

Créer un serveur HTML+ Javascript + Ajax avec Arduino et contrôler Arduino depuis le navigateur client en utilisant des requêtes Ajax.



Ateliers Arduino

par X. HINAULT
www.mon-club-elec.fr



Tous droits réservés – 2012.

Ce document légèrement payant est soumis au droit d'auteur et est réservé à l'usage personnel.

Afin d'encourager la production de supports didactiques de qualité, ce document est légèrement payant.

La licence d'utilisation est attribuée pour un usage personnel uniquement, dans le cercle familial. Mise en ligne et diffusion non autorisées.

Si vous n'êtes pas le détenteur de la licence attribuée pour l'usage de ce document, soyez sympa, merci d'acheter votre exemplaire personnel ici : <https://monclubelec.dpdcart.com/>

Pour tout problème lié à l'utilisation de ce document, veuillez envoyer une copie ici : support@mon-club-elec.fr

Pour obtenir tout autres types de licence d'utilisation (enseignement, commercial, etc...), veuillez contacter l'auteur ici : support@mon-club-elec.fr

Vous avez constaté une erreur ? une coquille ? N'hésitez pas à nous le signaler à cette adresse : support@mon-club-elec.fr

Truc d'utilisation : visualiser ce document en mode diaporama dans le visionneur PDF. Navigation avec les flèches HAUT / BAS ou la souris.

En mode fenêtre, activer le panneau latéral vous facilitera la navigation dans le document. Bonne lecture !

Lancer également le logiciel Arduino et connecter votre carte Arduino afin de pouvoir tester au fur et à mesure les codes d'exemples !

1. Intro

L'objectif ici est :

- d'apprendre à associer un code Javascript à un élément HTML lors de la capture d'un événement provoqué par utilisateur
- d'apprendre à utiliser la technologie AJAX pour déclencher l'envoi de chaînes vers le serveur à partir du navigateur client
- d'apprendre à contrôler des broches Arduino à partir navigateur client
- d'apprendre à contrôler des dispositifs en envoyant des chaînes avec paramètres numériques à partir du navigateur client

... afin d'être en mesure de créer un serveur Arduino AJAX permettant le contrôle d'Arduino à distance par le réseau à partir du navigateur client en utilisant des requêtes Ajax.

Cet atelier fait suite à des atelier précédents consacrés à l'utilisation du javascript et d'Ajax pour réaliser des affichages que je conseille de travailler au préalable.
Nous utiliserons le même réseau.



Prêt ? C'est parti !

Pratique :

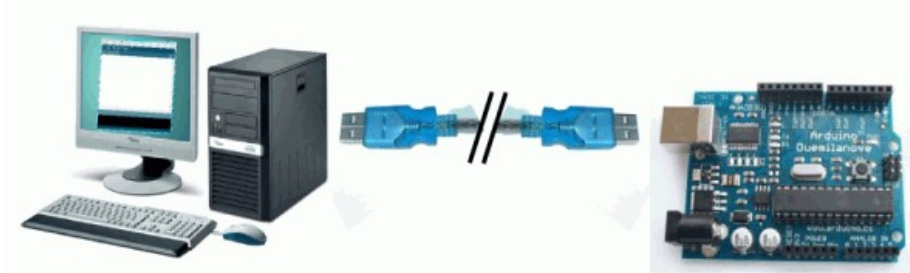
Les codes de cet atelier sont disponibles ici :

http://www.mon-club-elec.fr/mes_downloads/tutos_arduino/12b9.atelier_arduino_ethernet_tcp_javascript_ajax_controler_arduino.tar.gz

2. Matériel nécessaire pour les ateliers Arduino

Pour cet atelier, vous aurez besoin de tout ou partie des éléments suivants pour pouvoir réaliser les exemples proposés :

De l'espace de développement Arduino

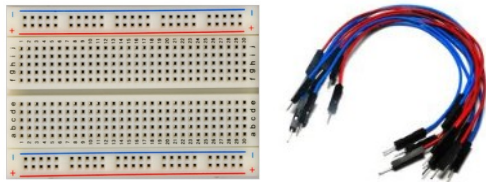


L'espace de développement Arduino associe :

- un ordinateur sous Windows, Mac Os X ou Gnu/Linux (Ubuntu)
- avec le logiciel Arduino installé (voir : <http://www.arduino.cc/>)
- un câble USB
- une carte Arduino UNO ou équivalente.

disponible chez : <http://shop.snootlab.com/> ou <http://www.gotronic.fr/>

Du nécessaire pour réaliser des montages sans soudure

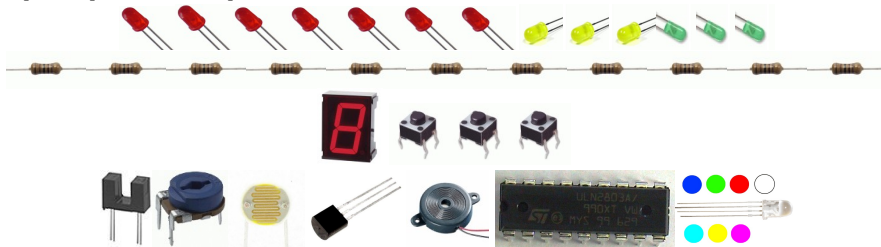


Pour réaliser des montages sans soudure, vous aurez besoin :

- d'une plaque d'essai ou breadboard moyenne (450 points)
- de quelques câbles souples (ou jumpers) mâle/mâle

disponible chez : <http://www.gotronic.fr/>

De quelques composants de base



Pour vous simplifier la vie, nous avons négocié ce kit pour vous !

Vous pouvez commander ce kit complet directement en 1 clic chez notre partenaire

<http://www.gotronic.fr/> avec le code express **701710**

GO TRONIC
ROBOTIQUE ET COMPOSANTS ÉLECTRONIQUES

Pour plus de détails, voir : http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS

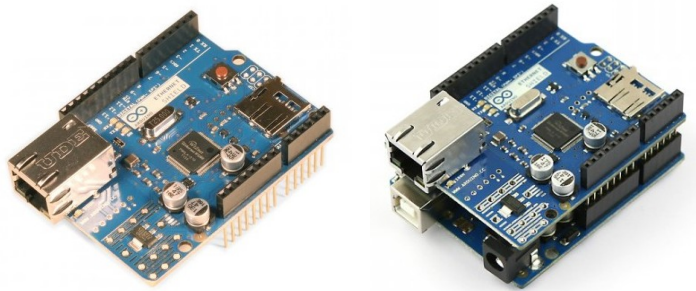
Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- des LEDs 5mm Rouges(x20), Vertes (x5) et 3 Jaunes (x5)
- digit à cathode commune rouge 13mm (x1)
- Résistances (1/4w - 5%) de 270 Ohms (x20), 4,7K Ohms (x1), 1K Ohms (x1)
- mini bouton-poussoir (x3)
- Opto-fourche (x 1)
- Résistance variable linéaire 10K (x 1)
- Photo-résistance 7mm (x 1)
- Capteur de température LM35DZ (-55/+150°C - 10mV/°C) (x 1)
- Capsule son piézoélectrique (x 1)
- ULN 2803A (CI amplificateur 8 voies, 500mA/ voie) (x 1)
- LED 5mm multicolore RVB cathode commune (x 1)

3. Matériel spécifique nécessaire pour cet atelier

Pour cet atelier vous aurez besoin également :

D'une carte d'extension (shield) Ethernet



La carte d'extension (ou shield) ethernet Arduino est une carte électronique enfichable broche à broche sur la carte Arduino et qui permet d'utiliser Arduino sur un réseau ethernet local voire même sur internet.

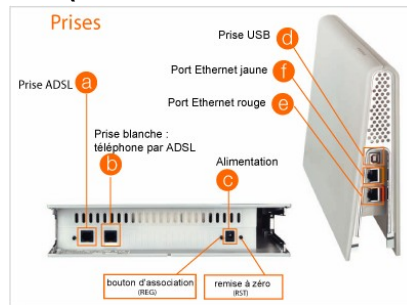
Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 +/- 4) pour communiquer avec Arduino.

Ce shield intègre également un emplacement pour **carte mémoire SD** pour des stockage de données ou de pages HTML locales.

Ne pas confondre ce shield avec la carte UNO Ethernet qui est une variante d'une carte UNO avec ethernet intégré.

disponible chez : <http://snootlab.com> | 33€ environ

D'un routeur Ethernet (ou d'une « box » internet)



Le routeur est un élément central du réseau qui permet de réaliser simplement un réseau local avec plusieurs postes. Ce routeur devra être de type Ethernet (réseau par fil) : si votre routeur supporte aussi le wifi, tant mieux, mais ça ne vous servira à rien ici. Votre routeur devra disposer de la fonction d'attribution automatique des adresses (ou DHCP), ce qui est le cas dans la grande majorité des cas.

A noter qu'une box internet est un routeur Ethernet (associé à un modem ADSL) et pourra ici être utilisée.

Ce routeur devra disposer d'au moins une prise réseau libre RJ45.

+/- d'un switch Ethernet (si le routeur n'a pas au moins 2 prises Ethernet libres)



Si votre routeur ne dispose que d'une seule prise RJ45, il faudra probablement que vous utilisiez également un switch réseau qui est une sorte de « multiprises » RJ45.

Bien qu'il ne soit pas toujours indispensable, je vous conseille fortement de disposer d'un switch car ce n'est pas cher (on en trouve à 10€) et ça vous permettra d'ajouter facilement des postes sur votre réseau.

4. Matériel spécifique nécessaire pour cet atelier (suite)

D'un ou 2 câbles réseau RJ45



Pour connecter les éléments du réseau Ethernet entre eux, vous devrez disposer d'au moins 2 câbles réseaux RJ45 (modèle classique, pas « croisé ») :

- 1 pour connecter votre PC au routeur
- 1 pour connecter le shield Ethernet au routeur

A moins que vous ayez l'intention de mettre votre carte Arduino loin de votre poste fixe, vous pouvez utiliser des câbles courts de 1m par exemple.

Noter qu'il existe des câbles RJ45 de petite longueur sur petit enrouleur : pratiques pour réduire l'encombrement !

Conseil d'ami : ne pas hésiter à avoir quelques câbles ethernet d'avance sous le coude...

+/- de 2 blocs CPL (seulement si vous souhaitez déployer le réseau Ethernet via le réseau électrique 220V)



Les blocs CPL (technologie à courant porteur) permettent assez facilement de déployer un réseau Ethernet sur le circuit 220V domestique, avec une portée de 200m sans difficulté.

Vous aurez besoin de cet équipement si vous souhaitez créer un réseau entre Arduino + shield Ethernet et votre poste fixe dans des pièces différentes par exemple.

Cet équipement un peu plus coûteux (compter 40€ pour un bloc de qualité) n'est pas indispensable dans une première approche. Mais sachez que ça existe.

A titre indicatif : j'utilise et je conseille les blocs Delovo AvPlus 200, qui disposent d'une prise terre en façade, sont faciles à utiliser, sont robustes au quotidien et sont livrés avec un utilitaire Linux pour la configuration.

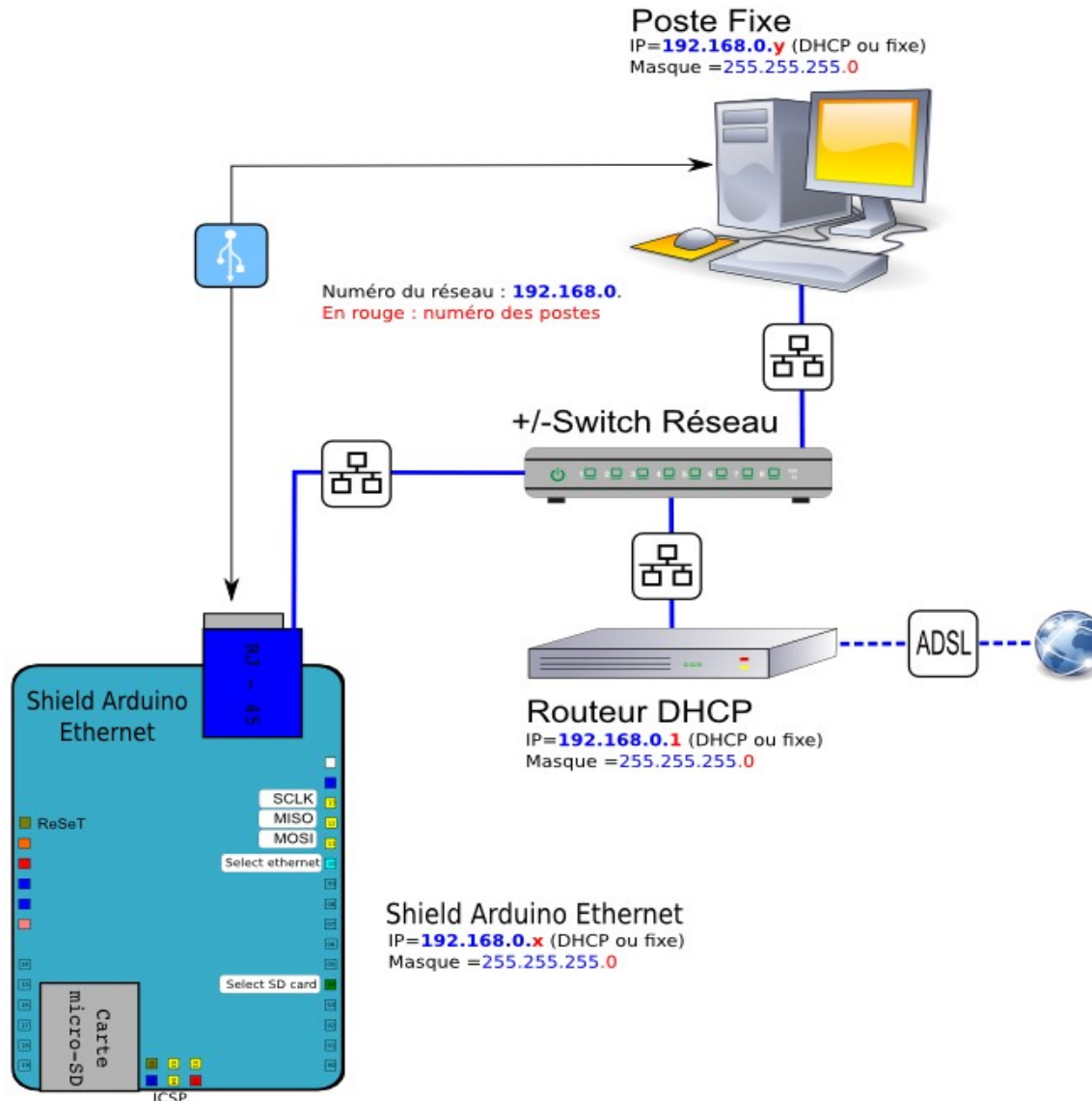
Et d'un poste fixe (PC, Mac, Netbook,...) disposant d'une carte Ethernet !



Je pense que c'est évident, mais je préfère quand même le dire... Vous avez besoin d'un poste fixe disposant d'une carte réseau Ethernet. Celui où vous lisez cette page et avec lequel vous programmez votre carte Arduino devrait faire l'affaire.

Votre poste peut-être sous Windows, Mac OS X ou Gnu/Linux, peu importe. Vous pouvez utiliser indifféremment un PC de bureau, un netbook ou un portable.

5. La structure du réseau que nous allons réaliser



Notre réseau utilisant Arduino va être constitué au minimum :

- d'un **routeur ethernet** (ou d'une box) fonctionnant en mode DHCP (=attribution automatique des adresses) avec au moins 1 prise ethernet RJ45 libre
- +/- d'un **switch réseau** (=«multiprise » réseau) si le routeur ne dispose que d'une prise ethernet RJ45
- d'un **poste fixe**, le pc sur lequel vous travaillez, connecté au routeur directement au routeur ou sur le switch avec un câble ethernet RJ45
- d'un **couple « carte Arduino + shield Ethernet »** connecté également directement au routeur ou sur le switch avec un câble ethernet RJ45

Dans un premier temps, le routeur n'a pas besoin d'être connecté à Internet.

Si il y a plus d'éléments sur votre réseau, cela n'a aucune importance, mais dans un premier temps, mieux vaut faire simple.

