.println("\t}");
  // Et enfin on termine notre JSON par une accolade fermante
  client.println("}");
}

boolean interpreter() {
  // On commence par mettre à zéro tous les états
  etats[0] = LOW;
  etats[1] = LOW;
  etats[2] = LOW;
  pwm = 0;

  // Puis maintenant on va chercher les caractères/marqueurs un par un.
  index = 0; // Index pour se promener dans la chaîne (commence à 4 pour enlever "GET "
  while(url[index-1] != 'b' && url[index] != '=') { // On commence par chercher le "b="
    index++; // Passe au caractère suivant
    if(index == 100) {
      // On est rendu trop loin !
      Serial.println("Oups, probleme dans la recherche de 'b='");
      return false;
    }
  }
  // Puis on lit jusqu’à trouver le '&' séparant les broches de pwm
  while(url[index] != '&') { // On cherche le '&'
    if(url[index] >= '3' && url[index] <= '5') {
      // On a trouvé un chiffre identifiant une broche
      char broche = url[index]-'0'; // On ramène ça au format décimal
      etats[broche-3] = HIGH; // Puis on met la broche dans un futur état haut
    }
    index++; // Passe au caractère suivant
    if(index == 100) {
      // On est rendu trop loin !
      Serial.println("Oups, probleme dans la lecture des broches");
      return false;
    }
    // NOTE : Les virgules séparatrices sont ignorées
  }
  // On a les broches, reste plus que la valeur de la PWM
  // On cherche le "p="
  while(url[index-1] != 'p' && url[index] != '=' && index<100) {
    index++; // Passe au caractère suivant
    if(index == 100) {
      // On est rendu trop loin !
      Serial.println("Oups, probleme dans la recherche de 'p='");
      return false;
    }
  }
  // Maintenant, on va fouiller jusqu'a trouver un espace
  while(url[index] != ' ') { // On cherche le ' ' final
    if(url[index] >= '0' && url[index] <= '9') {
      // On a trouve un chiffre !
      char val = url[index]-'0'; // On ramene ca au format decimal
```

```
      pwm = (pwm*10) + val; // On stocke dans la pwm
    }
    index++; // Passe au caractère suivant
    if(index == 100) {
      // On est rendu trop loin !
      Serial.println("Oups, probleme dans la lecture de la pwm");
      return false;
    }
    // NOTE : Les virgules séparatrices sont ignorées
  }
  // Rendu ici, on a trouvé toutes les informations utiles !
  return true;
}