Remarquer que le couple « Arduino/shield Ethernet) est connecté au PC fixe de 2 façons :

- par USB d'une part
- et par ethernet d'autre part

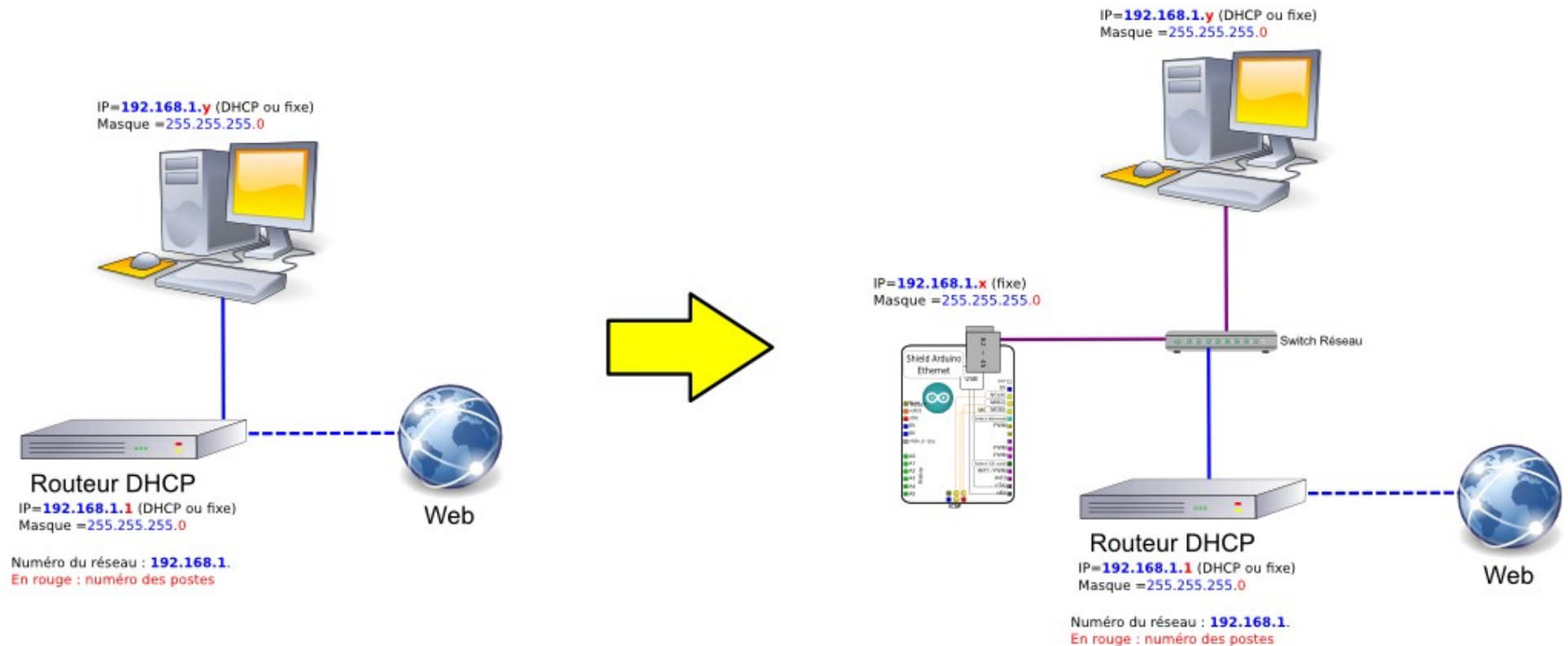
Ceci est très pratique en phase de test et de mise au point, mais une fois la programmation terminée, on pourra bien sûr déconnecter le câble USB.

La signification des numéros (adresses IP) indiqués sur ce schéma seront expliqués par la suite.

6. Monter le réseau utilisant le shield Ethernet Arduino sur un réseau avec « box » existant

Remarque :

Si votre poste fixe est déjà connecté à votre box internet, la manip' à réaliser est simple : il suffit de débrancher le câble ethernet de votre poste fixe et de le brancher sur le switch réseau. Ensuite, connecter un câble entre le switch réseau et votre PC. Puis un second câble entre le switch réseau et le shield Ethernet enfiché sur la carte Arduino. C'est tout.



7. Rappel : Syntaxe de base du langage Javascript

Intro

- Il n'est pas question, ni possible, ici, de présenter toutes les subtilités du Javascript : je vous présente simplement les bases qui vont vous permettre de démarrer à partir de ce que vous connaissez déjà du langage Arduino.

Structure

- Le Javascript utilise la même syntaxe générale que le C et donc qu'Arduino :
 - // : commentaire 1 ligne
 - /* */ : commentaire multiligne
 - ; en fin de ligne
 - { et } de limitation des sections de code des fonctions, boucles,...

Variables

- Toutes les variables, quelque soit leur type, sont déclarées avec le mot-clé **var** selon :

```
x = 0; // Une variable globale
var y = 'Hello!'; // Une autre variable globale
```

Tableaux

- Noter la possibilité de déclarer un tableau à la façon Arduino ou alors avec le mot clé **new** :

```
monTableau = [0,1,,4,5]; // crée un tableau de longueur 6 avec 4 éléments
monTableau = new Array(0,1,2,3,4,5); // crée un tableau de longueur 6 avec 6 éléments
monTableau = new Array(365); // crée un tableau vide de longueur 365
```

Condition if

- Identique à Arduino :

```
if (expression1)
{
    //instructions réalisées si expression1 est vraie;
}
else if (expression2)
{
    //instructions réalisées si expression1 est fausse et expression2 est vraie;
}
else
{
    //instructions réalisées dans les autres cas;
}
```

Boucle For

- Idem Arduino :

```
for (var i = 0; i < 5; i++) {
    alert('Itération n°' + i);
}
```

Boucle While

- Idem Arduino :

```
while (number < 10) {
    number++;
}
```

Fonctions

- La déclaration d'une fonction se fait avec le mot clé **function** :

```
function nom_fonction(argument1, argument2, argument3) {
    instructions;

    return expression;
}
```

Déclarer un objet

- Une différence d'avec Arduino : pour déclarer un objet, on utilise le mot clé **new** selon :

```
var premierObjet = new Object();
```

Fonctions de l'objet window

- Au sein de la page web, certaines fonctions implicites attachées à l'objet window (classe DOM) sont directement accessibles :

```
alert("Hello world"); // affiche message
window.alert("Hello world"); // équivalent – affiche message
```

Pour aller plus loin

- La première chose à dire, c'est qu'à priori, **vous n'avez pas besoin d'en savoir beaucoup plus pour faire ce que je vais vous proposer ici** : vous apprendrez au fur et à mesure au besoin.
- Voici cependant quelques ressources utiles :
 - http://fr.wikipedia.org/wiki/Syntaxe_JavaScript
 - <http://www.siteduzero.com/informatique/tutoriels/dynamisez-vos-sites-web-avec-javascript>

8. Rappel : Ecrire un script Javascript intégré dans une page HTML

De quoi avez-vous besoin ?

- De façon comparable à ce dont vous aviez besoin pour écrire une page HTML, pour écrire et exécuter vos premiers codes en script, vous allez avoir besoin :
 - d'un **éditeur de texte** à coloration syntaxique, ma préférence va à l'éditeur libre Bluefish
 - d'un **navigateur Web**, ma préférence allant à Firefox
- Une fois que vous avez tout ça, vous êtes parés pour passer à l'action et écrire votre premier code Javascript.

Pour info : différentes façon d'utiliser Javascript

- En pratique, on peut utiliser Javascript de plusieurs façon :
 - soit en insérant le code directement dans la page HTML : c'est ce que nous allons faire ici, car c'est le plus simple !
 - soit en mettant le code javascript dans un fichier séparé et en l'appelant dans la page HTML : intéressant pour des codes longs... mais nécessite un serveur pour le fichier.
 - soit en l'appelant lors d'un événement : nous ne l'utiliserons pas ici.

Balise HTML d'insertion d'un code javascript

- Logiquement, il existe une balise HTML pour insérer du code javascript au sein d'une page HTML. La balise est la suivante :

```
<script language="javascript" type="text/javascript">
<!--

    // code Javascript ici, avec sa syntaxe spécifique...

-->
</script>
```

- Dans le cas où l'on appelle un fichier externe, on fera :

```
<script src="url/fichierjavascript.js"></script>
```

Head ou Body ?

On placera typiquement le code Javascript dans le Head. Un point essentiel : une fonction javascript devra avoir été insérée AVANT d'être appelée. Le/les scripts seront exécutés ou pris en compte dans leur ordre d'apparition dans la page, de haut en bas.

Fonction onload()

- Pour éviter tout problème lié à l'insertion du code javascript au sein du code HTML, il est préférable de placer le code à exécuter au sein de la fonction onload() de l'objet window : de cette façon, on est sûr que le code Javascript sera chargé au lancement de la page web.

```
<script language="javascript" type="text/javascript">
<!--

    window.onload = function () { // au chargement de la page

        // code Javascript ici, avec sa syntaxe spécifique...

    } // fin onload

-->
</script>
```

Votre première page HTML + Javascript

- Il ne reste plus qu'à intégrer ça au sein d'une page HTML :

```
<!DOCTYPE HTML>

<!-- Debut de la page HTML -->
<html>
    <!-- Debut entete -->
    <head>
        <meta charset="utf-8" /> <!-- Encodage de la page -->
        <title>JavaScript: Test Canva </title> <!-- Titre de la page -->
        <!-- Début du code Javascript -->
        <script language="javascript" type="text/javascript">
            <!--
                window.onload = function () { // au chargement de la page
                    // code Javascript ici, avec sa syntaxe spécifique...
                    alert('hello world!');
                } // fin onload
            -->
        </script>
        <!-- Fin du code Javascript -->

    </head>
    <!-- Fin entete -->

    <!-- Debut Corps de page HTML -->
    <body >
        Ma belle page Web !

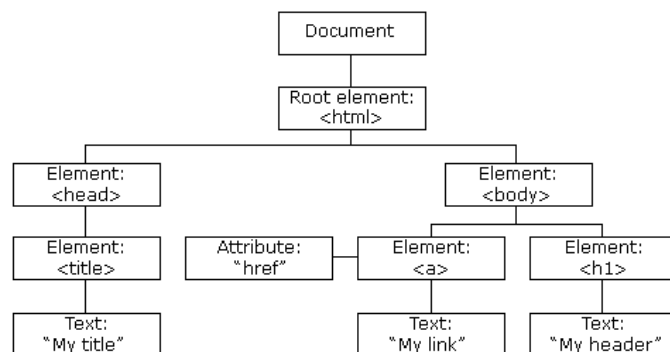
    </body>
    <!-- Fin de corps de page HTML -->

</html>
<!-- Fin de la page HTML -->
```

9. Rappel : le DOM, l'accès aux éléments d'une page HTML et les fonctions de l'objet window

Le DOM

- Le DOM, ou **Document Model Object**, est un standard du web qui permet de décrire et d'accéder aux différents éléments d'une interface, et notamment d'une page web, à partir de tout langage de programmation et notamment le Javascript.
- Chaque élément d'une page web va ainsi pouvoir être facilement désigné et être modifié/utilisé au sein du code Javascript.
- D'un point de vue conceptuel, la page HTML ou le document est vu au sein du DOM sous la forme d'une arborescence dont chaque élément est un nœud :



- Les éléments qui dépendent d'un nœud sont appelés les **enfants** (child) de ce nœud, et le nœud dont dépendent d'autres éléments est appelé **parent**.
- Bien plus, le DOM va permettre de **détecter la survenue d'événement au sein de la page web** : le moment où elle est chargée (onload), où un click souris survient (onclick), où elle est fermée, etc...
- Pour plus de détails, voir :
 - http://fr.wikipedia.org/wiki/Document_Object_Model
 - http://www.w3schools.com/html/dom/dom_intro.asp

Accéder à un élément

- Pour accéder à un élément de la page, on utilisera la fonction `getElementById(« nom »)` :

```
var element=document.getElementById("intro");
```

Associer un élément HTML à un événement du DOM

- La plupart des éléments HTML peuvent être associés à un événement sous la forme (où nomFonction est la fonction Javascript à appeler) :

```
<element onload="nomFonction(param)">
```

Accéder au contenu d'une balise HTML

- Pour accéder au contenu inséré dans une balise HTML, on utilisera la propriété `innerHTML` d'un objet donné :

```
<p id="intro">Hello World!</p>
```

```
<script>
var txt=document.getElementById("intro").innerHTML;
document.write(txt);
</script>
```

Les fonctions disponibles

- Les fonctions disponibles pour gérer, modifier, créer, ajouter, supprimer des éléments du DOM sont très nombreuses.
- Nombreuses également les fonctions permettant de capturer les événements.
- Il n'est pas possible ici d'en dire davantage : nous présenterons les fonctions utilisées au besoin.
- Pour en apprendre plus :
http://www.w3schools.com/html/dom/dom_intro.asp

A part, l'objet implicite window

- Quand une page web est affichée dans un navigateur, un objet implicite est créé, attaché au navigateur, et appelé window
- L'objet window représente la fenêtre du navigateur dans laquelle la page web est affichée.
- Les fonctions de l'objet window ont la particularité d'être accessibles directement. Ces fonctions sont nombreuses et utilisées fréquemment au sein d'un code javascript.
- L'une d'entre elle est la fonction `alert()` qui ouvre un popup avec un bouton OK. Cette fonction est donc accessible de 2 façons :

```
alert("Hello world"); // affiche message
window.alert("Hello world"); // équivalent – affiche message
```

- Un événement de l'objet window utile est notamment `onload` qui permet d'attendre que tous les éléments de la page soient chargés avant d'utiliser le code javascript (voir http://www.w3schools.com/tags/ref_eventattributes.asp) :

```
window.onload = function () {
```

Pour aller plus loin :

- <http://www.w3schools.com/jsref/default.asp>
- Les événements HTML5 :
http://www.w3schools.com/tags/ref_eventattributes.asp

10. Javascript : Associer une fonction à la survenue d'un événement attaché à un élément du DOM

Intro

- Pour pouvoir interagir avec l'utilisateur à partir du navigateur, il faut être en mesure d'intercepter les événements « utilisateur » lors de leur survenue : typiquement, il faut pouvoir détecter un clic souris sur un bouton par exemple.
- HTML et Javascript permettent très simplement d'associer un code javascript à la survenue d'un événement.

Les éléments HTML disponibles :

- Les éléments HTML (ou balises HTML) sont très nombreux : http://www.w3schools.com/tags/ref_byfunc.asp
- Les plus intéressants pour la capture de survenue d'événements vont être notamment les éléments de formulaire notamment :
 - <button> : bouton simple cliquable
 - <select>, <option>, <optgroup>, etc...
- D'une manière générale, **retenir que tout élément HTML peut potentiellement générer des événements qui peuvent être associé à un script.**

Note technique

L'envoi de données vers Arduino à partir d'un formulaire HTML classique (balises <form> et <input>) a été traité dans un tuto précédent : s'y reporter au besoin.
L'envoi de données par formulaire simple entraîne l'envoi d'une requête au serveur et un rafraîchissement de la page.
Ici, l'envoi de données sera fait par capture de l'événement et exécution d'un script de requête Ajax qui ne nécessitera pas de rafraîchissement de la page Web.

Les événements disponibles :

- Les événements disponibles sont nombreux. On distingue :
 - les événements de l'objet implicite **window**, qui correspond rappelons-le à la fenêtre de la page HTML :
 - un des événements important de l'objet window est l'év »nement onload qui survient une fois la page chargée
 - les événements de **formulaire**, notamment onselect, onsubmit...
 - les événements **clavier** : onkeydown, onkeypress, onkeyup
 - les événements **souris** : **onclick**, **ondblclick**, ...
 - les événements des **médias**, incluant les images.
- Au final, seuls quelques événements vont nous être vraiment utiles... mais en même temps, vous comprenez que les possibilités de combinaison sont quasi-illimitées !

Syntaxe générale d'association d'un événement à un élément HTML :

- La plupart des éléments HTML peuvent être associé à un événement sous la forme :

```
<element evenement="nomFonction(param)">
```

- où nomFonction est la fonction Javascript à appeler.

Syntaxe générale d'association d'un événement à une fonction au sein du code Javascript

- Il est également possible, au sein du code javascript lui-même, de définir une fonction attachée à un événement : vous allez reconnaître une syntaxe que nous avons déjà beaucoup utilisée :

```
window.onload = function () { // au chargement de la page  
  
    // code Javascript ici, avec sa syntaxe spécifique...  
  
} // fin onload
```

- Ici, on définit directement la fonction à exécuter sur l'événement window.onload qui survient lorsque la page HTML est chargée : cette façon de faire évite que le code Javascript ne soit appelé avant que les éléments de la page ne soient accessibles, sinon une erreur aurait lieu.
- On pourra faire la même chose pour tous les éléments de la page HTML et tous les événements disponibles, ce qui ouvre pas mal de possibilités.

Exemple avec un bouton :

- Au niveau du code HTML, on fera :

```
<button type="button" onclick="maFonction()">Cliquez moi !</button>
```

- Au niveau du code Javascript, on fera :

```
function maFonction() {  
  
    // code Javascript ici, avec sa syntaxe spécifique...  
  
} // fin maFonction
```

- A noter la possibilité de mettre du code javascript directement dans la balise HTML :

```
<button type="button" onclick="alert('Hello world!')">Click Me!</button>
```

11. HTML + Javascript : associer un événement utilisateur à l'exécution d'un code Javascript

Ce que l'on va faire ici :

- Maintenant que nous avons présenté le principe d'utilisation des événements attachés aux objets d'une page HTML avec Javascript, nous allons passer à un exemple que vous allez pouvoir tester de suite dans votre navigateur.
- Ici, sur un clic bouton nous allons déclencher l'apparition d'une fenêtre de popup : un classique du genre, mais qui permet de se faire la main avant de passer à des choses plus intéressantes.

Le codage HTML

- Au niveau HTML, nous allons insérer dans notre page un simple bouton pour lequel l'événement onclick sera associé à l'appel d'une fonction javascript appelée tout simplement onclickButton() : noter qu'on peut utiliser n'importe quel nom pour la fonction, mais autant faire simple et logique...

```
<button type="button" onclick="onclickButton()">Cliquez-moi</button>
```

Le code Javascript

- Au niveau du code Javascript, nous allons définir la fonction que nous nommerons onclickButton() et qui sera appelée lors de la survenue de l'événement onclick du bouton que nous venons de déclarer. Ici, un clic sur le bouton devra déclencher un popup ce qui se fait avec la fonction alert() de l'objet implicite window, ce qui donne :

```
function onclickButton() { // fonction appelee si clic bouton  
    alert("Clic sur bouton a eu lieu !"); // popup avec message + OK  
} // fin fonction onclickButton
```

Remarque :

A priori, il n'est pas nécessaire d'encadrer cette fonction au sein de la fonction initiale déclenchée par l'événement window.onload car, par définition, lorsqu'un clic bouton aura lieu, le bouton aura déjà été chargé. Placer simplement la fonction dans le head de la page HTML.

Page HTML+Javascript complète d'exemple

- Voici la page HTML complète ainsi obtenue :

```
<!DOCTYPE HTML>

<!-- Debut de la page HTML -->
<html>

    <!-- Debut entete -->
    <head>
        <meta charset="utf-8" /> <!-- Encodage de la page -->
        <title>JavaScript: Test Clic bouton </title> <!-- Titre de la page -->

        <!-- Début du code Javascript -->
        <script language="javascript" type="text/javascript">
            <!--
                function onclickButton() { // fonction appelee si clic bouton

                    alert("Clic sur bouton a eu lieu !"); // popup avec message + OK

                } // fin fonction onclickButton
            //-->
        </script>
        <!-- Fin du code Javascript -->

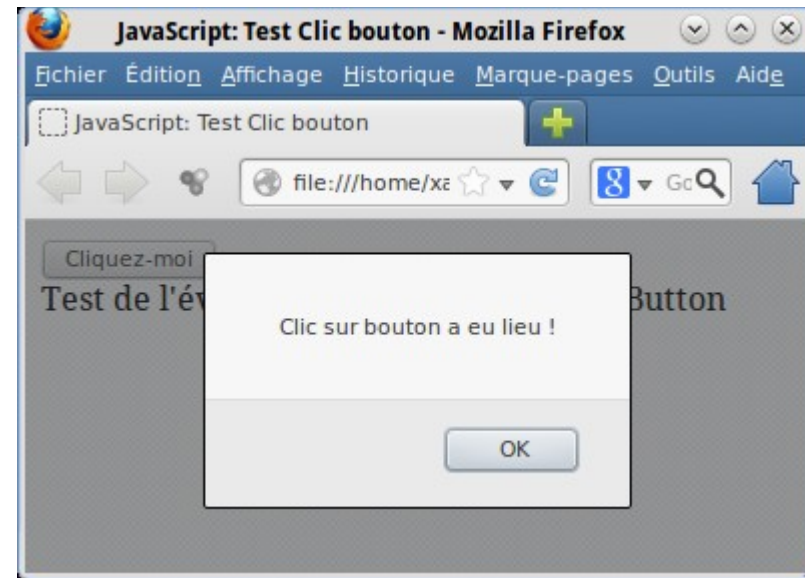
    </head>
    <!-- Fin entete -->

    <!-- Debut Corps de page HTML -->
    <body >
        <button type="button" onclick="onclickButton()">Cliquez-moi</button>
        <br />
        Test de l'&#233;v&#233;nement onclick de l'objet Button

    </body>
    <!-- Fin de corps de page HTML -->
</html>
<!-- Fin de la page HTML -->
```

Résultat

- On obtient une simple page avec un bouton à cliquer : lorsque l'on clique sur le bouton, un popup s'affiche...



- Je pense que vous avez compris le principe. Maintenant que les bases sont posées, nous allons pouvoir faire beaucoup mieux...

12. Rappel : AJAX : principe et intérêt.

Intro

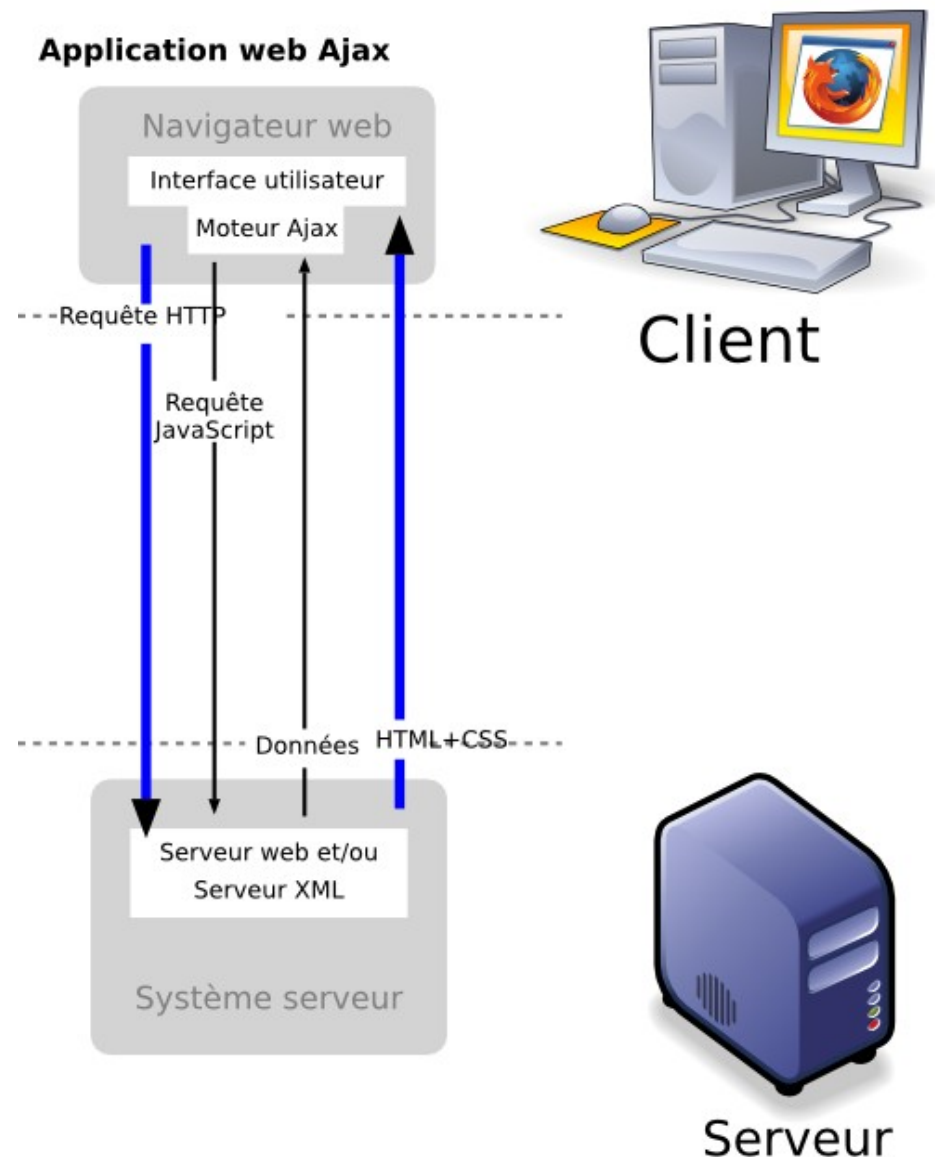
- Dans un tuto précédent, souvenez-vous, nous avons réussi à obtenir un affichage graphique « dynamique » de notre page HTML + Javascript en utilisant la possibilité d'auto-rafraîchissement de la page HTML en ajoutant dans le head une ligne de la forme :

```
<meta http-equiv="refresh" content="3" />
<!-- pour actualisation auto toutes les 3 secondes -->
```

- Comment ça marche ? Comme on l'a dit, toutes les 3 secondes, le navigateur client envoie une requête GET vers le serveur Arduino qui renvoie tout le contenu de la page HTML+Javascript.
- **Le navigateur, à chaque fois, réactualise et affiche l'ensemble de la page**, ce qui a plusieurs inconvénients :
 - cela peut entraîner des « saccades », voir des « blancs » de l'affichage entre 2 rafraîchissements,
 - **la bande passante utilisée et le « travail » du serveur sont importants** puisque c'est toute la page qui est envoyée à chaque fois, alors que seule une petite partie de l'information utile est modifiée entre 2 envois
 - la vitesse de rafraîchissement est réduite, de l'ordre de la seconde.

AJAX : le principe

- Imaginez à présent qu'au lieu de recharger la page complète, le navigateur client, une fois la page chargée une première fois, soit en mesure d'envoyer une requête au serveur pour **simplement récupérer les données utiles pour modifier la page HTML déjà chargée** : c'est exactement ce que va permettre de faire AJAX !!
- AJAX est une technologie web développée dans ce but et veut dire **Asynchronous JavaScript and XML** : cette technologie permet au navigateur d'envoyer une requête au serveur à partir du code Javascript à tout moment et sans avoir à rafraîchir la page !
- Les avantages sont évidents :
 - pas de rafraîchissement visuel de la page complète, donc pas de « saccades » et obtention d'un aspect « progressif » de l'affichage.
 - **bande passante très réduite** : au lieu de centaines de caractères (la page HTML complète), le serveur enverra simplement quelques dizaines de caractères au plus (les valeurs utiles),
 - il sera possible d'**obtenir beaucoup plus rapidement de nouvelles données**, améliorant la rapidité d'affichage « temps-réel » via le réseau.



13. Javascript : L'objet XMLHttpRequest et son utilisation

Intro

Avant toute chose, vous pouvez constater qu'un simple tuto consacré à l'Arduino vous emmène très loin : vous allez ici vous retrouver au cœur des techniques utilisées sur le web... enfin, vous allez en découvrir les fondements, et bien sûr apprendre à les détourner pour arriver à nos fins !

- Le XMLHttpRequest, c'est quoi ça ? Comme j'ai pu le lire quelque part : « [The XMLHttpRequest object is a developer's dream](#) », c'est à dire, l'objet XMLHttpRequest est un rêve de développeur...
- En fait, si vous avez bien suivi ce que je vous ai expliqué, vous allez vite comprendre : [le XMLHttpRequest \(sigle XHR pour la suite\) va permettre d'envoyer une requête vers le serveur à tout moment à partir du code Javascript](#), sans avoir à recharger la page.
- Ce même objet va également permettre de [recevoir des données en provenance du serveur](#)...
- Enfin bref, exactement ce que nous voulons faire. Et la bonne nouvelle, c'est que cet objet est directement disponible sur les navigateurs modernes... notamment Firefox.

Principe général d'utilisation

- On commence par [déclarer l'objet XHR](#) comme on le ferait pour n'importe quel autre objet Javascript,
- Ensuite, on [envoie la requête](#) vers le serveur à l'aide des fonctions `open()` et `send()`
- Puis, l'objet XHR émet un événement lorsque son état se modifie : on teste cet état et [on vérifie qu'une réponse a bien été envoyée](#) par le serveur.
- Enfin, [on gère la réponse du serveur](#) pour en extraire l'information utile et on met à jour les éléments de la page au besoin.

Pour des raisons de sécurité, le serveur à qui est envoyé la requête sera obligatoirement le serveur qui a envoyé la page.

Initialisation

- L'objet XHR s'initialise de la façon suivante :

```
var xhr = new XMLHttpRequest();
```

Cette initialisation est valide avec Firefox, pas forcément avec IE...

Envoi de la requête vers le serveur

- On commence par paramétrer la requête à envoyer vers le serveur à l'aide de la fonction `.open()` de l'objet XHR. Cette fonction reçoit en paramètre :
 - le type de requête `http` : pour nous, on utilisera `GET`
 - l'adresse du fichier à obtenir ou exécuter :
 - en fait, c'est la chaîne qui sera envoyée après `GET` sous la forme `/chemin/fichier`
 - comme c'est nous qui allons coder notre serveur Arduino, on utilisera une chaîne qui nous servira à savoir qu'il s'agit d'une requête AJAX et non une requête classique...
 - Noter qu'on pourra facilement prévoir plusieurs types de requêtes, le code du serveur devant être prévu pour...
 - un drapeau, laisser à `true`
- On fait suivre la fonction `open()` de la fonction `send()` qui envoie la requête, ce qui donne :

```
xhr.open('GET', url, true);  
xhr.send();
```

Gestion de l'évènement onreadystatechange

- L'objet XHR est attaché à l'évènement `onreadystatechange` qui est généré à chaque fois que son état se modifie. Pour faire simple, sachez que cet état vaut 4 lorsque le serveur a envoyé une réponse valable.
- Tant qu'on y est, on pourra tester le statut `http` du serveur, qui devra être 200 si tout est OK.
- On gèrera l'évènement `onreadystatechange` au sein d'une fonction :

```
xhr.onreadystatechange = function() {  
    if (xhr.readyState == 4 && xhr.status == 200) {  
        alert(xhr.responseText); // Données textuelles récupérées  
    }  
};
```

Gestion des données récupérées

- Une fois que les données sont récupérées, on les passera à une fonction de traitement pour mettre à jour les éléments voulus. On réalisera cela grâce à ce que l'on appelle le `callback()` (fonctions enchaînées)

Liens utiles

- Une page très bien faite sur le site du Zéro : <http://www.siteduzero.com/informatique/tutoriels/ajax-et-l-echange-de-donnees-en-javascript/introduction-26>
- http://www.w3schools.com/xml/xml_http.asp

14. Javascript : Code type de gestion d'une requête par XMLHttpRequest

- Maintenant que nous avons présenté l'objet XMLHttpRequest (ou XHR), voyons le code type complet de gestion d'une requête AJAX. Bon, pour être franc, ça se gâte un peu, côté codage : ici, on va passer la fonction de gestion des données reçues en paramètre à la fonction de requête Ajax, pour que les données ne soient traitées que quand elles ont été reçues... On appelle ça le callback... Au pire, si vous comprenez pas tout de suite, revenez-y plus tard. Voici le code :

```
//-----  
window.onload = function() { // fonction au lancement  
    setTimeout(function () {requeteAjax(drawData);}, 5000); // appelle la fonction 1. requete puis 2.drawData (en passant les donnees recues)  
  
} // fin window.onload  
//-----  
  
function requeteAjax(callback) { // la fonction de requete AJAX recoit en parametre une autre fonction qui sera executee une fois donnees recues  
    var xhr = XMLHttpRequest(); // declare objet XHR  
    xhr.onreadystatechange = function() { // fonction de gestion etat objet XHR appelee lors changement etat  
        if (xhr.readyState == 4 && xhr.status == 200) { // verifie etat 4 = reponse serveur finie et http 200 = OK  
            alert(xhr.responseText); // pour debug  
            callback(xhr.responseText); // appelle la fonction de callback recue en parametre en lui passant texte recu du serveur  
        } // fin if  
    }; // fin fonction onreadystatechange  
  
    xhr.open("GET", "/ajax", true); // definition de la requete - ici GET et chaine qui sera reconnue par serveur Arduino ou url  
    xhr.send(null); // envoi de la requete  
  
} // fin fonction requeteAjax  
//-----  
  
function drawData(stringDataIn) { // fonction de gestion des donnees recues - fonction de callback passee en parametre a la fonction de requete AJAX  
    alert(stringDataIn); // debug  
  
} // fin fonction drawData
```

- Cette fois, il n'est pas possible de tester ce code directement dans le navigateur sur le poste fixe : il va falloir d'emblée le tester à partir du serveur Arduino. Allez, c'est parti !

15. Petit retour sur la chaîne d'url passée en paramètre lors d'une requête avec l'objet XMLHttpRequest

- Comme nous venons de le rappeler, la fonction `open()` de l'objet XMLHttpRequest va permettre de configurer la requête envoyée au serveur : la fonction `open()` reçoit en paramètre :
 - le type de requête http : pour nous, on utilisera GET
 - l'adresse (ou l'url) du fichier à obtenir ou à exécuter sur le serveur : **en fait, c'est la chaîne qui sera envoyée après GET sous la forme /chemin/fichier**
 - un drapeau, laisser à true

- Par exemple, si la fonction `open()` est ainsi définie :

```
xhr.open("GET", "/ajax", true); // définition de la requete
```

- le serveur (et donc Arduino dans notre cas) recevra l'entête Http suivante (vous pouvez la visualiser dans le Terminal Série avec l'un des programmes précédents de serveur Arduino Ajax déjà utilisé) :

```
GET /ajax HTTP/1.1
```

```
Host: 192.168.1.100
```

```
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:18.0)
```

- Dans la « vraie vie » d'un « vrai » développeur web, qui travaille avec un « vrai » serveur Http tournant sous Apache (au hasard... non pas tout à fait : en effet, 65 % des serveurs web http utilisent Apache – voir ici : http://fr.wikipedia.org/wiki/Apache_HTTP_Server)... Pour un développeur donc qui travaille avec un vrai serveur http tournant sous Apache, il faudra que le fichier indiqué existe... sinon, l'utilisateur recevra une belle erreur 404 ou ce genre de chose...
- Mais nous, avec notre serveur Arduino « fait maison » et « cuisinés aux petits oignons », on peut faire ce qu'on veut... !!! Le serveur, c'est nous qui le programmons... et donc, en pratique, **TOUT SE PASSE COMME SI LE PARAMETRE D'URL ETAIT EN FAIT UNE SIMPLE CHAÎNE ENVOYÉE À ARDUINO !**
- Et là du coup, on se retrouve en terrain connu : **tout se passe comme si on recevait une chaîne en provenance du port série... à la différence près que la chaîne arrive par le réseau en provenance du navigateur client** et donc il va être possible de la décoder, de l'interpréter, de l'associer à l'exécution d'une fonction ou d'un bout de code Arduino... et même de passer des paramètres numériques comme on le ferait avec une chaîne texte reçue sur le port série.
- Les spécialistes me diront qu'il est possible de passer des paramètres avec une requête envoyée avec l'objet XMLHttpRequest tout comme on le ferait à partir d'un formulaire, qui donnerait quelque chose de la forme (pour une requête GET, car avec POST c'est différent... mais passons :

```
xhr.open("GET", "handlingData.php?variable1=truc&variable2=bidule", true);
```

- Moi, je vous propose de faire quelque chose de beaucoup plus simple et habituel pour un utilisateur d'Arduino : passer en paramètre à la fonction `open()`, à la place d'une url classique valide, **une simple chaîne de son choix** qui sera reconnue par le code Arduino en fonction des besoins. **La seule différence avec un envoi sur le port série est que la fonction `open()` ajoutera un / au début de la chaîne**, mais à part ça, c'est kif kif... On utilisera des chaînes de la forme :
 - « LED=ON » par exemple pour allumer une LED
 - ou avec paramètres numériques de la forme « maFonction(xxx,yyy,zzz) »
 - en fait, les possibilités sont infinies...
- Du coup, je pense que vous saisissez toute les possibilités qui s'offrent à nous : à partir de la simple capture d'un événement « utilisateur » dans le navigateur, il va être possible d'envoyer une requête qui contiendra la chaîne de son choix, chaîne qui sera « interprétée » par le serveur Arduino pour réaliser l'action voulue. **Et le tout sur le réseau à partir d'un simple navigateur client !**
- Pour faire simple, un clic sur un bouton dans le navigateur client va ainsi permettre de contrôler les broches de l'Arduino (une LED par exemple) ou même des dispositifs (positionner un servomoteur par clic bouton à partir d'une valeur saisie dans un champ texte) : en clair, libre à vous de contrôler votre Arduino chez vous depuis New-York si ça vous chante... ou plus simplement de contrôler un dispositif Arduino à partir d'une tablette Android.