void action() {
  // On met à jour nos broches
  digitalWrite(3, etats[0]);
  digitalWrite(4, etats[1]);
  digitalWrite(5, etats[2]);
  // Et la PWM
  analogWrite(6, pwm);
}
```

Sortir de son réseau privé

À ce stade, nous arrivons à récupérer des informations et donner des ordres à notre Arduino. Cependant, on est bloqué dans notre réseau local. En effet, si vous essayez d'y accéder depuis un autre ordinateur à l'autre bout du monde, il est fort probable que cela ne marche pas…

Pour pallier cela, il va falloir faire un peu d'administration réseau. Je vais couvrir la démarche ici mais ne rentrerai pas trop dans les détails, car ce n'est pas non plus le but de ce tutoriel. Je partirai aussi du principe que vous êtes à votre domicile et utilisez une box ou un routeur que vous pouvez administrer.

L'opération que nous allons faire ici s'appelle une redirection NAT (Network address translation). Mais tout d'abord, essayons de comprendre le problème.

Le souci

Le problème dans l'état actuel c'est que votre box laisse passer le trafic vers l'extérieur, accepte les réponses, mais ne tolère pas trop qu'on accède directement à son adresse sur n'importe quel port. En effet, après tout c'est logique. Imaginez que vous avez plusieurs équipements connectés a votre routeur/box. Si vous essayez d'y accéder depuis l'extérieur, comment cette dernière saura à quel matériel vous souhaitez accéder ?

Dans votre réseau local, chaque appareil à sa propre adresse IP qui le représente **localement** . Cette adresse est très souvent de la forme 192.168.0.xyz. Vous pourriez avoir par exemple votre téléphone en 192.168.0.142, votre ordinateur en 192.168.0.158 et votre Arduino en

192.168.0.199. Votre box (qui gère ce réseau local) possède quant à elle une IP **publique** . C'est cette IP qui la représente aux yeux du monde. Admettons, pour l'exemple, que cette adresse soit 42.128.12.13 (trouvez la vôtre avec un service comme my-ip.com). Si vous cherchez à accéder à l'adresse publique avec le port 4200 en espérant atteindre votre Arduino vous serez déçus. En effet, vous allez bien atteindre votre box, mais cette dernière n'aura aucune idée de quel équipement doit être interrogé ! Est-ce votre ordinateur ? Votre smartphone ? Votre Arduino ?

Il va donc falloir lui expliquer…

La solution

Chaque constructeur de box/routeur possède sa propre interface. Je vais essayer d'être le plus générique possible pour que les explications parlent au plus grand nombre, mais ne m'en voulez pas si toutes les dénominations ne sont pas exactement comme chez vous !

Et cette explication s'appelle une redirection **NAT** , ou redirection de port. En faisant cela, vous expliquerez à votre box que "tout le trafic qui arrive sur ce port particulier doit être envoyé sur cet équipement local" (et vous pouvez même rerouter le trafic pour le changer de port si vous voulez).

Mettons cela en place. Pour cela, commencez par aller dans l'interface d'administration de votre box (souvent c'est à l'adresse 192.168.0.1). Vous devez être dans le réseau local de la box pour le faire ! Ensuite, il vous faudra trouver la partie parlant de "NAT" ou de "redirection de port". Une fois dans cette dernière, il va falloir demander à ce que tout ce qui rentre dans le port 4200 (ou la plage 4200–4200) soit redirigé vers l'équipement "Arduino" (Wiznet) ou son adresse IP locale si vous la connaissez et que vous l'avez déjà imposée au routeur comme fixe.

Vous aurez alors quelque chose comme ça :

Réglages NAT

Sauvegardez et éventuellement redémarrez votre box (normalement ce n'est pas nécessaire, mais sur du vieux matériel ça peut arriver…). Maintenant, démarrez votre Arduino avec un des programmes ci-dessus et essayez d'accéder à votre adresse publique et le port 4200

avec un navigateur internet. Normalement, l'Arduino devrait répondre comme si vous l'interrogiez avec son adresse locale !

Faire une interface pour dialoguer avec son Arduino

Pour terminer ce tutoriel, je vous propose de réaliser une petite interface de pilotage de l'Arduino via internet. Pour cela, on va garder le programme que nous avions dans l'Arduino pour le paragraphe concernant les requêtes avancées, et nous nous contenterons de simplement faire de l'HTML et du JavaScript. Le HTML servira à faire l'interface et le JavaScript fera les interactions via des requêtes ajax.

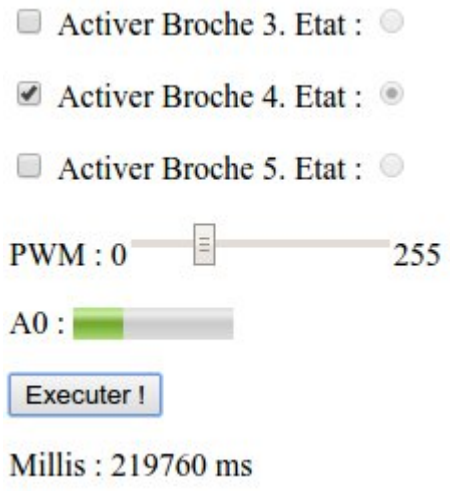
Avant toute chose, je vous ai menti ! Il faut en fait rajouter une petite ligne dans notre code de la fonction `repondre()` faite plus tôt. En effet, pour des raisons de sécurité les requêtes ajax ne peuvent pas aller d'un domaine à un autre (donc de "n'importe où sur le web" à "votre Arduino"). Il faut donc rajouter une ligne dans le header renvoyé pour signaler que l'on autorise le "cross-domain". Rajoutez donc cette ligne juste après le "content-type" :

```
// Autorise le cross origin
client.println("Access-Control-Allow-Origin: *");
```

Maintenant que cela est fait, nous allons créer une structure HTML toute simple pour avoir nos boutons.

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Interface de pilotage Arduino</title>
  </head>
  <body>
    <div class="main">
      <p>
        Adresse publique du shield : <br />
        http://<input type="text" id="ip" value="123.123.123.123" size="15"/>
        : <input type="text" id="port" value="4200" size="5"/>
      </p>
      <hr />
      <p>
        <input type="checkbox" id="broche3" name="broche3" />
        <label for="broche3">Activer Broche 3.</label>
        Etat : <input type="radio" id="etat3" disabled />
      </p>
      <p>
        <input type="checkbox" id="broche4" name="broche4" />
        <label for="broche4">Activer Broche 4.</label>
        Etat : <input type="radio" id="etat4" disabled />
      </p>
      <p>
        <input type="checkbox" id="broche5" name="broche5" />
        <label for="broche5">Activer Broche 5.</label>
        Etat : <input type="radio" id="etat5" disabled />
      </p>
      <p>
```

```
        PWM : 0<input type="range" min="0" max="255" id="pwm" />255
      </p>
      <p>
        A0 : <meter min="0" max="1023" id="a0" />
      </p>
      <button id="envoyer">Executer !</button>
      <p>
        Millis : <span id="millis">0</span> ms
      </p>
    </div>
  </body>
</html>
```