16. Fonction de requête Ajax modifiée pour passer une chaîne texte

- En résumé, nous allons envoyer à la demande une requête Ajax vers le serveur Arduino en y incluant une chaîne de son choix. Du coup, il est nécessaire de modifier légèrement la fonction de gestion de requête Ajax que nous avons présenté précédemment de manière à ce que la chaîne envoyée puisse être passée en paramètre à la fonction.
- ce qui nous donne :

```
function requeteAjax(chaineIn , callback) { // la fonction de requete AJAX recoit en parametre la chaine a envoyer au serveur ET une autre fonction qui sera executee une fois donnees recues

    var xhr = XMLHttpRequest(); // declare objet XHR

    xhr.open("GET", chaineIn, true); // definition de la requete - ici GET + chaine personnalisee qui sera reconnue par serveur Arduino
    xhr.send(null); // envoi de la requete

    xhr.onreadystatechange = function() { // fonction de gestion etat objet XHR appelee lors changement etat

        if (xhr.readyState == 4 && xhr.status == 200) { // verifie etat 4 = reponse serveur finie et http 200 = OK

            alert(xhr.responseText); // pour debug
            callback(xhr.responseText); // appelle la fonction de callback recue en parametre en lui passant texte recu du serveur

        } // fin if

    }; // fin fonction onreadystatechange

} // fin fonction requeteAjax
```

- Noter qu'ici, on maintient la réception d'une réponse du serveur qui devra donc au minimum renvoyer une entête http 200 OK, éventuellement suivie d'une chaîne de réponse qui pourra être utilisée par le code Javascript au besoin.

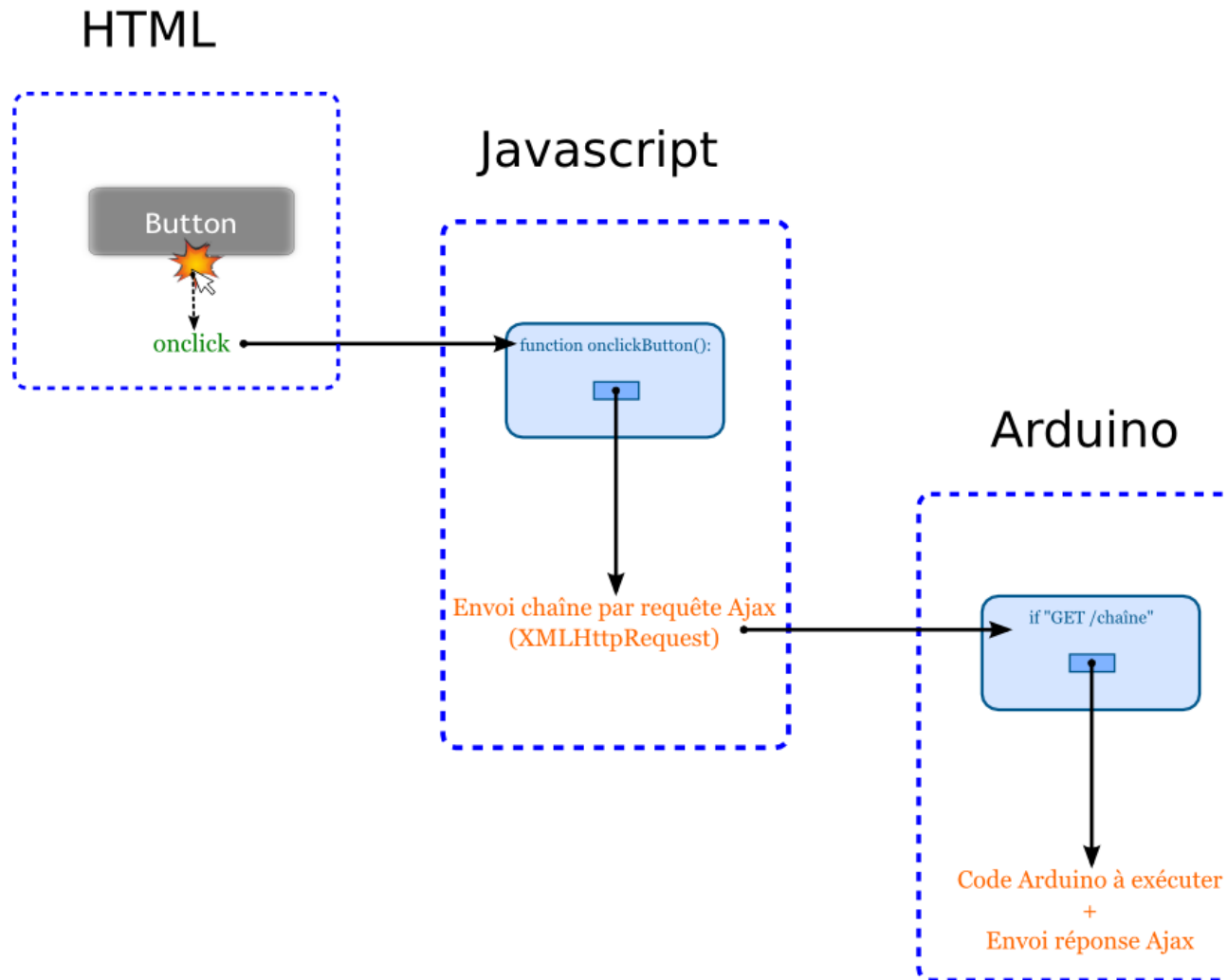


Un truc qui vous semble pas clair ? Pas de panique, vous allez comprendre avec la suite...

En fait, si vous avez déjà bossé les tutos sur la réception de chaînes par le port série, vous allez vite retomber sur vos pieds... !

17. Synthèse : Evènement DOM appelant fonction Javascript envoyant requête AJAX vers Arduino

- Pour être systématique, comprendre que pour chaque action à contrôler depuis le navigateur client, il faudra ajouter dans le code serveur Arduino :
 - l'envoi de la ligne HTML définissant côté client l'élément et la fonction à appeler sur l'évènement voulu
 - l'envoi du code Javascript de la fonction côté client qui enverra la chaîne de requête Ajax vers le serveur Arduino
 - le code Arduino, côté serveur, à exécuter à la réception d'une requête /chaîne Ajax reçue du client

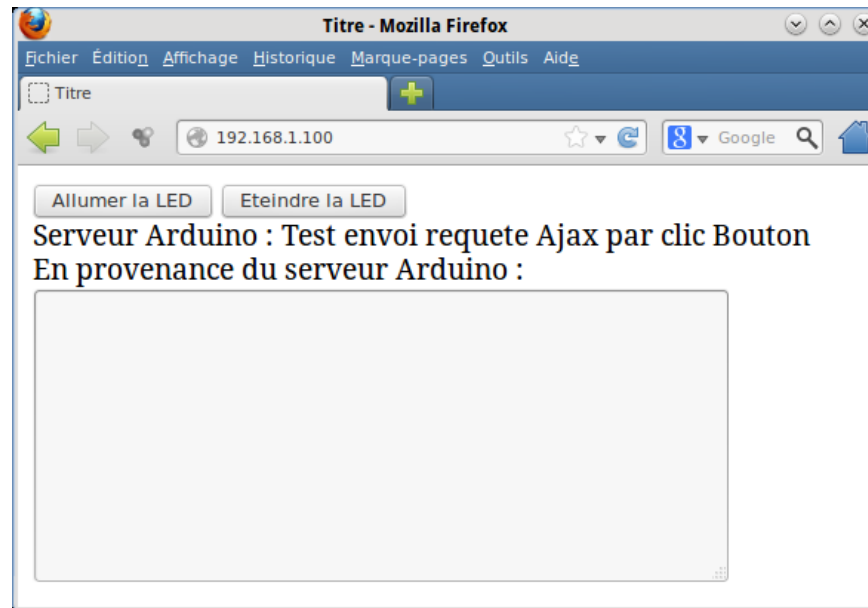


18. Serveur Arduino : Allumer/éteindre une LED par envoi d'une requête Ajax par clic sur un bouton

Ce qu'on va faire ici...

- Ici, nous allons :
 - déclencher l'envoi d'une requête avec la chaîne « LED=ON » lors d'un clic sur un bouton
 - déclencher l'envoi d'une requête avec la chaîne « LED=OFF » lors d'un clic sur un autre bouton.
 - les messages envoyés de réponse par le serveur Arduino seront affichés dans une zone de texte.
- Une LED connectée à la carte Arduino sera allumée/éteinte en conséquence.
- Une nouvelle fois, les échanges http serveur<->client seront visualisés grâce au Terminal Série qui montre ici tout son intérêt !

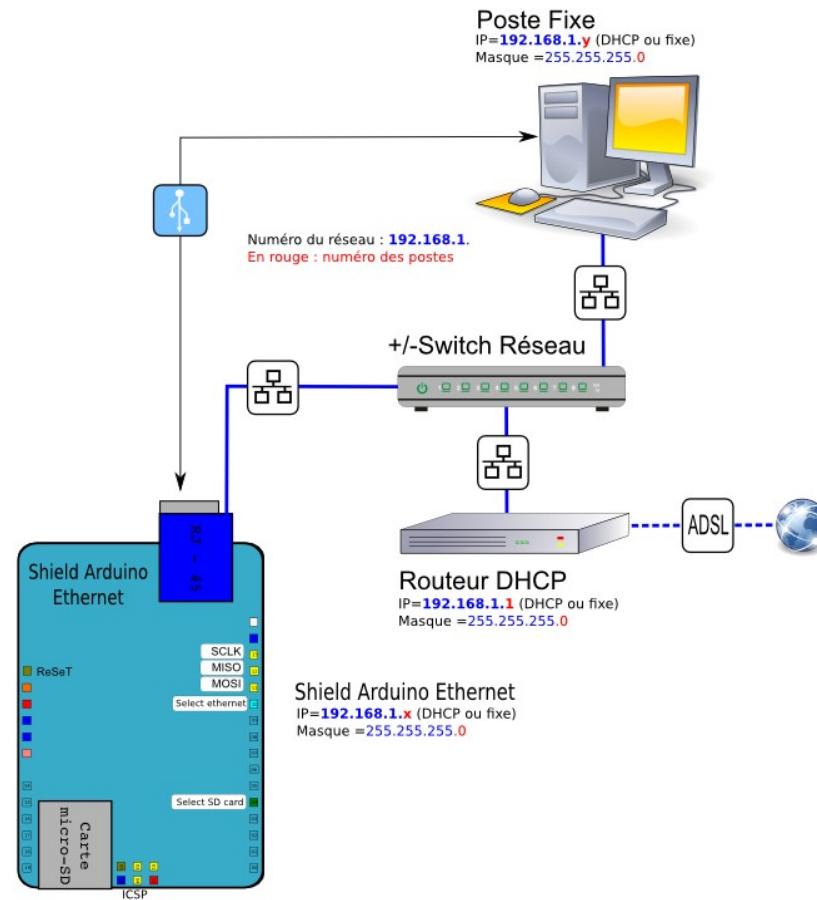
Je rappelle une nouvelle fois qu'il est nécessaire d'utiliser la version **Arduino 1.01** (ou suivante) avec les codes qui suivent.



L'interface que nous allons mettre en place ici...

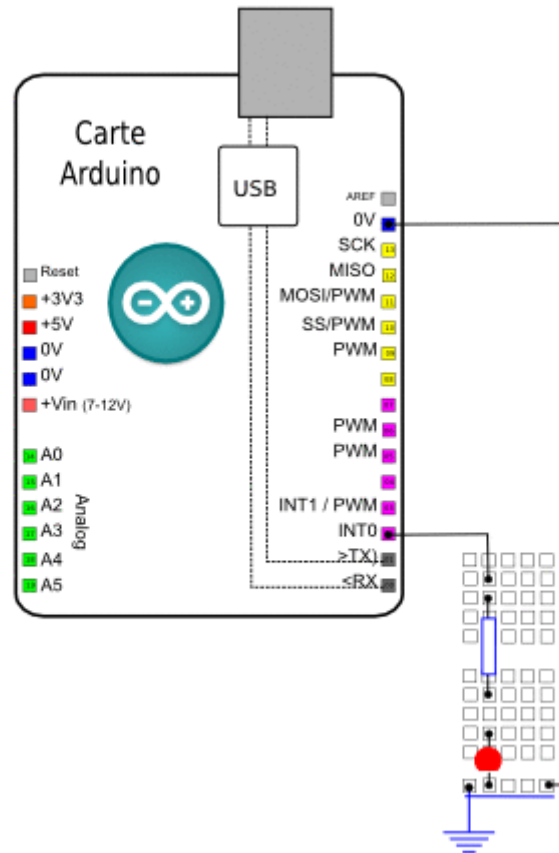
Le schéma du réseau utilisé

- Nous reprenons ici le schéma du réseau local de base que nous avons déjà présenté par ailleurs :



Le montage Arduino utilisé

- Nous allons ici connecter une LED en série avec une résistance sur une broche de la carte Arduino (le shield Ethernet n'est pas représenté ici par souci de simplification) :



Entête déclarative

Inclusion des bibliothèques utiles

- On commence par inclure les bibliothèques
 - **SPI** qui permet au shield Ethernet de communiquer avec la carte Arduino
 - et la bibliothèque **Ethernet** qui comporte toutes les fonctions nécessaires pour la communication du shield Ethernet sur le réseau Ethernet local.

Configuration du shield Ethernet

- On déclare ensuite :
 - un tableau de **byte** correspondant à l'adresse MAC du shield ethernet .
 - un ou plusieurs objets **IPAddress** correspondant aux différentes adresses IP de configuration utilisée. Ici, nous ne définirons que l'adresse IP locale du shield Ethernet.
- On déclare ensuite un objet **EthernetServer** qui configure le shield en tant que serveur. On fixe l'utilisation du port 80 (le port des connexions Web, le plus simple à utiliser car déjà ouvert par défaut sur le routeur...)

Variables utiles

- On déclare une constante de type **int** pour désigner la broche 2, appelée LED
- On déclare enfin des variables utiles pour la réception de la chaîne sur le réseau.

```
// --- Inclusion des bibliothèques ---

#include <SPI.h> // bibliothèque SPI - obligatoire avec bibliothèque Ethernet
#include <Ethernet.h> // bibliothèque Ethernet

// --- Déclaration des variables globales ---

//--- l'adresse mac = identifiant unique du shield
// à fixer arbitrairement ou en utilisant l'adresse imprimée sur l'étiquette du shield
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x1A, 0x71 };

//----- l'adresse IP fixe à utiliser pour le shield Ethernet ---
IPAddress ipLocal(192,168,1,100); // l'adresse IP locale du shield Ethernet
// ATTENTION : il faut utiliser une adresse hors de la plage d'adresses du routeur DHCP
// pour connaître la plage d'adresse du routeur : s'y connecter depuis un navigateur à l'adresse xxx.xxx.xxx.1
// par exemple : sur livebox : plage adresses DHCP entre .10 et .50 => on peut utiliser .100 pour le shield ethernet

// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---

//--- création de l'objet serveur ---
EthernetServer serveurHTTP(80); // crée un objet serveur utilisant le port 80 = port HTTP

String chaineRecue=""; // déclare un string vide global pour réception chaîne requête
int comptChar=0; // variable de comptage des caractères reçus

int LED=2; // broche utilisée pour la LED
```

Fonction **setup()**

Initialisation série

- On initialise la connexion série

Configuration de la broche utilisée

- On configure la broche utilisée en sortie

Initialisation du shield Ethernet

- On initialise le module Ethernet avec la fonction `Ethernet.begin()`. Bien comprendre que cette fonction initialise simplement le shield Ethernet d'un point de vue matériel. A ce stade, il n'est configuré ni en serveur, ni en client.

Affichage de l'adresse IP du shield Ethernet

- On affiche l'adresse IP attribuée au module. Remarquer que l'instruction `print` supporte l'objet `IPAddress`.

Initialisation du serveur

- Logiquement, on initialise le serveur à l'aide de l'instruction `begin()`

```
void setup() { // debut de la fonction setup()

// --- ici instructions à exécuter 1 seule fois au démarrage du programme ---

// ----- Initialisation fonctionnalités utilisées -----

Serial.begin(115200); // Initialise connexion Série

//----- broches ----
pinMode(LED,OUTPUT); // met la broche en sortie

//---- initialise la connexion Ethernet avec l'adresse MAC du module Ethernet, l'adresse IP Locale
//---- +/- l'adresse IP du serveurDNS , l'adresse IP de la passerelle internet et le masque du réseau local

//Ethernet.begin(mac); // forme pour attribution automatique DHCP - utilise plus de mémoire Flash (env + 6Ko)
Ethernet.begin(mac, ipLocal); // forme conseillée pour fixer IP fixe locale
//Ethernet.begin(mac, ipLocal, serverDNS, passerelle, masque); // forme complète

delay(1000); // donne le temps à la carte Ethernet de s'initialiser

Serial.print(F("Shield Ethernet OK : L'adresse IP du shield Ethernet est : " ));

Serial.println(Ethernet.localIP());

//---- initialise le serveur ----
serveurHTTP.begin();
Serial.println(F("Serveur Ethernet OK : Ecoute sur port 80 (http)"));

} // fin de la fonction setup()
```

Fonction **loop()** (1) : Réception des caractères en provenance du client distant

Déclaration d'un objet client

- On commence par créer un objet **EthernetClient** qui sera local à la boucle **loop()** : ce client existera seulement si une connexion entrante existe, ce qui est testé à l'aide de la fonction **.available()** de l'objet **EthernetServer** précédemment configuré.

Réception des caractères

- Ensuite, si le client existe, après avoir affiché quelques messages,...
- on teste si le client est connecté : ceci est testé à l'aide de la fonction **.connected()** de l'objet **EthernetClient**.
- Puis, à l'aide d'une boucle **while()** et de la fonction **.available()** de l'objet **EthernetClient**, qui bouclera tant qu'un caractère sera présent : on affiche le caractère reçu et on l'ajoute à une chaîne de réception
- Une condition permet d'éviter la surcharge en réception au delà de 100 caractères.

```
void loop(){ // debut de la fonction loop()

// crée un objet client basé sur le client connecté au serveur
EthernetClient client = serveurHTTP.available();

if (client) { // si l'objet client n'est pas vide
// le test est VRAI si le client existe

// message d'accueil dans le Terminal Série
Serial.println (F("-----"));
Serial.println (F("Client present !"));
Serial.println (F("Voici la requete du client:"));

////////// Réception de la chaine de la requete //////////

//-- initialisation des variables utilisées pour l'échange serveur/client
chaineRecue=""; // vide le String de reception
comptChar=0; // compteur de caractères en réception à 0

if (client.connected()) { // si le client est connecté

////////// Réception de la chaine par le réseau //////////
while (client.available()) { // tant que des octets sont disponibles en lecture
// le test est vrai si il y a au moins 1 octet disponible

char c = client.read(); // l'octet suivant reçu du client est mis dans la variable c
comptChar=comptChar+1; // incrémente le compteur de caractère reçus

Serial.print(c); // affiche le caractère reçu dans le Terminal Série

//--- on ne mémorise que les n premiers caractères de la requete reçue
//--- afin de ne pas surcharger la RAM et car cela suffit pour l'analyse de la requete
if (comptChar<=100) chaineRecue=chaineRecue+c; // ajoute le caractère reçu au String pour les N premiers caractères
//else break; // une fois le nombre de caractères dépassés sort du while

} // --- fin while client.available = fin "tant que octet en lecture"

Serial.println (F("Reception requete terminee"));
```


Fonction **loop()** (2) : Affichage, analyse de la chaîne reçue et envoi de la réponse aux requêtes Ajax

- Ensuite, tout simplement, on affiche la chaîne reçue
- **La clé de ce programme se trouve ici : pour toutes les requêtes reçues de la forme « GET /chaîne », on exécutera le code Arduino voulu**
 - si la requête reçue commence par « GET /LED=ON » : on allumera la LED et on enverra une entête http + message de réponse
 - si la requête reçue commence par « GET /LED=OFF » : on éteindra la LED et on enverra une entête http + message de réponse

```
//////////////////// Analyse de la requete reçue //////////////////////
Serial.println(F("----- Analyse de la requete recue -----")); // analyse le String de la requete

//////////////////// Affichage de la requete reçue //////////////////////
Serial.println(F("----- Affichage de la requete recue -----")); // affiche le String de la requete
Serial.println (F("Chaîne prise en compte pour analyse : "));
Serial.println(chaineRecue); // affiche le String de la requete pris en compte pour analyse

//////////////////// Analyse de la requete reçue //////////////////////
Serial.println(F("----- Analyse de la requete recue -----")); // analyse le String de la requete

//----- analyse si la chaîne reçue est une requete GET -----
if (chaineRecue.startsWith("GET /LED=ON")) {

    // -- message debug --
    Serial.println (F("Chaîne recue = "));
    Serial.println ("LED=ON");

    // action à exécuter
    digitalWrite(LED,HIGH); // allume la LED

    // envoi reponse à la requete Ajax
    envoiEnteteHTTP(client); // envoi entete HTTP OK 200 vers le client
    client.println(F("LED allumee"));
    Serial.println(F("LED allumee")); // debug

} // fin if GET /LED=ON

else if (chaineRecue.startsWith("GET /LED=OFF")) {

    // -- message debug --
    Serial.println (F("Chaîne recue = "));
    Serial.println ("LED=OFF");

    // action à exécuter
    digitalWrite(LED,LOW); // allume la LED

    // envoi reponse à la requete Ajax
    envoiEnteteHTTP(client); // envoi entete HTTP OK 200 vers le client
    client.println(F("LED eteinte"));
    Serial.println(F("LED eteinte")); // debug

} // fin if GET /LED=OFF
```

Fonction **loop()** (3) : Envoi de la page HTML + Javascript : Envoi de la réponse Http, du début et du head

- Sinon, si la requête commence par GET sans être suivie de la chaîne attendue, on considère qu'il s'agit d'une requête principale et dans ce cas on envoie la page HTML + Javascript complète. On commence par envoyer une entête http (voir fonction dédiée commune ci-dessous)
- Par un jeu de **println()**, on envoie la page HTML avec :
 - les balises **<html>** et **</html>** de début et fin de page
 - les balises **<head>** et **</head>** d'entête
 - **<body>** et **</body>** du corps de la page

```
else if (chaineRecue.startsWith("GET")) { // si la chaine recue commence par GET et pas une réponse précédente = on envoie page entiere

    Serial.println (F("Requete HTTP valide !"));

    envoiEnteteHTTP(client); // envoi entete HTTP OK 200 vers le client

    //--- la réponse HTML à afficher dans le navigateur

    //----- début de la page HTML -----
    client.println(F("<!DOCTYPE html>"));
    client.println(F("<html>"));

    //----- head = entete de la page HTML -----
    client.println(F("<head>"));

    client.println(F("<meta charset=\"utf-8\" />")); // fixe encodage caractères - utiliser idem dans navigateur
    client.println(F("<title>Titre</title>")); // titre de la page HTML
```

Remarque technique :

Noter l'utilisation abondante de la forme **println(F(« chaîne »))** qui a pour effet de stocker les chaînes de caractères directement dans la mémoire programme Flash au lieu de les placer dans la RAM dont la taille est limitée : cette façon de faire est **INDISPENSABLE** dès que l'on utilise de nombreuses chaînes de caractères dans un code sous peine de bloquer l'exécution par saturation de la Ram de l'Arduino.

Je rappelle ici que l'Arduino dispose de 3 mémoires : la Ram (2Ko), la mémoire programme Flash (30Ko) et l'Eeprom de petite taille.

Fonction **loop()** (4) : Head (2) : Début du code Javascript et entête déclarative du code Javascript

- A ce niveau, nous insérons la balise script et nous insérons à l'aide de **println()** successifs le code javascript voulu.
- On définit les variables globales et objets utiles : [ici](#), [simplement l'objet zone de texte utilisé](#).

```
//===== bloc de code javascript =====  
client.println(F("<!-- Début du code Javascript  -->"));  
client.println(F("<script language=\"javascript\" type=\"text/javascript\">"));  
client.println(F("<!--      "));  
  
// variables / objets globaux - a declarer avant les fonctions pour eviter problemes de portee  
client.println(F("var textarea=null;"));
```

Fonction `loop()` (4) : Head (2) : Les fonctions associées aux événements des éléments HTML

- Ensuite, nous insérons les fonctions associées aux événements des éléments HTML :
 - la fonction associée au clic sur un bouton destiné à allumer la LED
 - la fonction associée au clic sur un bouton destiné à éteindre la LED
- Ces fonctions envoient la requête Ajax avec envoi de la chaîne personnalisée voulue : on appelle la fonction d'envoi de requête Ajax (appelée ici `requeteAjax`), en passant en paramètre la chaîne personnalisée ET la fonction de gestion de la réponse Ajax (fonction de « callback », appelée ici `drawData`) :

```
client.println(F("function onclickButtonON() { // click Button ON"}));

//client.println(F("setTimeout(function () {requeteAjax(drawData);}, delai;}));
client.println(F("requeteAjax(\"LED=ON\", drawData);")); // envoi requete avec chaine et fonction de gestion resultat

client.println(F("{} // fin onclickButtonON"));

client.println(F("function onclickButtonOFF() { // click Button OFF"}));

//client.println(F("setTimeout(function () {requeteAjax(drawData);}, delai;}));
client.println(F("requeteAjax(\"LED=OFF\", drawData);")); // envoi requete avec chaine et fonction de gestion resultat

client.println(F("{} // fin onclickButtonOFF"));
```

Note technique :

Le truc le plus « compliqué » à comprendre ici est le principe de fonction de « callback » passée en paramètre à une autre fonction.

Ici, l'instruction `requeteAjax(drawData);` signifie :

« appeler et exécuter la fonction `requeteAjax()` puis une fois l'exécution terminée appeler et exécuter la fonction `drawData()` en lui passant le résultat issu de l'exécution de la première fonction `requeteAjax` »

Les 2 fonctions javascript en question, `requeteAjax()` et `drawData()` ont des noms arbitraires que j'ai choisi et sont définies ci-dessous.

Si vous avez bien compris ça, alors il n'y aura aucun mystère dans ce code pour vous !

Sinon, revenez-y calmement à l'occasion, ce qui ne vous empêche pas de passer à la suite dès maintenant, bien sûr.

Fonction **loop()** (4) : Head (2) : Envoi de la fonction appelée au chargement de la page HTML

- Ensuite, on inclut la fonction appelée lors du chargement initial de la page HTML, sur l'évènement window.onload :
 - ici, on se contente de déclarer l'objet zone de texte utilisé et on initialise sa valeur à une chaîne vide.

```
//----- Fonction principale exécutée au chargement de la page -----  
  
client.println(F("window.onload = function () { // au chargement de la page"}));  
  
    //client.println(F("canvas = document.getElementById(\"nomCanvas\"); // declare objet canvas a partir id = nom ");  
    client.println(F("textarea = document.getElementById(\"textarea\"); // declare objet canvas a partir id = nom ");  
    client.println(F("textarea.value=\"\";")); // efface le contenu  
    client.println(F(""));  
  
    //client.println(F("setTimeout(function () {requeteAjax(drawData);}, delai);"));  
  
client.println(F("} // fin onload"));
```

Fonction **loop()** (5) : Head (3) : envoi de la fonction Javascript de requete Ajax avec chaîne personnalisée

- La suite du code Javascript de la page est constitué par la définition de la fonction de requete Ajax à l'aide du fameux objet XMLHttpRequest que nous avons présenté précédemment. Je rappelle ici que son utilisation passe successivement :
 - on déclare un objet XMLHttpRequest
 - on définit la requête à envoyer avec la fonction open() de l'objet XHR : **c'est ici qu'est définie la partie « chaîne » de la requête qui sera reconnue par le code Arduino comme expliqué précédemment. On utilise ici en paramètre la chaîne reçue par la fonction.**
 - et on envoie la requête au serveur avec la fonction send() de l'objet XHR
 - puis on capture l'évènement onreadystatechange de l'objet XHR et on y associe une fonction où :
 - lorsque sa valeur vaut 4, c'est à dire lorsque le serveur a fini de répondre,
 - et que le serveur a renvoyé une réponse http 200 OK
 - alors on exécute le code de gestion de la réponse reçue
- Comme expliqué juste avant, cette fonction reçoit en paramètre une fonction, appelée callback ici, à laquelle sera passé le résultat de la propriété **responseText** de l'objet XHR et qui correspond à la chaîne reçue en provenance du serveur.

ATTENTION : les explications que je donne ici concerne bien sûr le code Javascript qui est envoyé au client et qui sera exécuté par le client. Le code Arduino lui ne fait qu'envoyer les chaînes du code Javascript au client. En un mot, on envoie du code Javascript à partir du code Arduino.

```
client.println(F("function requeteAjax(chaineIn, callback) { "}); // debut envoi requete avec chaine personnalisee

client.println(F("var xhr = XMLHttpRequest(); "));

client.println(F("xhr.open(\"GET\", chaineIn, true);")); // envoi requete avec chaine personnalisee
client.println(F("xhr.send(null);"));

//----- gestion de l'évènement onreadystatechange -----
client.println(F("xhr.onreadystatechange = function() { "));

client.println(F("if (xhr.readyState == 4 && xhr.status == 200) {"));

client.println(F("//alert(xhr.responseText);"));
client.println(F("callback(xhr.responseText);"));

client.println(F("} // fin if "));

client.println(F("}; // fin function onreadystatechange"));
//----- fin gestion de l'évènement onreadystatechange -----

client.println(F("} // fin fonction requeteAjax"));
```

Fonction **loop()** (6) : Head (4) : envoi de la fonction Javascript de gestion des données reçues du serveur et fin de script

- Ensuite on envoie la fameuse fonction de « callback » qui sera appelée une fois la réponse à la requête Ajax reçue : [cette fonction reçoit en paramètre la chaîne reçue du serveur](#).
- Ici, on ne fait rien d'extraordinaire : on ajoute simplement la chaîne reçue à la zone de texte, à la façon Terminal Série du logiciel Arduino, grâce à des fonctions Javascript dont vous trouverez facilement l'explication par ailleurs : [le corps de la page sera modifié en conséquence, sans que la page ne se rafraîchisse par ailleurs rappelons-le](#). Le rafraîchissement pourra toujours être activé manuellement par un clic souris, mais cela entraînera l'effacement de la page et de la zone de texte par nouvelle réception de la page HTML de départ.