Interface HTML

Ensuite, un peu de JavaScript nous permettra les interactions. L'ensemble est grosso modo divisé en deux fonctions importantes. `setup()` qui sera exécuté lorsque la page est prête puis `executer()` qui sera appelée à chaque fois que vous cliquez sur le bouton. Cette dernière fera alors une requête à votre Arduino et attendra la réponse json. La fonction `afficher()` utilisera alors les informations pour changer les composants html.

Comme vous pouvez le constater, toute la partie affichage est gérée de manière quasi complètement indépendante de l'Arduino. Cela va nous permettre de transmettre un minimum de données et garder une souplesse maximale sur l'affichage. Si demain vous décidez de changer l'interface voire carrément de faire une application dans un autre langage, vous n'aurez pas besoin de toucher à votre Arduino car les données sont envoyées dans un format générique.

```
var broches = []; // Tableau de broches
var etats = []; // Tableau d'etat des broches
var pwm;
var a0;
var millis;
var adresse = "http://82.143.160.118:4200/"; // L'url+port de votre shield

document.addEventListener('DOMContentLoaded', setup, false);

function setup() {
    // fonction qui va lier les variables à leur conteneur HTML
```

```
    broches[3] = document.getElementById("broche3");
    broches[4] = document.getElementById("broche4");
    broches[5] = document.getElementById("broche5");
    etats[3] = document.getElementById("etat3");
    etats[4] = document.getElementById("etat4");
    etats[5] = document.getElementById("etat5");
    pwm = document.getElementById("pwm");
    a0 = document.getElementById("a0");
    millis = document.getElementById("millis");

    // La fonction concernant le bouton
    var bouton = document.getElementById("envoyer");
    bouton.addEventListener('click', executer, false);
}

function executer() {
    // Fonction qui va créer l'url avec les paramètres puis
    // envoyer la requête
    var requete = new XMLHttpRequest(); // créer un objet de requête
    var url = adresse;
    url += "?b=";
    for(i=3; i <= 5; i++) { // Pour les broches 3 à 5 de notre tableau
        if(broches[i].checked) // si la case est cochée
            url += i + ",";
    }
    // enlève la dernière virgule si elle existe
    if(url[url.length-1] === ',')
        url = url.substring(0, url.length-1);
    // Puis on ajoute la pwm
    url += "&p=" + pwm.value;
    console.log(url) // Pour debugguer l'url formée
    requete.open("GET", url, true); // On construit la requête
    requete.send(null); // On envoie !
    requete.onreadystatechange = function() { // on attend le retour
        if (requete.readyState == 4) { // Revenu !
            if (requete.status == 200) {// Retour s'est bien passé !
                // fonction d'affichage (ci-dessous)
                afficher(requete.responseText);
            } else { // Retour s'est mal passé :(
                alert(requete.status, requete.statusText);
            }
        }
    };
}

function afficher(json) {
    // Affiche l'état des broches/pwm/millis revenu en json
    donnees = JSON.parse(json);
    console.log(donnees);

    for(i=3; i <= 5; i++) { // Pour les broches 3 à 5 de notre tableau
        etats[i].checked = donnees["broches"][i];
    }
    pwm.value = parseInt(donnees["pwm"]);
    a0.value = parseInt(donnees["A0"]);
    millis.textContent = donnees["uptime"];
}
```

En cadeau de fin, une version utilisable en ligne de cette interface :

http://jsfiddle.net/f6c2kc11/7/

Pour l'utiliser, il suffit simplement de modifier l'url de base avec votre IP publique et le port utilisé par l'Arduino.

Magie de l'internet, votre Arduino fait maintenant partie de la grande sphère de l'IoT, le phénomène très à la mode de l'Internet of Things. Qu'allez-vous bien pouvoir envoyer comme informations dorénavant ?

---

tuto-arduino-802-arduino-et-ethernet-client