ATTENTION (bis) : Encore une fois les explications que je donne ici concerne bien sûr le code Javascript qui est envoyé au client et qui sera exécuté par le client. Le code Arduino lui ne fait qu'envoyer les chaînes du code Javascript au client. En un mot, on envoie du code Javascript à partir du code Arduino.

```
client.println(F("function drawData(stringDataIn) { ");

// ajoute la réponse au champ texte
client.println(F("textarea.value=textarea.value+stringDataIn;")); // ajoute la chaîne au début - décale vers le bas...
client.println(F("textarea.setSelectionRange(textarea.selectionEnd-1,textarea.selectionEnd-1) ;")); // se place à la fin -1 pour
avant saut de ligne

client.println(F("} // fin fonction drawData"));

// -----

client.println(F("//-->"));
client.println(F("</script>"));
client.println(F("<!-- Fin du code Javascript --> "));

//===== fin du bloc de code javascript =====

client.println(F("</head>"));
//----- fin head = fin entete de la page HTML -----
```


Fonction **loop()** (7) : envoi du body et de la fin de la page HTML de réponse

- De la même façon, par un jeu de **println()**, on envoie la suite du code HTML, c'est à dire ici le body de la page HTML.
- Ici, on ajoute :
 - les deux boutons simples **en associant l'évènement onclick aux fonctions définies précédemment,**
 - une zone de texte pour afficher les messages de réponse du serveur Arduino,
- Suivent les balises de clotûre de la page web

```
client.println("<body>");

client.println(F("Serveur Arduino : Test envoi chaine par requete Ajax sur clic Bouton"));
client.println(F("<br/>"));
client.println(F("<input type=\"text\" id=\"valeur\" />"));
client.println(F("<button type=\"button\" onclick=\"onclickButton()\">Envoyer</button>"));
client.println(F("<br/>"));
client.println(F("En provenance du serveur Arduino :"));
client.println(F("<br/>"));
client.println(F("<textarea id=\"textarea\" rows=\"10\" cols=\"50\" > </textarea>")); // ajoute zone texte vide à la page
client.println(F("<br/>"));

client.println(F("</body>"));
//----- fin body = fin corps de la page -----

client.println(F("</html>"));
//----- fin de la page HTML -----

} // fin if GET
```

Fonction **loop()** (8) : Fermeture de la connexion avec le client

- si la requête reçue n'est pas une « GET », un message indique que la requête n'est pas valide.
- Puis la connexion avec le client est clotûrée.

```
        else { // si la chaine recue ne commence pas par GET
          Serial.println (F("Requete HTTP non valide !"));
        } // fin else

        //----- fermeture de la connexion -----

        // fermeture de la connexion avec le client après envoi réponse
        delay(1); // laisse le temps au client de recevoir la réponse
        client.stop();
        Serial.println(F("----- Fermeture de la connexion avec le client -----")); // affiche le String de la requete
        Serial.println (F(""));

        } // --- fin if client connected

    } //---- fin if client ----

} // fin de la fonction loop()
```

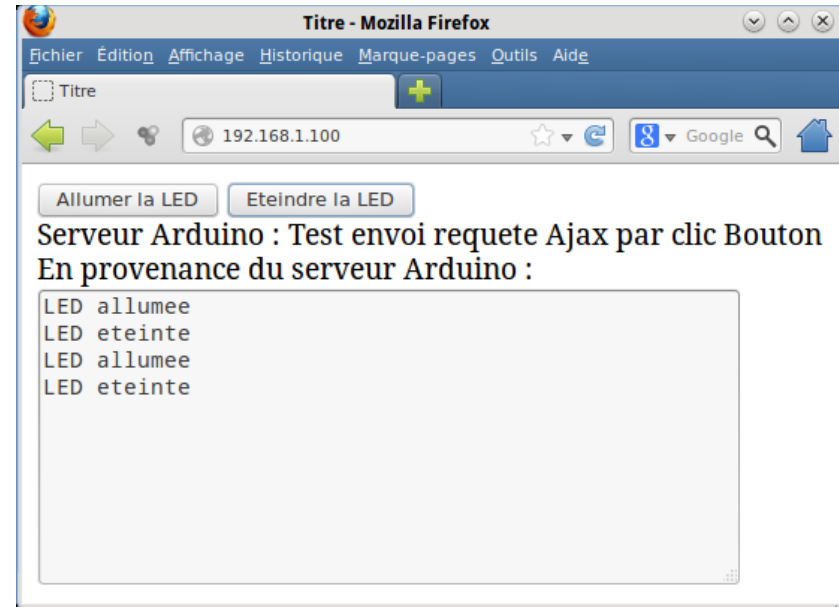
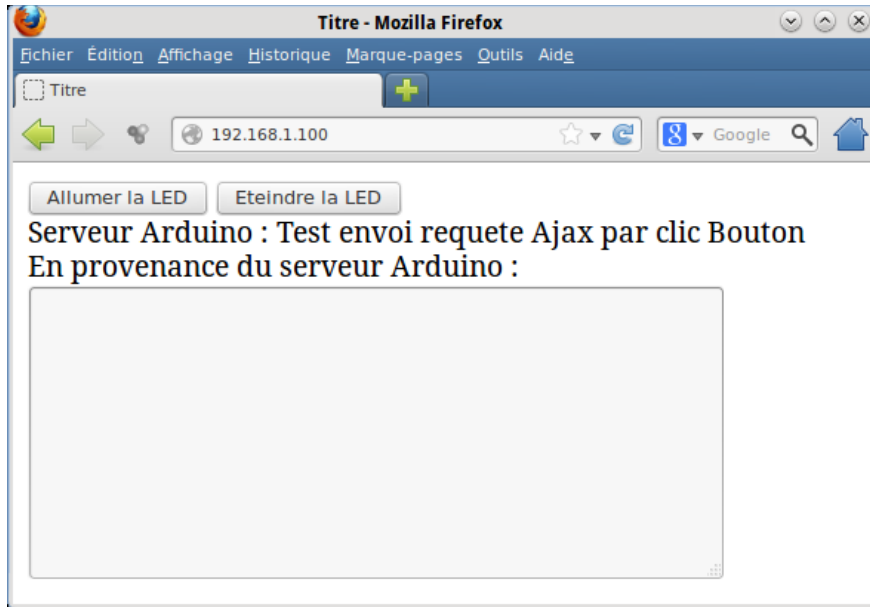
Fonctions communes : Fonction d'envoi de l'entête http

- Dans la mesure où l'on est amené à envoyer plusieurs fois une entête http, autant rassembler le code dans une fonction commune :
 - **HTTP/1.1 200 OK** indique que le serveur a pu traiter la requête
 - Le champ **Content-Type: text/html** indique que la réponse sera du texte ou de l'html (on va voir ça après)
 - Le champ **Connection: close** indique que la connexion doit être fermée après réception de la réponse.
 - Un saut de ligne précède le message de réponse

```
//----- fonctions communes -----  
  
void envoiEnteteHTTP(EthernetClient clientIn){  
  if (clientIn) {  
    //-- envoi de la réponse HTTP ---  
    clientIn.println(F("HTTP/1.1 200 OK")); // entete de la réponse : protocole HTTP 1.1 et exécution requete réussie  
    clientIn.println(F("Content-Type: text/html")); // précise le type de contenu de la réponse qui suit  
    clientIn.println(F("Connection: close")); // précise que la connexion se ferme après la réponse  
    clientIn.println(); // ligne blanche  
  
    //-- envoi en copie de la réponse http sur le port série  
    Serial.println(F("La reponse HTTP suivante est envoyee au client distant :"));  
    Serial.println(F("HTTP/1.1 200 OK"));  
    Serial.println(F("Content-Type: text/html"));  
    Serial.println(F("Connection: close"));  
  
  } // fin si client  
}  
// fin envoiEnteteHTTP
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne un message indiquant l'adresse du serveur (ici 192.168.1.100) et le port d'écoute (ici 80)
- A présent, **ouvrir une fenêtre de navigateur Firefox sur le poste fixe connecté au réseau et saisir l'adresse du shield dans la barre d'adresse** (ici 192.168.1.100). On doit alors voir apparaître dans le Terminal Série toute une série de lignes de texte correspondant à la requête envoyée par le navigateur. Dans le navigateur, on doit ici voir dans la fenêtre Firefox, :
 - les 2 boutons ainsi que la zone texte : cliquez sur le bouton de votre choix : la LED s'allume/s'éteint et le message s'affiche dans la zone de texte.



Encore une fois, un code totalement réalisé du « côté Arduino » est qui assure la programmation du navigateur client en Javascript !

Ici, l'envoi d'une requête Ajax personnalisée permet de contrôler une LED...

Les bases pour le contrôle de dispositifs ON/OFF via le réseau sont posées.... à vous de jouer !

La suite ?

Envoyer des chaînes avec valeurs numériques passées en paramètre pour contrôler des dispositifs « variables » : contrôler un servomoteur par exemple... à partir du navigateur.

Allez, on continue... !

19. Recevoir des chaînes avec paramètres : Installation et présentation de ma librairie Utils

La librairie Utils

- La librairie Utils, dans laquelle j'ai rassemblé plusieurs fonctions utiles, permet de recevoir et d'extraire des paramètres numériques au sein de chaînes texte reçues notamment sur le port série.
- Le très gros avantage de cette librairie est de simplifier grandement le code qui serait beaucoup plus lourd pour obtenir le même résultat uniquement avec des fonctions Arduino de base.

Télécharger la librairie

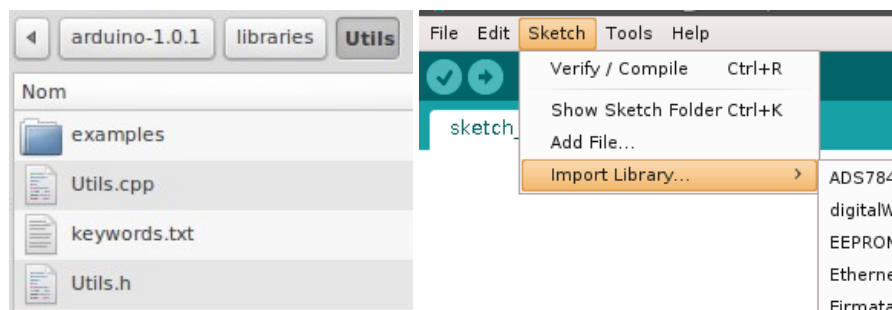
- Ma librairie Utils est disponible ici : http://www.mon-club-elec.fr/pmwiki_reference_lib_arduino_perso/pmwiki.php?n=Main.HomePage

Documentation de la librairie

- http://www.mon-club-elec.fr/pmwiki_reference_lib_arduino_perso/pmwiki.php?n=Main.HomePage

Installation

- Télécharger l'archive. au format zip ou autre. L'extraire
- **Vérifier que le nom du répertoire de la librairie est strictement le même que le nom du fichier *.h ou *.cpp principal. Corriger au besoin. Ici le nom est Utils**
- Copier/coller le répertoire de la librairie dans le répertoire libraries de votre répertoire Arduino
- Relancer Arduino et vérifier que la librairie est présente dans le menu **Sketch > ImportLibrary**.



Le constructeur principal

- Le constructeur principal se nomme Utils et est de la forme :

Utils utils;

Fonctions de la librairie

Fonctions de réception de chaîne de caractères sur le port Série :

- String [waitingString](#) (boolean debugIn) : réception d'une chaîne sur le port Série
- String [waitingString](#) () : réception d'une chaîne sur le port Série
- void [waitForString](#)(String stringForWaitIn) : attente de la réception d'une chaîne précise sur le port Série

Fonctions d'analyse de chaîne de caractères :

- String [testInstructionString](#) (String chaîneTest, String chaîneRefIn): extraction d'un paramètre texte
- boolean [testInstructionLong](#) (String chaîneReception,String chaîneTest, int nbParam, long paramsIn[]): extraction d'un ou plusieurs paramètres entiers

Code d'exemple

```
#include <Utils.h> // inclusion de la librairie
Utils utils; // déclare objet racine d'accès aux fonctions de la librairie Utils

String chaîneReception=""; // déclare un String
long params[6]; // déclare un tableau de long - taille en fonction nombre max
paramètres attendus

void setup() {

    Serial.begin(115200); // Initialisation vitesse port Série
    // Utiliser vitesse idem coté Terminal série
    Serial.println("Saisir une chaîne de la forme FONCTION(valeur)"); // message
    initial

} // fin setup

void loop() {

    chaîneReception=utils.waitingString();// sans debug

    if (chaîneReception!="") { // si une chaîne a été reçue

        if(utils.testInstructionLong(chaîneReception,"FONCTION(",1,params)==true)
        { // si chaîne FONCTION(valeur) bien reçue

            Serial.println("Arduino a reçu le parametre : " + (String)params[0]);

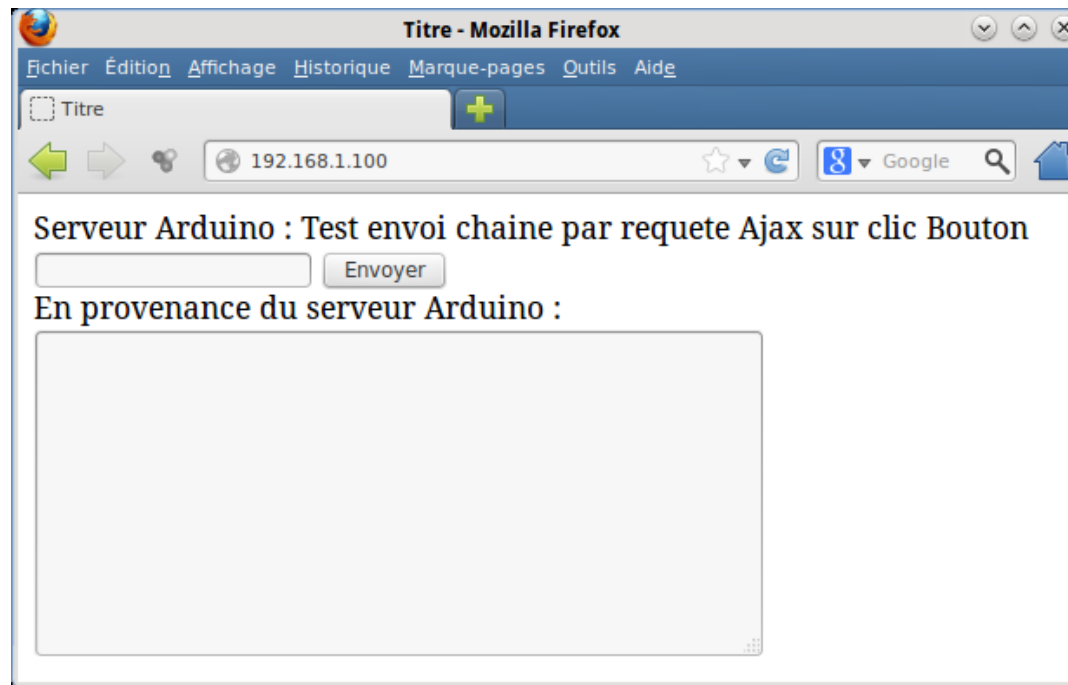
        } // fin si testInstructionLong==true
    } // fin // si une chaîne a été reçue
} // fin loop
```

20. Serveur Arduino : Contrôler un servomoteur par envoi d'une requête Ajax avec paramètre numérique

Ce qu'on va faire ici...

- Ici, nous allons :
 - déclencher l'envoi lors d'un clic sur un bouton d'une requête avec une chaîne saisie dans un champ texte (ici « servo(xxx) », où xxx sera une valeur numérique). Noter que le champ et le bouton permettront facilement d'envoyer des chaînes de test vers le serveur Arduino.
 - afficher les messages de réponse envoyés par le serveur Arduino dans une zone de texte.
- Un servomoteur connecté à la carte Arduino sera positionné en conséquence si une chaîne valide de la forme servo(xxx) est reçue.
- Une nouvelle fois, les échanges http serveur<->client seront visualisés grâce au Terminal Série qui montre ici tout son intérêt !

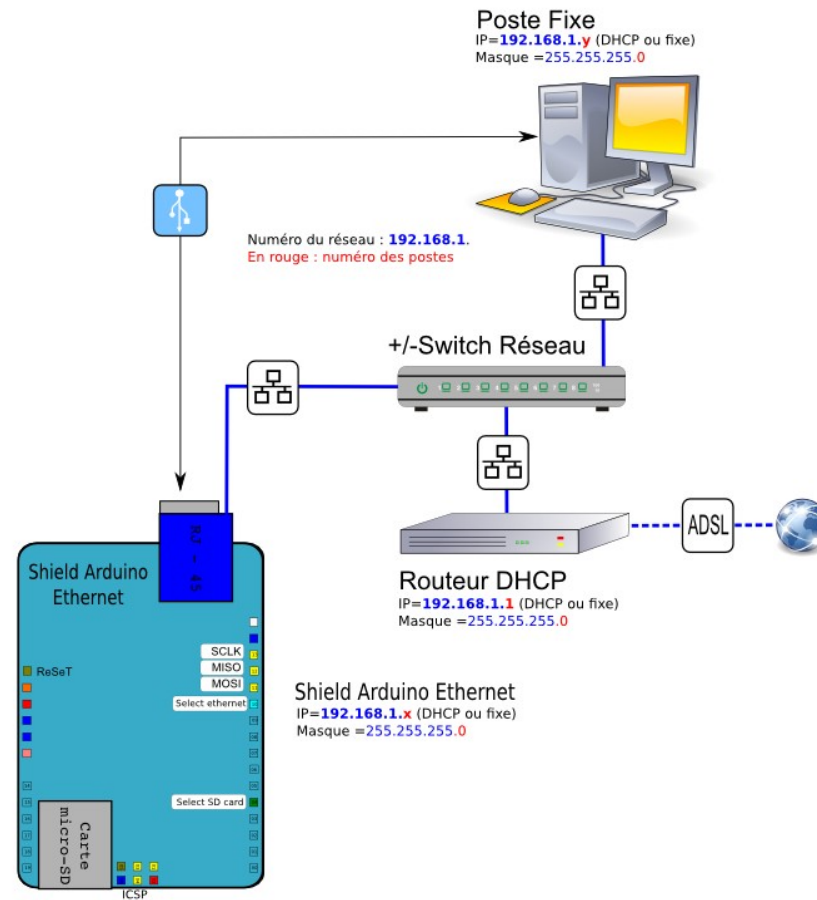
Je rappelle une nouvelle fois qu'il est nécessaire d'utiliser la version **Arduino 1.01** (ou suivante) avec les codes qui suivent.



L'interface que nous allons mettre en place ici...
Une sorte de Terminal Série Arduino, mais fonctionnant sur le réseau !

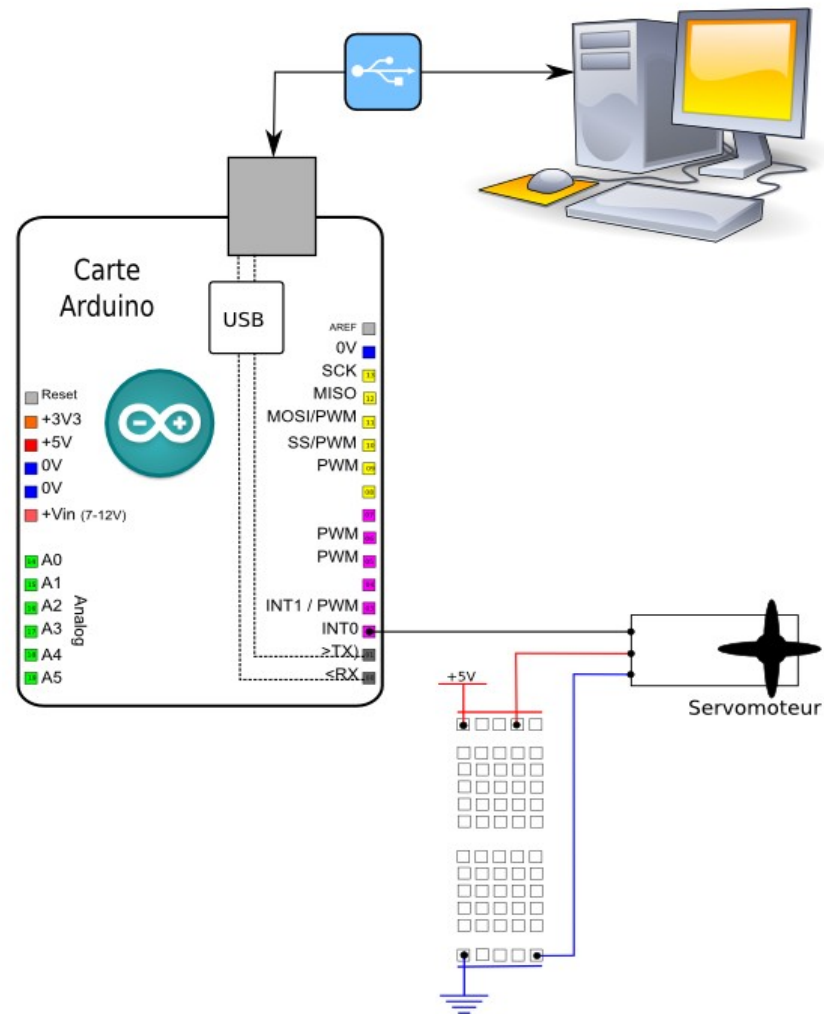
Le schéma du réseau utilisé

- Nous reprenons ici le schéma du réseau local de base que nous avons déjà présenté par ailleurs :



Le montage Arduino utilisé

- Nous allons ici connecter un servomoteur sur une broche de la carte Arduino (le shield Ethernet n'est pas représenté ici par souci de simplification) :



Entête déclarative (1)

Inclusion des bibliothèques utiles

- On commence par inclure les bibliothèques
 - la bibliothèque **Utils** qui contient plusieurs fonctions facilitant le décodage de chaînes numériques avec paramètres
 - la bibliothèque **Servo** qui permet d'utiliser un servomoteur
 - la bibliothèque **SPI** qui permet au shield Ethernet de communiquer avec la carte Arduino
 - et la bibliothèque **Ethernet** qui comporte toutes les fonctions nécessaires pour la communication du shield Ethernet sur le réseau Ethernet local.

```
// --- Inclusion des bibliothèques ---  
  
#include <Utils.h> // inclusion de la bibliothèque  
#include <Servo.h> // inclut la bibliothèque Servo  
#include <SPI.h> // bibliothèque SPI - obligatoire avec bibliothèque Ethernet  
#include <Ethernet.h> // bibliothèque Ethernet
```

Entête déclarative (2)

Configuration du shield Ethernet

- On déclare ensuite :
 - un tableau de **byte** correspondant à l'adresse MAC du shield ethernet .
 - un ou plusieurs objets **IPAddress** correspondant aux différentes adresses IP de configuration utilisée. Ici, nous ne définirons que l'adresse IP locale du shield Ethernet.
- On déclare ensuite un objet **EthernetServer** qui configure le shield en tant que serveur. On fixe l'utilisation du port 80 (le port des connexions Web, le plus simple à utiliser car déjà ouvert par défaut sur le routeur...)

Variables utiles

- On déclare également un objet **Servo** et une constante de broche utilisée pour contrôler le servomoteur
- On déclare un objet **Utils** qui donnera accès à l'ensemble des fonctions de la librairie **Utils**, notamment pour le décodage de chaînes avec paramètres
- On déclare les constantes de paramétrage du servomoteur utilisé (un classique Futaba S3003 dans mon cas)
- On déclare enfin des variables utiles pour la réception de la chaîne sur le réseau et les chaînes d'analyse.

```
// --- Déclaration des variables globales ---

//--- l'adresse mac = identifiant unique du shield
// à fixer arbitrairement ou en utilisant l'adresse imprimée sur l'étiquette du shield
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x1A, 0x71 };

//----- l'adresse IP fixe à utiliser pour le shield Ethernet ---
IPAddress ipLocal(192,168,1,100); // l'adresse IP locale du shield Ethernet
// ATTENTION : il faut utiliser une adresse hors de la plage d'adresses du routeur DHCP
// pour connaître la plage d'adresse du routeur : s'y connecter depuis un navigateur à l'adresse xxx.xxx.xxx.1
// par exemple : sur livebox : plage adresses DHCP entre .10 et .50 => on peut utiliser .100 pour le shield ethernet

// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---

Servo servo; // déclaration d'un objet servomoteur
const int brocheServo=2; // broche du servomoteur

//--- création de l'objet serveur ---
EthernetServer serveurHTTP(80); // crée un objet serveur utilisant le port 80 = port HTTP

Utils utils; // déclare objet racine d'accès aux fonctions de la librairie Utils
long params[6]; // déclare un tableau de long - taille en fonction nombre max paramètres attendus

String chaineRecue=""; // déclare un string vide global pour réception chaine requete
String chaineAnalyse=""; // string vide global pour chaine retenue pour analyse
int comptChar=0; // variable de comptage des caractères reçus

//----- constantes de paramétrage du servomoteur ----
const int posMin=550; // largeur impulsion en µs correspondant à la position 0° du servomoteur
const int posMax=2350; // largeur impulsion en µs correspondant à la position 180° du servomoteur
//----- valeur pour un Futaba S3003 - à adapter à votre situation
```

Fonction **setup()**

Initialisation série

- On initialise la connexion série

Initialisation servomoteur

- On attache le servomoteur à la broche utilisée à l'aide de la fonction **attach()**

Initialisation du shield Ethernet

- On initialise le module Ethernet avec la fonction **Ethernet.begin()**. Bien comprendre que cette fonction initialise simplement le shield Ethernet d'un point de vue matériel. A ce stade, il n'est configuré ni en serveur, ni en client.

Affichage de l'adresse IP du shield Ethernet

- On affiche l'adresse IP attribuée au module. Remarquer que l'instruction **print** supporte l'objet **IPAddress**.

Initialisation du serveur

- Logiquement, on initialise le serveur à l'aide de l'instruction **begin()**

```
void setup() { // debut de la fonction setup()

// --- ici instructions à exécuter 1 seule fois au démarrage du programme ---

// ----- Initialisation fonctionnalités utilisées -----

Serial.begin(115200); // Initialise connexion Série

servo.attach(brocheServo, posMin, posMax); // attache le servomoteur à la broche
// et initialisation des positions extremes

//---- initialise la connexion Ethernet avec l'adresse MAC du module Ethernet, l'adresse IP Locale
//---- +/- l'adresse IP du serveurDNS , l'adresse IP de la passerelle internet et le masque du réseau local

//Ethernet.begin(mac); // forme pour attribution automatique DHCP - utilise plus de mémoire Flash (env + 6Ko)
Ethernet.begin(mac, ipLocal); // forme conseillée pour fixer IP fixe locale
//Ethernet.begin(mac, ipLocal, serverDNS, passerelle, masque); // forme complète

delay(1000); // donne le temps à la carte Ethernet de s'initialiser

Serial.print(F("Shield Ethernet OK : L'adresse IP du shield Ethernet est : " ));

Serial.println(Ethernet.localIP());

//---- initialise le serveur ----
serveurHTTP.begin();
Serial.println(F("Serveur Ethernet OK : Ecoute sur port 80 (http)"));

} // fin de la fonction setup()
```

Fonction **loop()** (1) : Réception des caractères en provenance du client distant

Déclaration d'un objet client

- On commence par créer un objet **EthernetClient** qui sera local à la boucle **loop()** : ce client existera seulement si une connexion entrante existe, ce qui est testé à l'aide de la fonction **.available()** de l'objet **EthernetServer** précédemment configuré.

Réception des caractères

- Ensuite, si le client existe, après avoir affiché quelques messages,...
- on teste si le client est connecté : ceci est testé à l'aide de la fonction **.connected()** de l'objet **EthernetClient**.
- Puis, à l'aide d'une boucle **while()** et de la fonction **.available()** de l'objet **EthernetClient**, qui bouclera tant qu'un caractère sera présent : on affiche le caractère reçu et on l'ajoute à une chaîne de réception
- Une condition permet d'éviter la surcharge en réception au delà de 100 caractères.

```
void loop(){ // debut de la fonction loop()

// crée un objet client basé sur le client connecté au serveur
EthernetClient client = serveurHTTP.available();

if (client) { // si l'objet client n'est pas vide
// le test est VRAI si le client existe

// message d'accueil dans le Terminal Série
Serial.println (F("-----"));
Serial.println (F("Client present !"));
Serial.println (F("Voici la requete du client:"));

////////// Réception de la chaine de la requete //////////

//-- initialisation des variables utilisées pour l'échange serveur/client
chaineRecue=""; // vide le String de reception
comptChar=0; // compteur de caractères en réception à 0

if (client.connected()) { // si le client est connecté

////////// Réception de la chaine par le réseau //////////
while (client.available()) { // tant que des octets sont disponibles en lecture
// le test est vrai si il y a au moins 1 octet disponible

char c = client.read(); // l'octet suivant reçu du client est mis dans la variable c
comptChar=comptChar+1; // incrémente le compteur de caractère reçus

Serial.print(c); // affiche le caractère reçu dans le Terminal Série

//--- on ne mémorise que les n premiers caractères de la requete reçue
//--- afin de ne pas surcharger la RAM et car cela suffit pour l'analyse de la requete
if (comptChar<=100) chaineRecue=chaineRecue+c; // ajoute le caractère reçu au String pour les N premiers caractères
//else break; // une fois le nombre de caractères dépassés sort du while

} // --- fin while client.available = fin "tant que octet en lecture"

Serial.println (F("Reception requete terminee"));
```

Fonction **loop()** (2) : Affichage, analyse de la chaîne reçue et envoi de la réponse aux requêtes Ajax

- Ensuite, tout simplement, on affiche la chaîne reçue
- **La clé de ce programme se trouve à nouveau ici :**
 - les requêtes Ajax envoyées seront de la forme « GET /&chaîne= », les caractères & et = permettant de fixer le début et la fin de la chaîne envoyée avec la requête.
 - Pour toutes les requêtes reçues de la forme « GET /& », on exécutera le code Arduino d'analyse de chaîne de façon à analyser et extraire les paramètres numériques : :
 - extraction de la chaîne
 - puis on analysera la chaîne pour s'assurer qu'elle est bien au format servo(xxx) et on extraira la valeur numérique xxx, à l'aide de la fonction `testInstructionLong` de ma librairie `Utils` : si c'est le cas, le servomoteur sera positionné à l'angle indiqué. Des messages sont envoyés au client.

```
//----- analyse si la chaîne reçue est une requête GET avec chaîne format /&chaîne= -----
if (chaîneReçue.startsWith("GET /&")) {

    //----- extraction de la chaîne allant de & à =
    int indexStart=chaîneReçue.indexOf("&");
    int indexEnd=chaîneReçue.indexOf("=");
    Serial.print (F("index debut ="));
    Serial.println (indexStart);
    Serial.print (F("index fin ="));
    Serial.println (indexEnd);

    chaîneAnalyse=chaîneReçue.substring(indexStart+1,indexEnd); // garde chaîne fonction(xxxx) à partir de GET /&fonction(xxxx)=
    // substring : 1er caractère inclusif (d'où le +1) , dernier exclusif

    // -- message debug --
    Serial.print (F("Chaîne reçue = "));
    Serial.println (chaîneAnalyse);

    // -- analyse de la chaîne à analyser --

    if (chaîneAnalyse!="") { // si une chaîne à analyser non vide

        //----- si la chaîne servo(xxx) est reconnue
        if(Utils.testInstructionLong(chaîneAnalyse,"servo(",1,params)==true) { // si chaîne FONCTION(valeur) bien reçue

            Serial.println("Arduino a reçu le parametre : " + (String)params[0]);

            // action à exécuter
            servo.write(params[0]); // positionne le servomoteur dans l'angle voulu

            // envoi réponse à la requête Ajax
            envoiEnteteHTTP(client); // envoi entete HTTP OK 200 vers le client
            client.print(F("Chaîne reçue :"));
            client.println(chaîneAnalyse);
            client.print(F("Parametre reçu :"));
            client.println(params[0]);
            //-- une fois la réponse Ajax terminée, la fonction de callback drawData est exécutée
        } // fin si testInstructionLong==true

    } // fin // si une chaîne analyse a été reçue

} // fin if GET /&
```

Fonction loop() : Réponse Ajax si chaîne reçue non valide :

- On peut également prévoir l'envoi d'une réponse Ajax au cas où aucune chaîne valide n'a été reçue, ce qui donne :

```
// +/- message si chaîne pas reconnue
else { // sinon si chaîne pas reconnue

    // envoi réponse à la requête Ajax
    envoiEnteteHTTP(client); // envoi entête HTTP OK 200 vers le client
    client.print(F("Chaîne reçue :"));
    client.println(chaineAnalyse);
    client.println(F("Chaîne non reconnue !"));
    client.println(F("Saisir chaîne servo(xxx) avec xxx, un angle entre 0 et 180"));
    //-- une fois la réponse Ajax terminée, la fonction de callback drawData est exécutée

} // fin else
```


Fonction **loop()** (3) : Envoi de la page HTML + Javascript : Envoi de la réponse Http, du début et du head

- Sinon, si la requête commence par GET sans être suivie de la chaîne attendue, on considère qu'il s'agit d'une requête principale et dans ce cas on envoie la page HTML + Javascript complète. On commence par envoyer une entête http (voir fonction dédiée commune ci-dessous)
- Par un jeu de **println()**, on envoie la page HTML avec :
 - les balises **<html>** et **</html>** de début et fin de page
 - les balises **<head>** et **</head>** d'entête
 - **<body>** et **</body>** du corps de la page

```
else if (chaineRecue.startsWith("GET")) { // si la chaine recue commence par GET et pas une réponse précédente = on envoie page entiere

    Serial.println (F("Requete HTTP valide !"));

    envoiEnteteHTTP(client); // envoi entete HTTP OK 200 vers le client

    //--- la réponse HTML à afficher dans le navigateur

    //----- début de la page HTML -----
    client.println(F("<!DOCTYPE html>"));
    client.println(F("<html>"));

    //----- head = entete de la page HTML -----
    client.println(F("<head>"));

    client.println(F("<meta charset=\"utf-8\" />")); // fixe encodage caractères - utiliser idem dans navigateur
    client.println(F("<title>Titre</title>")); // titre de la page HTML
```

Remarque technique :

Noter l'utilisation abondante de la forme **println(F(« chaîne »))** qui a pour effet de stocker les chaînes de caractères directement dans la mémoire programme Flash au lieu de les placer dans la RAM dont la taille est limitée : cette façon de faire est **INDISPENSABLE** dès que l'on utilise de nombreuses chaînes de caractères dans un code sous peine de bloquer l'exécution par saturation de la Ram de l'Arduino.

Je rappelle ici que l'Arduino dispose de 3 mémoires : la Ram (2Ko), la mémoire programme Flash (30Ko) et l'Eeprom de petite taille.

Fonction **loop()** (4) : Head (2) : Début du code Javascript et entête déclarative du code Javascript

- A ce niveau, nous insérons la balise script et nous insérons à l'aide de **println()** successifs le code javascript voulu.
- On définit les variables globales et objets utiles : [ici](#), [simplement l'objet zone de texte](#) et [l'objet champ texte utilisés](#).

```
//===== bloc de code javascript =====  
client.println(F("<!-- Début du code Javascript  -->"));  
client.println(F("<script language=\"javascript\" type=\"text/javascript\">"));  
client.println(F("<!--      "));  
  
// variables / objets globaux - a declarer avant les fonctions pour eviter problemes de portee  
client.println(F("var textarea=null;"));  
client.println(F("var textInputX=null;"));
```

Fonction **loop()** (4) : Head (2) : Les fonctions associées aux évènements des éléments HTML

- Ensuite, nous insérons les fonctions associées aux évènements des éléments HTML :
 - la fonction associée au clic sur un bouton destiné à envoyer la chaîne contenue dans le champ texte, encadrée des caractères & et = correspondant au début et à la fin de la chaîne.
- Cette fonction envoie la requête Ajax avec envoi de la chaîne personnalisée voulue : on appelle la fonction d'envoi de requête Ajax (appelée ici `requeteAjax`), en passant en paramètre la chaîne personnalisée ET la fonction de gestion de la réponse Ajax (fonction de « callback », appelée ici `drawData`) :

```
client.println(F("function onclickButton() { // click Button ON"));
```

```
resultat client.println(F("requeteAjax(\"&\"+textInput.value+\"=\"\", drawData);")); // envoi requete avec &chaîne= et fonction de gestion
```

```
client.println(F("} // fin onclickButton"));
```

Note technique :

Le truc le plus « compliqué » à comprendre ici est le principe de fonction de « callback » passée en paramètre à une autre fonction.

Ici, l'instruction `requeteAjax(drawData);` signifie :

« appeler et exécuter la fonction `requeteAjax()` puis une fois l'exécution terminée appeler et exécuter la fonction `drawData()` en lui passant le résultat issu de l'exécution de la première fonction `requeteAjax` »

Les 2 fonctions javascript en question, `requeteAjax()` et `drawData()` ont des noms arbitraires que j'ai choisi et sont définies ci-dessous.

Si vous avez bien compris ça, alors il n'y aura aucun mystère dans ce code pour vous !

Sinon, revenez-y calmement à l'occasion, ce qui ne vous empêche pas de passer à la suite dès maintenant, bien sûr.

Fonction **loop()** (4) : Head (2) : Envoi de la fonction appelée au chargement de la page HTML

- Ensuite, on inclut la fonction appelée lors du chargement initial de la page HTML, sur l'évènement `window.onload` :
 - ici, on se contente de déclarer l'objet zone de texte utilisé et on initialise sa valeur à une chaîne vide
 - ainsi que de déclarer l'objet champ texte,
 - ... de façon à ce que ces 2 objets soient directement accessibles dans le reste du code javascript.

```
//----- Fonction principale exécutée au chargement de la page -----  
  
client.println(F("window.onload = function () { // au chargement de la page"}));  
  
    client.println(F("textarea = document.getElementById(\"textarea\"); // declare objet canvas a partir id = nom "));  
    client.println(F("textarea.value=\"\";")); // efface le contenu  
  
    client.println(F("textInput= document.getElementById(\"valeur\"); // declare objet champ text a partir id = nom"));  
    client.println(F(""));  
  
client.println(F("} // fin onload"));
```

Fonction **loop()** (5) : Head (3) : envoi de la fonction Javascript de requete Ajax avec chaîne personnalisée

- La suite du code Javascript de la page est constitué par la définition de la fonction de requete Ajax à l'aide du fameux objet XMLHttpRequest que nous avons présenté précédemment. Je rappelle ici que son utilisation passe successivement :
 - on déclare un objet XMLHttpRequest
 - on définit la requête à envoyer avec la fonction open() de l'objet XHR : **c'est ici qu'est définie la partie « chaîne » de la requête qui sera reconnue par le code Arduino comme expliqué précédemment. On utilise ici en paramètre la chaîne reçue par la fonction.**
 - et on envoie la requête au serveur avec la fonction send() de l'objet XHR
 - puis on capture l'évènement onreadystatechange de l'objet XHR et on y associe une fonction où :
 - lorsque sa valeur vaut 4, c'est à dire lorsque le serveur a fini de répondre,
 - et que le serveur a renvoyé une réponse http 200 OK
 - alors on exécute le code de gestion de la réponse reçue
- Comme expliqué juste avant, cette fonction reçoit en paramètre une fonction, appelée callback ici, à laquelle sera passé le résultat de la propriété **responseText** de l'objet XHR et qui correspond à la chaîne reçue en provenance du serveur.

ATTENTION : les explications que je donne ici concerne bien sûr le code Javascript qui est envoyé au client et qui sera exécuté par le client. Le code Arduino lui ne fait qu'envoyer les chaînes du code Javascript au client. En un mot, on envoie du code Javascript à partir du code Arduino.

```
client.println(F("function requeteAjax(chaineIn, callback) { "}); // debut envoi requete avec chaine personnalisee

client.println(F("var xhr = XMLHttpRequest(); "));

client.println(F("xhr.open(\"GET\", chaineIn, true);")); // envoi requete avec chaine personnalisee
client.println(F("xhr.send(null);"));

//----- gestion de l'évènement onreadystatechange -----
client.println(F("xhr.onreadystatechange = function() { "));

client.println(F("if (xhr.readyState == 4 && xhr.status == 200) {"));

client.println(F("    //alert(xhr.responseText);"));
client.println(F("    callback(xhr.responseText);"));

client.println(F("} // fin if "));

client.println(F("}; // fin function onreadystatechange"));
//----- fin gestion de l'évènement onreadystatechange -----

client.println(F("} // fin fonction requeteAjax"));
```

Fonction **loop()** (6) : Head (4) : envoi de la fonction Javascript de gestion des données reçues du serveur et fin de script

- Ensuite on envoie la fameuse fonction de « callback » qui sera appelée une fois la réponse à la requête Ajax reçue : [cette fonction reçoit en paramètre la chaîne reçue du serveur](#).
- Ici, on ne fait rien d'extraordinaire : on ajoute simplement la chaîne reçue à la zone de texte, à la façon Terminal Série du logiciel Arduino, grâce à des fonctions Javascript dont vous trouverez facilement l'explication par ailleurs : [le corps de la page sera modifié en conséquence, sans que la page ne se rafraîchisse par ailleurs rappelons-le](#). Le rafraîchissement pourra toujours être activé manuellement par un clic souris, mais cela entraînera l'effacement de la page et de la zone de texte par nouvelle réception de la page HTML de départ.

ATTENTION (bis) : Encore une fois les explications que je donne ici concerne bien sûr le code Javascript qui est envoyé au client et qui sera exécuté par le client. Le code Arduino lui ne fait qu'envoyer les chaînes du code Javascript au client. En un mot, on envoie du code Javascript à partir du code Arduino.

```
client.println(F("function drawData(stringDataIn) { ");

// ajoute la réponse au champ texte
client.println(F("textarea.value=textarea.value+stringDataIn;")); // ajoute la chaîne au début - décale vers le bas...
client.println(F("textarea.setSelectionRange(textarea.selectionEnd-1,textarea.selectionEnd-1) ;")); // se place à la fin -1 pour
avant saut de ligne

client.println(F("} // fin fonction drawData"));

// -----

client.println(F("//-->"));
client.println(F("</script>"));
client.println(F("<!-- Fin du code Javascript --> "));

//===== fin du bloc de code javascript =====

client.println(F("</head>"));
//----- fin head = fin entete de la page HTML -----
```

Fonction **loop()** (7) : envoi du body et de la fin de la page HTML de réponse

- De la même façon, par un jeu de `println()`, on envoie la suite du code HTML, c'est à dire ici le body de la page HTML.
- Ici, on ajoute :
 - les un bouton simple **en associant l'évènement onclick à la fonction définie précédemment**,
 - **le champ texte pour saisir la chaîne à envoyer vers Arduino**
 - une zone de texte pour afficher les messages de réponse du serveur Arduino,
- Suivent les balises de clôture de la page web

```
//----- body = corps de la page HTML -----
client.println("<body>");

// affiche chaines caractères simples
//client.println(F("<CENTER>")); //pour centrer la suite de la page
//client.println(F("<canvas id=\"nomCanvas\" width=\"300\" height=\"300\"></canvas>"));
//client.println(F("<br/>"));
client.println(F("<button type=\"button\" onclick=\"onclickButtonON()\">Allumer la LED</button>"));
client.println(F("<button type=\"button\" onclick=\"onclickButtonOFF()\">Eteindre la LED</button>"));
client.println(F("<br/>"));
client.println(F("Serveur Arduino : Test envoi requete Ajax par clic Bouton"));
client.println(F("<br/>"));
client.println(F("En provenance du serveur Arduino :"));
client.println(F("<br/>"));
client.println(F("<textarea id=\"textarea\" rows=\"10\" cols=\"50\" > </textarea>")); // ajoute zone texte vide à la page
client.println(F("<br/>"));

client.println(F("</body>"));
//----- fin body = fin corps de la page -----

client.println(F("</html>"));
//----- fin de la page HTML -----

} // fin if GET
```


Fonction **loop()** (8) : Fermeture de la connexion avec le client

- si la requête reçue n'est pas une « GET », un message indique que la requête n'est pas valide.
- Puis la connexion avec le client est clotûrée.

```
        else { // si la chaine recue ne commence pas par GET
          Serial.println (F("Requete HTTP non valide !"));
        } // fin else

//----- fermeture de la connexion -----

// fermeture de la connexion avec le client après envoi réponse
delay(1); // laisse le temps au client de recevoir la réponse
client.stop();
Serial.println(F("----- Fermeture de la connexion avec le client -----")); // affiche le String de la requete
Serial.println (F(""));

} // --- fin if client connected

} //---- fin if client ----

} // fin de la fonction loop()
```

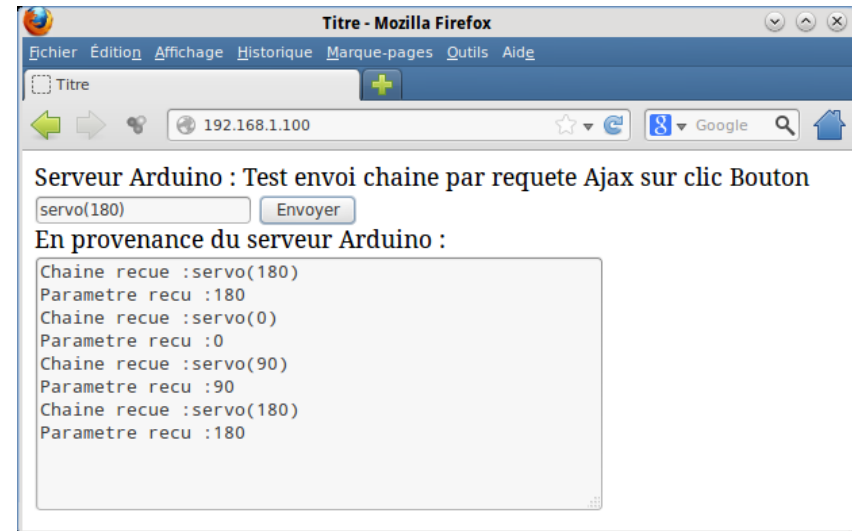
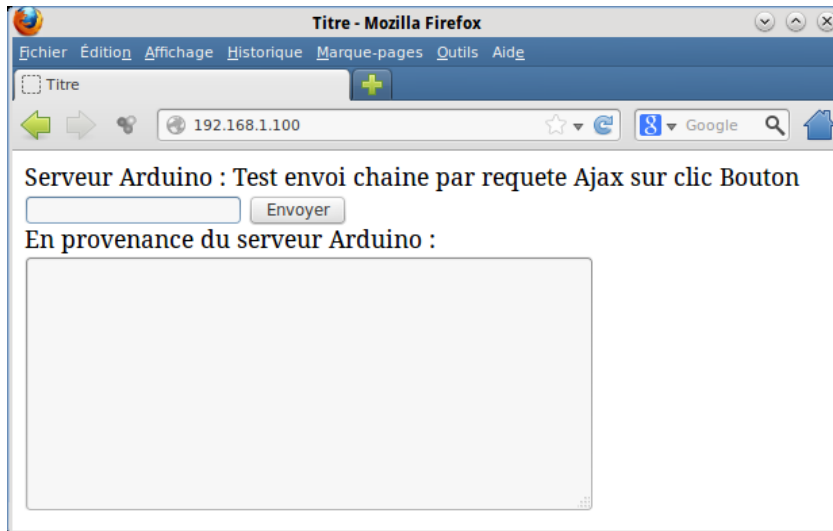
Fonctions communes : Fonction d'envoi de l'entête http

- Dans la mesure où l'on est amené à envoyer plusieurs fois une entête http, autant rassembler le code dans une fonction commune :
 - **HTTP/1.1 200 OK** indique que le serveur a pu traiter la requête
 - Le champ **Content-Type: text/html** indique que la réponse sera du texte ou de l'html (on va voir ça après)
 - Le champ **Connection: close** indique que la connexion doit être fermée après réception de la réponse.
 - Un saut de ligne précède le message de réponse

```
//----- fonctions communes -----  
  
void envoiEnteteHTTP(EthernetClient clientIn){  
  if (clientIn) {  
    //-- envoi de la réponse HTTP ---  
    clientIn.println(F("HTTP/1.1 200 OK")); // entete de la réponse : protocole HTTP 1.1 et exécution requete réussie  
    clientIn.println(F("Content-Type: text/html")); // précise le type de contenu de la réponse qui suit  
    clientIn.println(F("Connection: close")); // précise que la connexion se ferme après la réponse  
    clientIn.println(); // ligne blanche  
  
    //-- envoi en copie de la réponse http sur le port série  
    Serial.println(F("La reponse HTTP suivante est envoyee au client distant :"));  
    Serial.println(F("HTTP/1.1 200 OK"));  
    Serial.println(F("Content-Type: text/html"));  
    Serial.println(F("Connection: close"));  
  
  } // fin si client  
}  
// fin envoiEnteteHTTP
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne un message indiquant l'adresse du serveur (ici 192.168.1.100) et le port d'écoute (ici 80)
- A présent, **ouvrir une fenêtre de navigateur Firefox sur le poste fixe connecté au réseau et saisir l'adresse du shield dans la barre d'adresse** (ici 192.168.1.100). On doit alors voir apparaître dans le Terminal Série toute une série de lignes de texte correspondant à la requête envoyée par le navigateur. Dans le navigateur, on doit ici voir dans la fenêtre Firefox, :
 - le champ de saisie de chaîne et le bouton d'envoi
 - ainsi que la zone texte.
- **Saisissez une chaîne de la forme servo(xxx) où xxx est une valeur d'angle entre 0 et 180 et cliquez sur le bouton : le servomoteur va se positionner en conséquence... le tout par le réseau !**



Cette fois, vous avez les bases pour contrôler n'importe quel dispositif par le réseau à partir d'une interface dans le navigateur client, en passant à la demande un ou plusieurs paramètres numériques, le tout à partir d'un code entièrement écrit côté Arduino et s'exécutant du côté serveur Arduino et du côté client (Javascript)

Synthèse technique

Arrivé au terme de ce tuto, je pense que vous voyez tout l'intérêt de l'utilisation de Javascript et d'AJAX, mais cette fois pour l'envoi de chaînes avec paramètres vers Arduino :

Tout le code actif est intégré côté serveur, autrement dit dans la page HTML+Javascript qu'envoie le serveur lors de la requête initiale : côté client, un simple navigateur suffit et c'est tout ! Aucun paramétrage particulier à faire, aucune installation, etc...

La communication entre le navigateur client et le serveur Arduino se fait en « arrière-plan » grâce à la technologie AJAX, permettant un fonctionnement transparent et très polyvalent, à partir de simples événements « utilisateur » de la page HTML.

La suite ?

Utiliser des widgets graphiques pour passer des valeurs ou contrôler Arduino.
ça sera dans le tuto suivant.

Ensuite, pourquoi ne pas utiliser dans une même page les requêtes automatiques (réception de données) et les requêtes « à la demande » pour contrôler le comportement d'Arduino à distance et recevoir des données « temps-réel » ou stockées dans des fichiers.... ?
à suivre...

21. Les éléments du langage Arduino étudiés dans cet atelier

Les fonctions de la librairie Ethernet

Chaque classe dispose de plusieurs fonctions associées :

Classe *Ethernet* (configuration matérielle du shield Ethernet)

- | begin() | localIP() | maintain()

Classe *EthernetServer* (serveur TCP)

- | begin() | available() | write() | print() | println()

Classe *EthernetClient* (client TCP)

- | connected() | connect() | write() | print() | println() | available() | read() | flush() | stop()

La documentation complète du langage Arduino en français est disponible ici :
http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.ReferenceMaxi

22. *A présent, vous devriez être capable :*

- d'associer un code Javascript à un élément HTML lors de la capture d'un événement provoqué par utilisateur
- d'utiliser la technologie AJAX pour déclencher l'envoi de chaînes vers le serveur à partir du navigateur client
- de contrôler des broches Arduino à partir navigateur client
- de contrôler des dispositifs en envoyant des chaînes avec paramètres numériques à partir du navigateur client

Table des matières

Créer un serveur HTML+ Javascript + Ajax avec Arduino et contrôler Arduino depuis le navigateur client en utilisant des requêtes Ajax.

Intro |

Matériel nécessaire pour les ateliers Arduino |

Matériel spécifique nécessaire pour cet atelier |

Matériel spécifique nécessaire pour cet atelier (suite) |

La structure du réseau que nous allons réaliser |

Monter le réseau utilisant le shield Ethernet Arduino sur un réseau avec « box » existant |

Rappel : Syntaxe de base du langage Javascript |

Rappel : Ecrire un script Javascript intégré dans une page HTML |

Rappel : le DOM, l'accès aux éléments d'une page HTML et les fonctions de l'objet window |

Javascript : Associer une fonction à la survenue d'un événement attaché à un élément du DOM |

HTML + Javascript : associer un événement utilisateur à l'exécution d'un code Javascript |

Rappel : AJAX : principe et intérêt. |

Javascript : L'objet XMLHttpRequest et son utilisation |

Javascript : Code type de gestion d'une requête par XMLHttpRequest |

Petit retour sur la chaîne d'url passée en paramètre lors d'une requête avec l'objet XMLHttpRequest |

Fonction de requête Ajax modifiée pour passer une chaîne texte |

Synthèse : Evènement DOM appelant fonction Javascript envoyant requête AJAX vers Arduino |

Serveur Arduino : Allumer/éteindre une LED par envoi d'une requête Ajax par clic sur un bouton |

Recevoir des chaînes avec paramètres : Installation et présentation de ma librairie Utils |

Serveur Arduino : Contrôler un servomoteur par envoi d'une requête Ajax avec paramètre numérique |

Les éléments du langage Arduino étudiés dans cet atelier |

A présent, vous devriez être capable : |

Bravo !
vous avez terminé cet atelier Arduino !



Prêt pour la suite ? Retrouvez de nombreux autres thèmes d'ateliers Arduino ici :
http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS