

**Créer un serveur HTML+ Javascript + Ajax avec Arduino et contrôler Arduino **graphiquement** ou à l'aide de widgets depuis le navigateur client en utilisant des requêtes Ajax.**



## Ateliers Arduino

par X. HINAULT  
[www.mon-club-elec.fr](http://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](mailto: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](mailto: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](mailto: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 utiliser un canvas pour réaliser un simple slider (widget linéaire réglable)
- d'apprendre à déclencher l'envoi de chaînes vers le serveur par requête Ajax à partir d'un clic souris dans un canvas
- d'apprendre à détecter des événements associés à des widgets graphiques d'une librairie javascript dédiée
- d'apprendre à contrôler des dispositifs à l'aide de widgets graphiques par requête Ajax

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

Cet atelier fait suite à des ateliers 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 :**

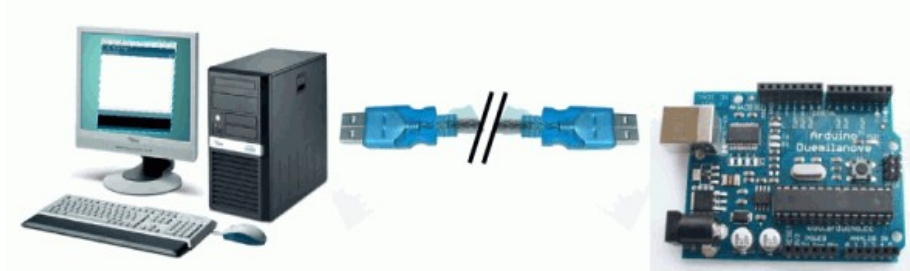
Les codes de cet atelier sont disponibles ici :

<https://github.com/sensor56/0aa90660f5f7bfb6a81311803dbob27f>

## 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

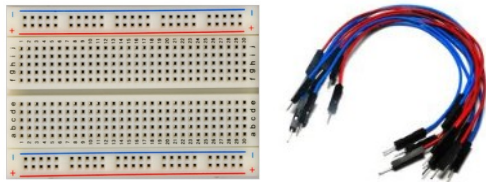


L'espace de développement Arduino associe :

- 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

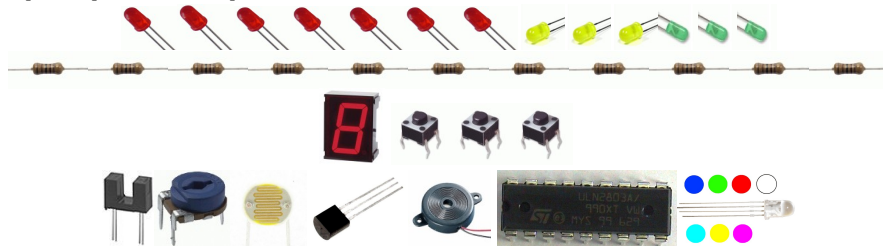


Pour réaliser des montages sans soudure, vous aurez besoin :

- 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](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS)

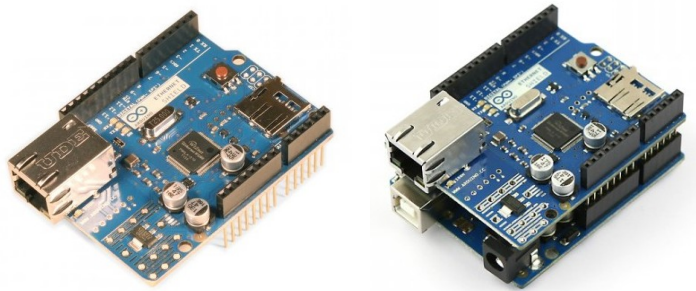
Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- 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.

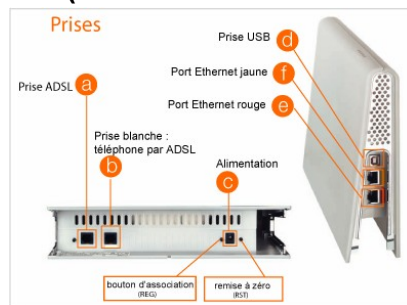
Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 +/- 4 ) pour communiquer avec Arduino.

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. Matériel spécifique nécessaire pour cet atelier (suite)

Nous utiliserons également un servomoteur dans certains des programmes, qui sera contrôlé par le réseau ethernet.

### 1 à 2 Servomoteurs standards



Avec un servomoteur standard, **une seule broche de la carte Arduino suffit pour contrôler le servomoteur, sans aucune autre interface !**

Dispose d'un connecteur type servomoteur dit « mâle » correspondant à l'association de 3 connecteurs femelle :  
broche numérique PWM servo / +5V / 0V

Couple : 3.2kg.cm en 4.8V

Consommation : 100mA env.

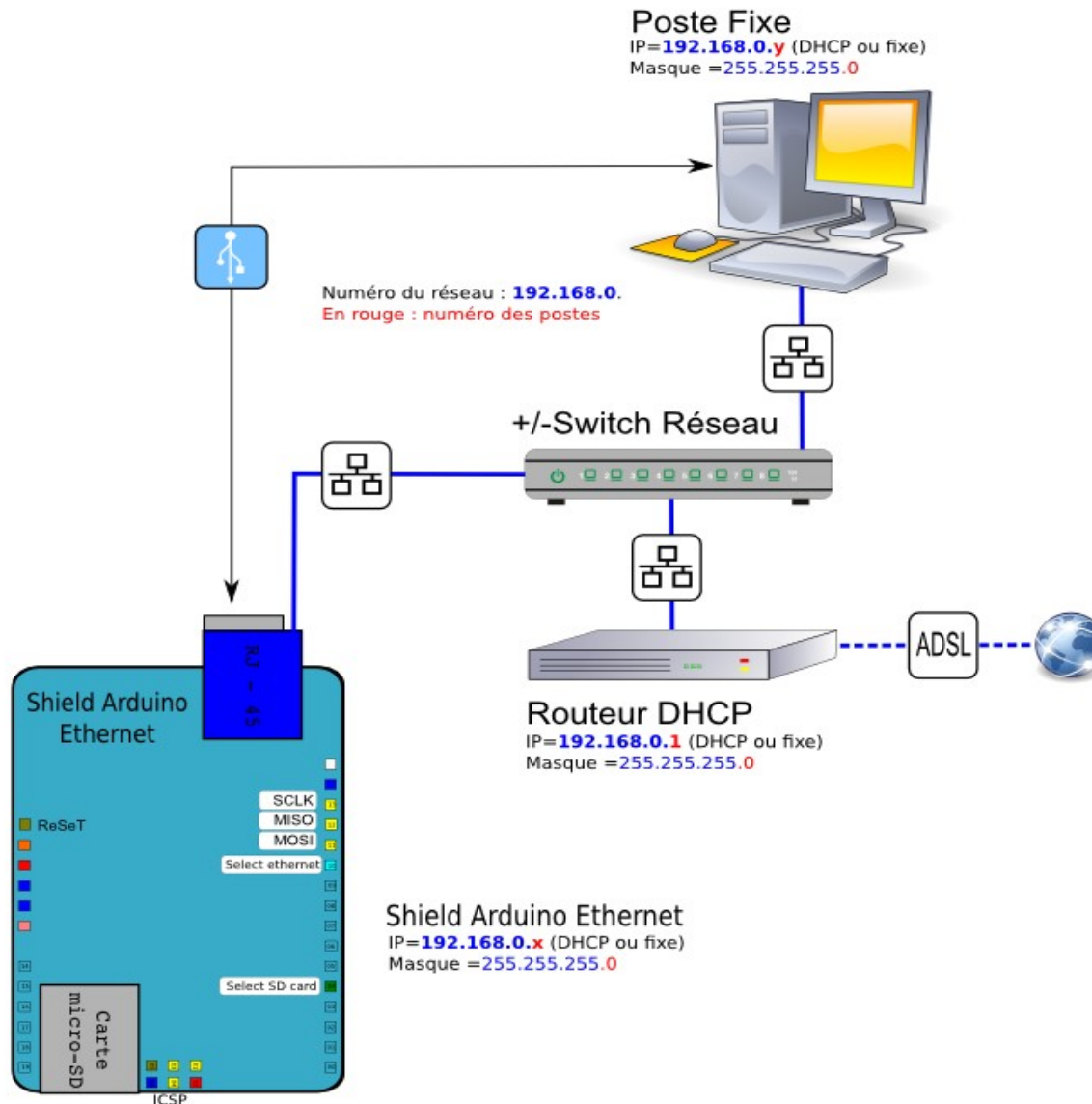
**Alimentable directement par le +5V de la carte Arduino jusqu'à 3 servomoteurs.** Au-delà, avec la carte Arduino, prévoir une alimentation externe régulée 5V (alim PC) ou non (Vin=6V).

Modèles possibles suggérés :

- un Futaba S3003 ici : <http://store.easyrobotics.fr/servomoteur/12-servomoteur-futaba-s3003.html>
- ou tout autre modèle de servomoteur standard...



## 6. 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

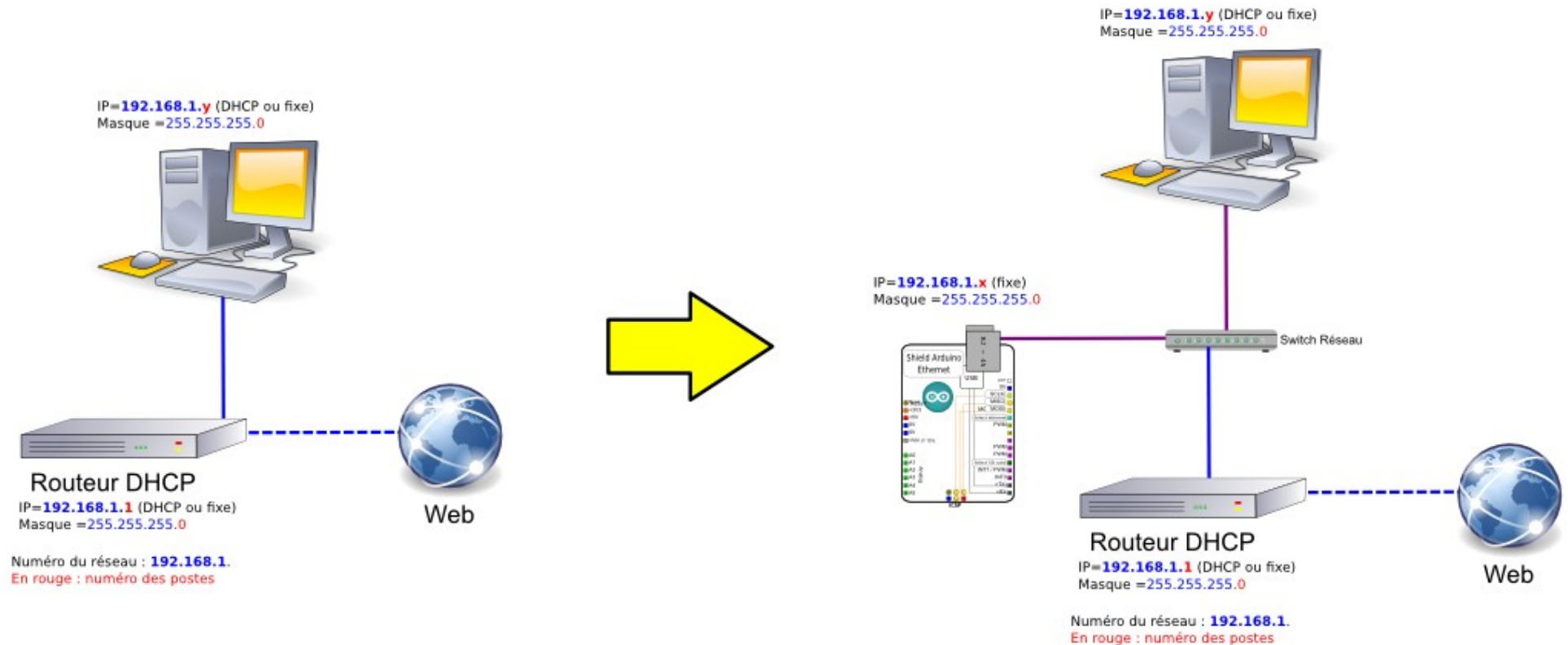
Ceci est très pratique en phase de test et de mise au point, mais une fois la programmation terminée, on pourra bien sûr déconnecter le câble USB.

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

## 7. 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.





## 8. 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://fr.wikipedia.org/wiki/Syntaxe_JavaScript)
  - <http://www.siteduzero.com/informatique/tutoriels/dynamisez-vos-sites-web-avec-javascript>

## 9. 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 !

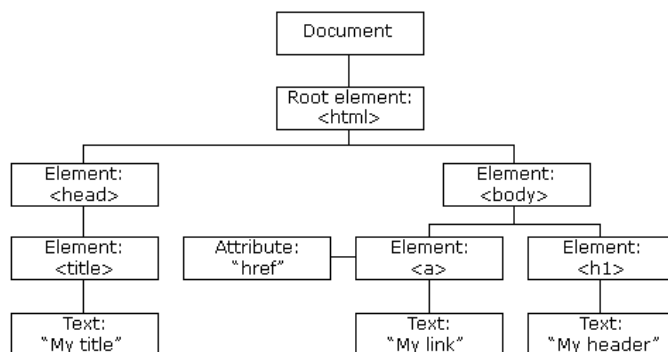
    </body>
    <!-- Fin de corps de page HTML -->

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

## 10. Rappel : Javascript : 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://fr.wikipedia.org/wiki/Document_Object_Model)
  - [http://www.w3schools.com/html/dom/dom\\_intro.asp](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](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](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](http://www.w3schools.com/tags/ref_eventattributes.asp)

## 11. Rappel : 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](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>
```

## 12. Rappel : HTML : Présentation de l'objet canvas

### Présentation

- Il existe de très nombreux objets HTML et il n'est pas question de les passer en revue ici, mais il y en a un qui mérite toute notre attention : le canvas !
- Un canvas est un objet HTML5 particulièrement intéressant : il s'agit d'un objet qui représente une zone de dessin 2D que l'on va pouvoir intégrer au sein d'une page HTML.
- Le très grand intérêt du canvas, c'est qu'il dispose de nombreuses fonctions de dessin qui vont permettre d'y dessiner simplement ce que l'on veut, et donc, dans notre cas, de présenter des données en provenance d'Arduino sous forme graphique dans une page web, ni plus ni moins !

### Balise d'insertion

- La balise HTML permettant d'intégrer un canvas dans une page est tout ce qu'il y a de plus classique :
  - une balise de début `<canvas>` et de fin `</canvas>`
  - des paramètres définissant le canvas, notamment :
    - un identifiant
    - une largeur et une hauteur en pixels
- Ce qui nous donne :

```
<canvas id="cvs" width="300" height="300"></canvas>
```

- Ici, on crée un canvas, autrement dit une zone de dessin :
  - appelée cvs
  - de 300 pixels de large sur 200 pixels de haut
- Difficile de faire plus simple...

### Page HTML utilisant un canvas

```
<!DOCTYPE HTML>

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

  <!-- Debut entete -->
  <head>
    <meta charset="utf-8" /> <!-- Encodage de la page -->
    <title>Test Canva seul</title> <!-- Titre de la page -->
  </head>
  <!-- Fin entete -->

  <!-- Debut Corps de page HTML -->
  <body>
    <canvas id="papier" width="300" height="300"></canvas>
    <br />
    Exemple de Canva
  </body>
  <!-- Fin de corps de page HTML -->

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

### Page HTML + Javascript utilisant un canvas

- L'utilisation la plus utile d'un canvas est de le coupler à du code Javascript qui va permettre de dessiner à l'intérieur, ce qui donne :

```
<!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 -->

    <!-- Debut du code Javascript -->
    <script language="javascript" type="text/javascript">
      <!--
      window.onload = function() {

        var canvas = document.getElementById("cvs"); // declare
        objet canvas a partir nom

        // mettre ici le code de dessin dans le canvas

      } // fin window.onload
      <!-->
    </script>
    <!-- Fin du code Javascript -->

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

  <!-- Debut Corps de page HTML -->
  <body >

    <canvas id="cvs" width="300" height="300"></canvas>
    <!-- IMPORTANT : le canvas doit etre declare AVANT le code qui
    l'utilise ! -->

    <br />
    Exemple de Canva

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

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

### En savoir plus

- Toutes les méthodes et propriétés de l'objet canvas :  
[http://www.w3schools.com/tags/ref\\_canvas.asp](http://www.w3schools.com/tags/ref_canvas.asp)
- Une petite page qui permet facilement de tester les possibilités du canvas :  
<http://jm.davalan.org/lang/jsc/js09.html>

## 13. Rappel : Javascript : Les fonctions essentielles de l'objet canvas

### Déclaration

- La première chose à faire au niveau du code Javascript qui va utiliser le Canvas, c'est de créer l'objet représentant le canvas, ce qui se fait avec la fonction `getElementById()` vue précédemment :

```
var canvas = document.getElementById("nomCanvas"); // declare objet canvas a partir id = nom
```

### Initialisation

- Une fois l'objet Canvas déclaré, on doit avant toute chose l'initialiser à l'aide d'une fonction particulière appelée `getContext()` et qui renvoie un objet Context qui servira pour appeler les fonctions de dessin (le canvas en lui-même n'est qu'un conteneur).
- Cette fonction reçoit en paramètre « 2d » pour signifier le type de dessin qui va être effectué.

**La 3D dans un Canvas est possible et arrive progressivement.**  
Voir ici par exemple : <http://www.canvasdemos.com/type/applications/3d-applications/>

- On a :

```
var ctx = canvas.getContext("2d"); // objet context permettant acces aux fonctions de dessin
```

- En pratique cependant, on teste au préalable le retour de la fonction `getContext()` avant d'appeler les fonctions de dessin, ce qui donne :

```
if (canvas.getContext){ // la fonction getContext() renvoie True si canva accessible

    var ctx = canvas.getContext("2d"); // objet context permettant acces aux fonctions de dessin

    // fonctions de dessin ici
}
else {

    // code si canvas non disponible
}
```

### Les fonctions de dessin de base

- Une fois que l'on dispose de l'objet Context d'accès aux fonctions de dessin du Canvas, on va pouvoir passer à l'action : **en fait, à ce stade, c'est tout un nouveau monde de possibilités qui s'ouvre à vous !** Un peu à la façon processing pour ceux qui connaissent...
- Tout d'abord, on peut modifier les dimensions du Canvas avec les propriétés `.width` et `.height`
- On peut également paramétrer le mode de dessin avec :
  - `fillStyle()` : pour fixer la couleur de remplissage
  - `strokeStyle()` : pour fixer la couleur de pourtour
  - etc...
- On dispose bien sûr des fonctions géométriques de base :
  - `strokeRect()` : pourtour d'un rectangle
  - `fillRect()` : un rectangle plein
  - etc...
- On dispose également d'une possibilité de tracer un élément sous la forme d'un « chemin » de points successifs avec les fonctions :
  - `beginPath()` et `closePath()`
  - `lineTo()`
  - `moveTo()`
  - `arc()` et `arcTo()`
  - etc...
- Pour plus de détails, voir : [http://www.w3schools.com/tags/ref\\_canvas.asp](http://www.w3schools.com/tags/ref_canvas.asp)
- Voici un exemple simple traçant un carré vert :

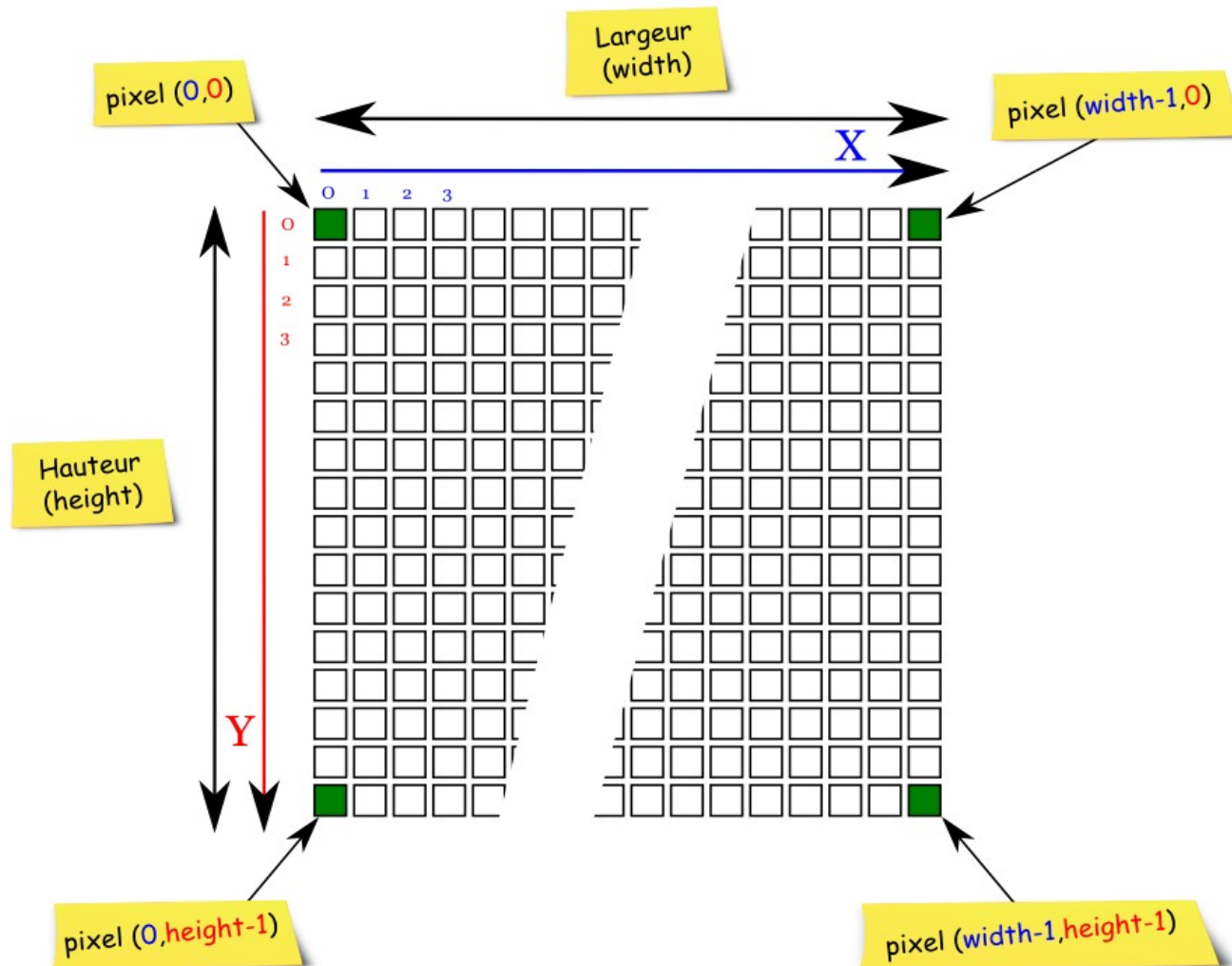
```
var ctx = canvas.getContext("2d"); // objet context permettant acces aux fonctions de dessin
```

```
// le code graphique ci-dessous
ctx.fillStyle = "rgb(0,500,0)"; // couleur remplissage
ctx.fillRect (50, 50, 200, 200); // rectangle plein
```



## 14. Rappel : Objet Canvas : système de coordonnées

- Le système de coordonnées d'un canvas de largeur width et de hauteur height est le suivant :



## 15. HTML + Javascript : Utiliser un canvas comme un « slider »

### Ce que l'on va faire ici :

- Notre objectif ici va être de pouvoir utiliser de façon interactive une interface graphique pour provoquer des événements qui vont tenir compte de la position de la souris au moment d'un clic par exemple.
- Une des solutions les plus simples est de transformer un canvas en « slider » : un « slider » est un élément graphique réglable linéaire, horizontal ou vertical.
- Le principe que nous allons utiliser est le suivant :
  - nous allons déclarer un canvas rectangulaire allongé
  - nous allons y dessiner un rectangle coloré de la taille du canvas (le fond) et par-dessus, un rectangle coloré de longueur variable (la partie « réglable »)
  - nous allons détecter la position de la souris dans le canvas lors d'un clic souris et adapter la taille du rectangle coloré en conséquence : le tour sera joué !

### Le codage HTML

- Au niveau HTML, nous allons insérer dans notre page un simple canvas, pour lequel :
  - nous utiliserons une forme allongée horizontale
  - et nous fixerons un style de positionnement relatif pour pouvoir obtenir la coordonnée du curseur à l'intérieur du canvas

```
<canvas id="nomCanvas" width="300" height="30" style="position: relative;"></canvas>
```

- On déclare par ailleurs les champs texte utilisés :
  - pour afficher les coordonnées de la souris d'une part
  - et pour afficher/éditer les valeurs minimales et maximales du « slider »

```
<br />
X=<input type="text" id="valeurX" />
Y=<input type="text" id="valeurY" />
<br />
Valeur Min=<input type="text" id="valeurMin" size="10"/>
Valeur Max=<input type="text" id="valeurMax" size="10"/>
<br />
Valeur Courante=<input type="text" id="valeur" size="10"/>
```

## Le code Javascript de la fonction initiale

- Au niveau du code Javascript, plusieurs choses vont être nécessaires.
- Tout d'abord, au niveau de la fonction initiale exécutée au chargement de la page :
  - nous allons dessiner nos 2 rectangles de fond et d'avant, le rectangle d'avant étant à 50 % de la longueur du canvas (choix arbitraire)
  - et nous allons, et **c'est là un point clé de ce code**, ajouter « l'écoute » de l'événement click et l'associer à l'appel d'une fonction appelée mouseClicked. **Il faut bien comprendre que l'objet événement sera passé en paramètre à ladite fonction :**

```
if (canvas.getContext){ // la fonction getContext() renvoie True si canvas accessible

    contextCanvas = canvas.getContext("2d"); // objet context global permettant acces aux fonctions de dessin

    // le code graphique ci-dessous

    // carre gris de la taille du canvas
    contextCanvas.fillStyle = "rgb(200,200,200)"; // couleur de remplissage rgb 0-255
    contextCanvas.fillRect (0, 0, canvas.width, canvas.height); // rectangle

    // carre vert
    contextCanvas.fillStyle = "rgb(0,255,0)"; // couleur remplissage
    contextCanvas.fillRect (0, 0, canvas.width/2, canvas.height); // rectangle

    canvas.addEventListener('click',mouseClick,false); // active la capture d'evenement - appelle la fonction mouseMove et
lui passe l'evenement

} // fin si canvas existe
```

## Code Javascript : Fonction de gestion de l'évènement « clic souris »

- Ensuite, nous définissons la fonction de gestion de l'évènement appelée lors de la survenue d'un clic souris : c'est la fonction utilisée avec `addEventListener()`.
- Cette fonction va recevoir l'objet événement, ici appelé `e`, en paramètre ce qui va permettre de récupérer les coordonnées de la souris avec les propriétés `e.layerX` et `e.LayerY` dans Firefox (le nom de cette propriété est variable selon les navigateurs, à adapter au besoin... )
- Ensuite, on utilise ces valeurs comme on le souhaite :
  - nous redessinons nos 2 rectangles du canvas, le rectangle avant ayant une longueur correspondant à la valeur du X souris, ce qui donne un effet de « slider »,
  - et nous « mappons » la valeur de X pour obtenir la correspondance de la valeur X du clic souris avec la valeur réelle fixée par les champs Min et Max.

```
function mouseClicked(e) { // fonction recoit evenement obtenu par addEventListener

    var mouseX, mouseY;

    // firefox - recupere les coordonnees de la souris dans le canvas
    mouseX = e.layerX;
    mouseY = e.layerY;

    textInputX.value=mouseX; // affiche valeur dans le champ texte
    textInputY.value=mouseY; // affiche valeur dans le champ texte

    // carre gris de la taille du canvas
    contextCanvas.fillStyle = "rgb(200,200,200)"; // couleur de remplissage rgb 0-255
    contextCanvas.fillRect (0, 0, canvas.width, canvas.height); // rectangle

    contextCanvas.fillStyle = "rgb(0,255,0)"; // couleur remplissage
    contextCanvas.fillRect (0, 0, mouseX, canvas.height); // rectangle

    // map valeur mouseX en valeur reelle
    textInputValeur.value=Number(textInputMin.value)+Math.round((Number(textInputMax.value)-
Number(textInputMin.value))*mouseX/canvas.width); // affiche valeur dans le champ texte

} // fin mouseClicked
```

## 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 Canva </title> <!-- Titre de la page -->

        <!-- Debut du code Javascript -->
        <script language="javascript" type="text/javascript">
            <!--

                // code javascript par X. HINAULT - www.mon-club-elec.fr - fev 2013 - tous droits reserves - GPL v3

                // variables / objets globaux - a declarer avant les fonctions pour eviter problemes de portee
                var canvas= null; // pour objet Canvas
                var contextCanvas = null; // pour objet context Canvas
                var textInputX=null;
                var textInputY=null;
                var textInputMin=null;
                var textInputMax=null;
                var textInputValeur=null;

                window.onload = function() {

                    canvas = document.getElementById("nomCanvas"); // declare objet canvas a partir id = nom

                    textInputX= document.getElementById("valeurX"); // declare objet champ text a partir id = nom
                    textInputY= document.getElementById("valeurY"); // declare objet champ text a partir id = nom
                    textInputMin= document.getElementById("valeurMin"); // declare objet champ text a partir id = nom
                    textInputMax= document.getElementById("valeurMax"); // declare objet champ text a partir id = nom
                    textInputValeur= document.getElementById("valeur"); // declare objet champ text a partir id = nom

                    canvas.width = 500; // largeur canvas
                    canvas.height = 20; // hauteur canvas

                    textInputMin.value="0"; // valeur minimale par défaut
                    textInputMax.value="100"; // valeur maximale par défaut
                    textInputValeur.value=(Number(textInputMax.value)-Number(textInputMin.value))/2; // valeur courante par default

                    if (canvas.getContext){ // la fonction getContext() renvoie True si canvas accessible

                        contextCanvas = canvas.getContext("2d"); // objet context global permettant acces aux fonctions de dessin

                        // rectangle gris de la taille du canvas
                        contextCanvas.fillStyle = "rgb(200,200,200)"; // couleur de remplissage rgb 0-255
                        contextCanvas.fillRect (0, 0, canvas.width, canvas.height); // rectangle

                        // rectangle avant
                        contextCanvas.fillStyle = "rgb(0,255,0)"; // couleur remplissage
                        contextCanvas.fillRect (0, 0, canvas.width/2, canvas.height); // rectangle

                        canvas.addEventListener('click',mouseClick,false); // active la capture d'evenement - appelle la fonction mouseMove et lui passe
                    }
                }
            <!--
        </script>
    </head>
</html>
```

Purchased by Franck Ourion, franck.ourion@univ-lorraine.fr #6280170

```

        } // fin si canvas existe

        else {
            alert("Canvas non disponible");// code si canvas non disponible
        } // fin else

    } // fin window.onload

    function mouseClicked(e) { // fonction recoit evenement obtenu par addEventListener

        var mouseX, mouseY;

        // firefox - recupere les coordonnees de la souris dans le canvas
        mouseX = e.layerX;
        mouseY = e.layerY;

        textInputX.value=mouseX; // affiche valeur dans le champ texte
        textInputY.value=mouseY; // affiche valeur dans le champ texte

        // rectangle gris de la taille du canvas
        contextCanvas.fillStyle = "rgb(200,200,200)"; // couleur de remplissage rgb 0-255
        contextCanvas.fillRect (0, 0, canvas.width, canvas.height); // rectangle

        // rectangle avant
        contextCanvas.fillStyle = "rgb(0,255,0)"; // couleur remplissage
        contextCanvas.fillRect (0, 0, mouseX, canvas.height); // rectangle

        // map valeur mouseX en valeur reelle
        textInputValeur.value=Number(textInputMin.value)+Math.round((Number(textInputMax.value)-Number(textInputMin.value))*mouseX/canvas.width); // affiche valeur
        dans le champ texte

    } // fin mouseClicked
    //-->
</script>
<!-- Fin du code Javascript -->
</head>
<!-- Fin entete -->

<!-- Debut Corps de page HTML -->
<body >
    <canvas id="nomCanvas" width="300" height="30" style="position: relative;"></canvas>
    <!-- style : position relative pour x-y souris ok -->
    <!-- IMPORTANT : le canvas doit etre declare AVANT le code qui l'utilise ! -->
    <br />
    X=<input type="text" id="valeurX" />
    Y=<input type="text" id="valeurY" />
    <br />
    Valeur Min=<input type="text" id="valeurMin" size="10"/>
    Valeur Max=<input type="text" id="valeurMax" size="10"/>
    <br />
    Valeur Courante=<input type="text" id="valeur" size="10"/>
    <br />
    Exemple de slider avec un canvas
</body>
<!-- Fin de corps de page HTML -->

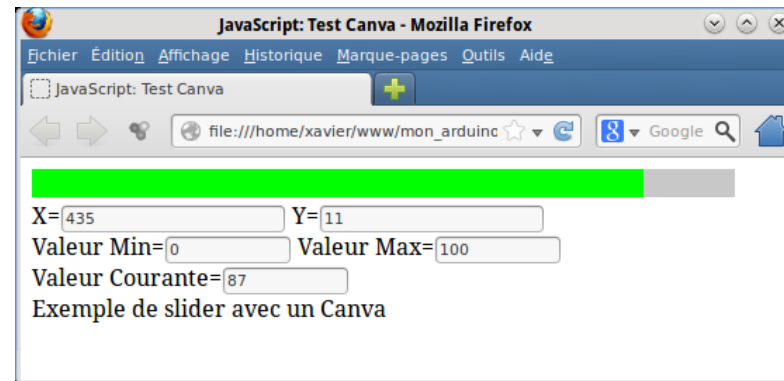
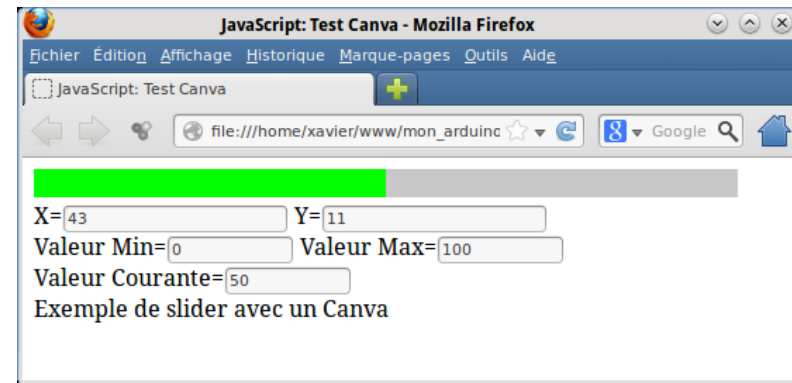
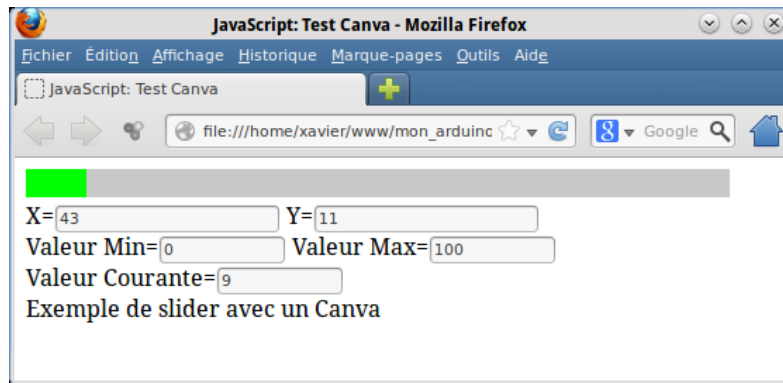
</html>
<!-- Fin de la page HTML -->

```



## Résultat

- Ce code permet de réaliser assez simplement un slider graphique basique dans un canvas, qui se positionnera à l'emplacement voulu lors d'un clic souris :



Un petit code Javascript intéressant qui montre comment capturer un événement souris et comment obtenir les coordonnées de la souris lors d'un clic !  
Le canvas devient un slider certes basique, mais opérationnel, facilement configurable graphiquement,  
et surtout qui **sera utilisable directement depuis le serveur Arduino sans librairie externe.**

**A présent, une fois ces bases posées,  
voyons comment utiliser ce slider pour envoyer une requête Ajax avec paramètre numérique lors d'un clic souris...**

## 16. Rappel : AJAX : principe et intérêt.

## Intro

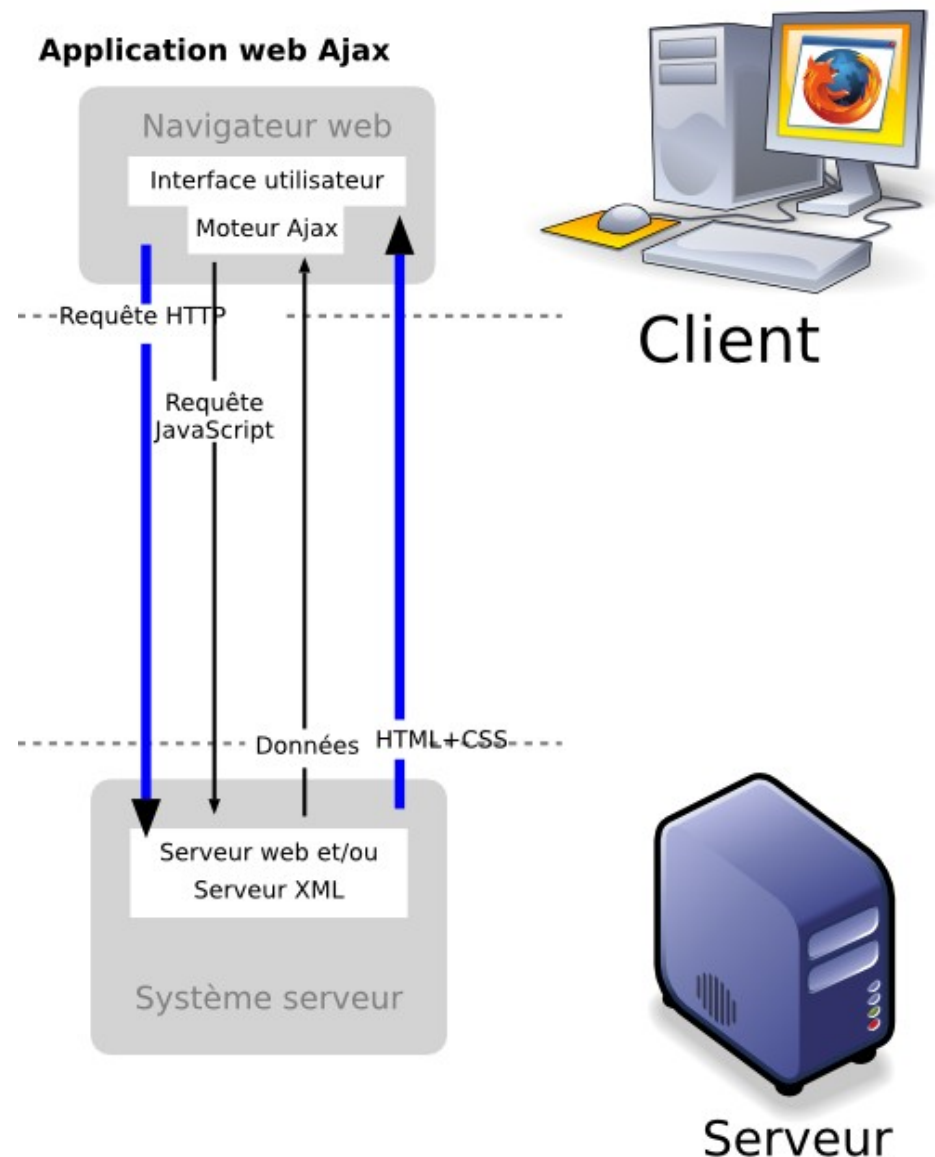
- Dans un tuto précédent, souvenez-vous, nous avons réussi à obtenir un affichage graphique « dynamique » de notre page HTML + Javascript en utilisant la possibilité d'auto-rafraîchissement de la page HTML en ajoutant dans le head une ligne de la forme :

```
<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.



## 17. Rappel : 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](http://www.w3schools.com/xml/xml_http.asp)

## 18. Rappel : 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 function onreadystatechange

} // fin fonction requeteAjax
```

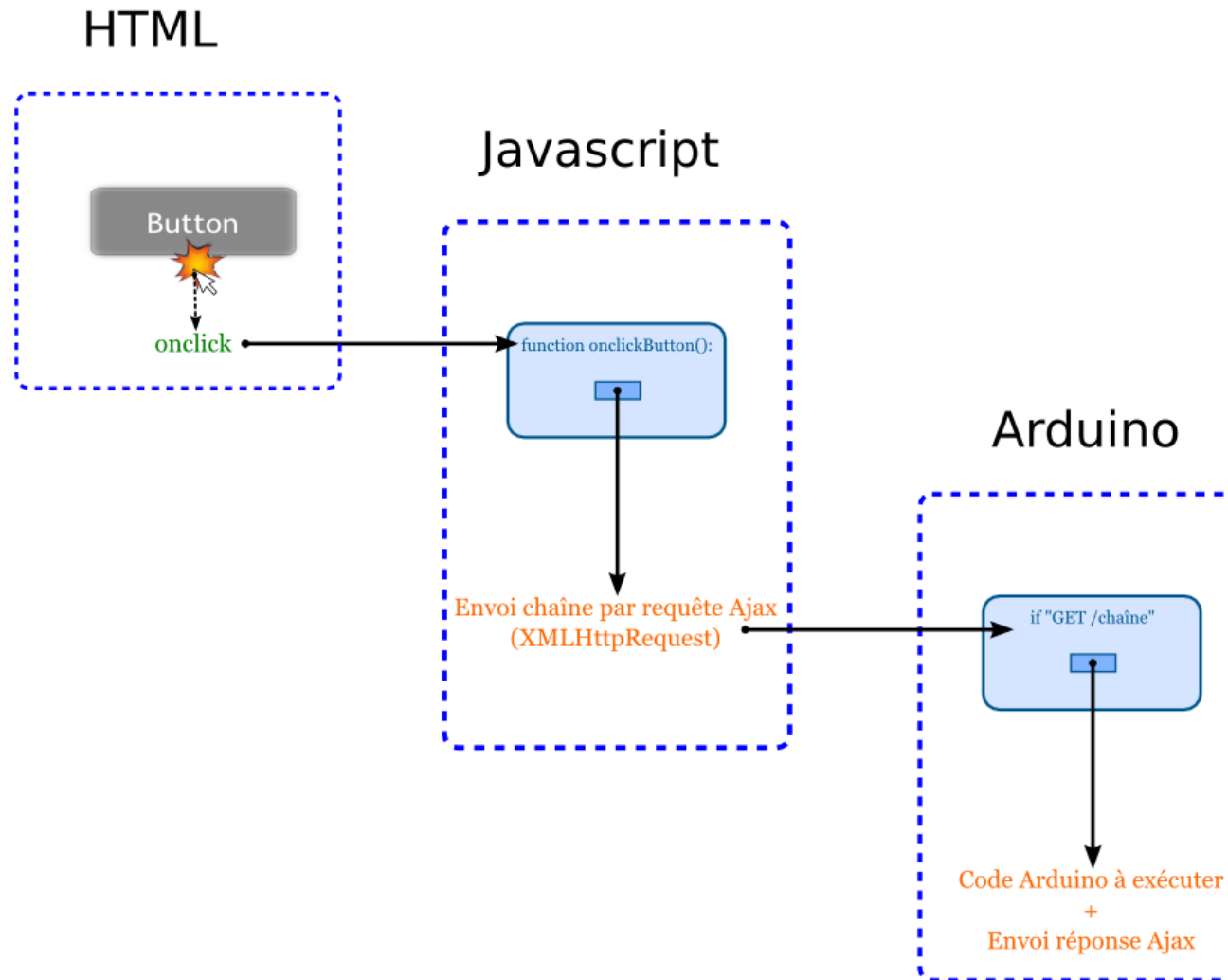
- Noter qu'ici, on maintient la réception d'une réponse du serveur qui devra donc au minimum renvoyer une entête http 200 OK, éventuellement suivie d'une chaîne de réponse qui pourra être utilisée par le code Javascript au besoin.



Nous avons déjà vu cette fonction dans un tuto précédent... donc normalement vous ne devriez pas être trop perdu ici.

## 19. 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



## 20. Arduino : 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

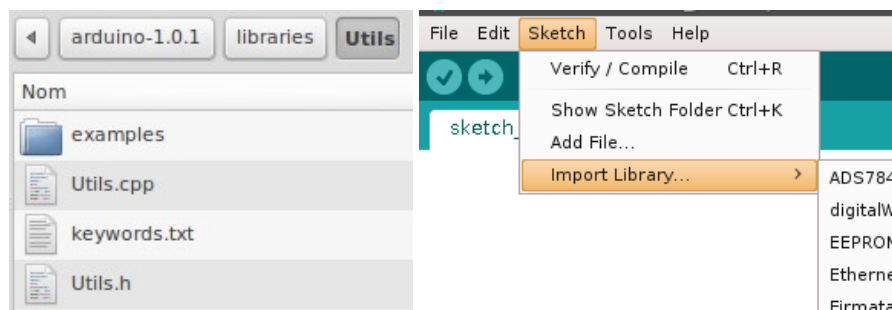
- Ma librairie Utils est disponible ici : [http://www.mon-club-elec.fr/pmwiki\\_reference\\_lib\\_arduino\\_perso/pmwiki.php?n=Main.HomePage](http://www.mon-club-elec.fr/pmwiki_reference_lib_arduino_perso/pmwiki.php?n=Main.HomePage)

### Documentation de la librairie

- [http://www.mon-club-elec.fr/pmwiki\\_reference\\_lib\\_arduino\\_perso/pmwiki.php?n=Main.HomePage](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 côté 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
```

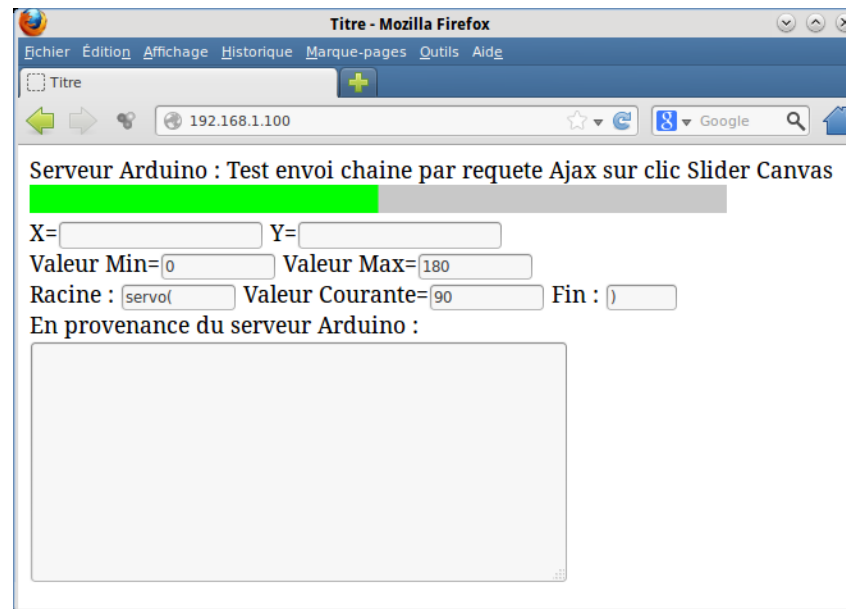


## 21. Serveur Arduino : Contrôler un servomoteur par envoi d'une requête Ajax avec paramètre numérique sur un clic souris dans un canvas utilisé en « slider »

Ce qu'on va faire ici...

- Ici, nous allons :
  - lors d'un clic sur canvas utilisé en slider**, déclencher l'envoi d'une requête AJAX intégrant une chaîne « racine » saisie dans un champ texte (ici « **servo()** »), suivie d'une valeur numérique correspondant à la valeur courante du slider et d'un caractère de fermeture (ici « **)** »).
  - 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.

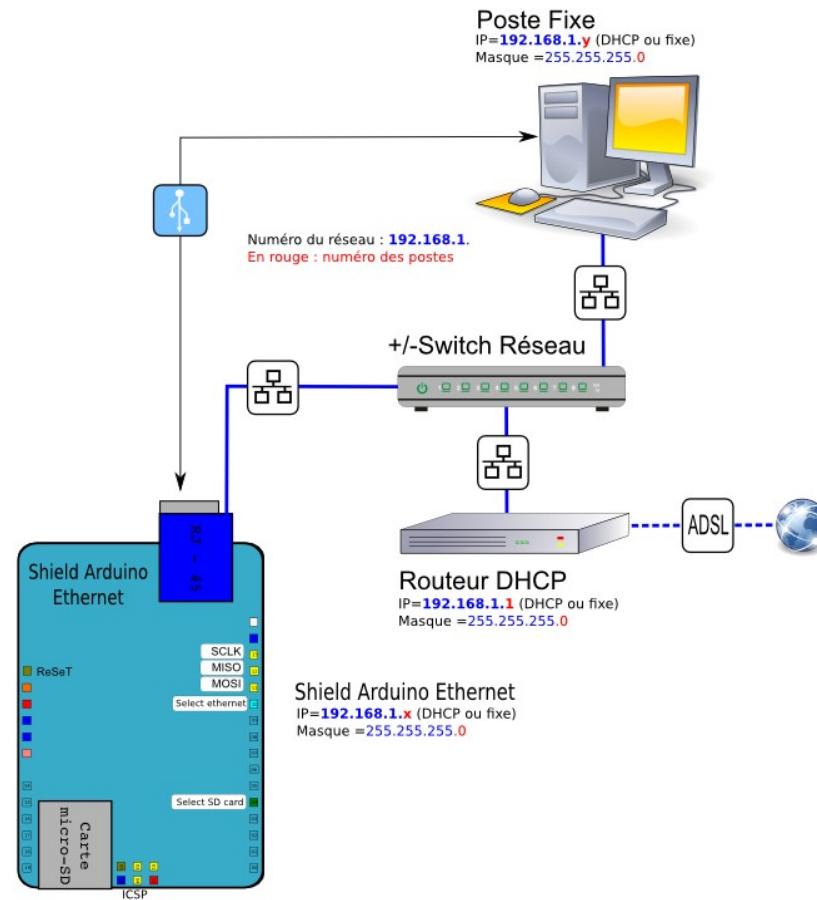


L'interface que nous allons mettre en place ici...

Une véritable « mini-application » graphique contrôlant Arduino via le réseau depuis le navigateur client !

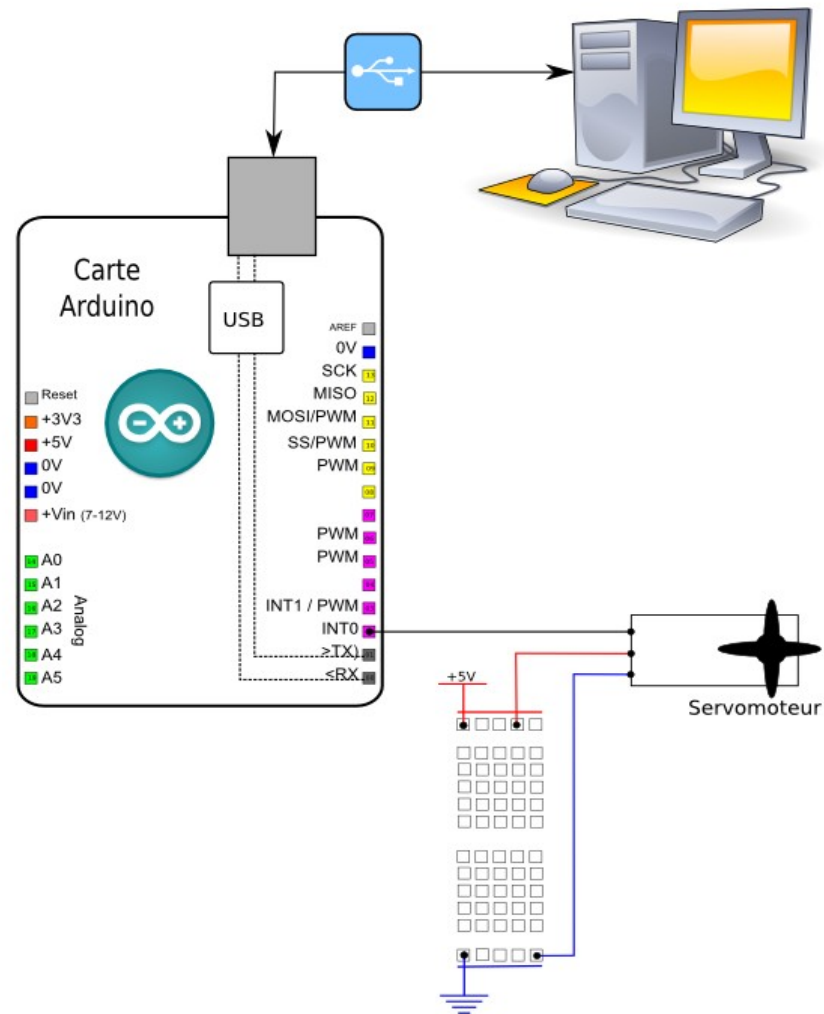
## Le schéma du réseau utilisé

- 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
```

## 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

```
} // fin if GET /&

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, l'objet zone de texte et les objets champ texte utilisés,
  - l'objet canvas et l'objet context associé

```
//===== 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 canvas= null;"));
client.println(F("var contextCanvas = null;"));

client.println(F("var textInputValeur=null;"));

client.println(F("var textInputX=null;"));
client.println(F("var textInputY=null;"));
client.println(F("var textInputMin=null;"));
client.println(F("var textInputMax=null;"));
client.println(F("var textInputValeur=null;"));

client.println(F(""));
```

## Fonction **loop()** : Head : Envoi de la fonction appelée au chargement de la page HTML (1)

- Ensuite, on inclut la fonction appelée lors du chargement initial de la page HTML, sur l'évènement `window.onload` :
  - dans un premier temps, on déclare et initialise les différents objets utilisés sur la page HTML
  - ... de façon à ce que ces 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("canvas = document.getElementById(\"nomCanvas\");"));   
client.println(F("canvas.width = 500; // largeur canvas"));   
client.println(F("canvas.height = 20; // hauteur canvas"));   
  
//----- zone texte ----  
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("textInputX= document.getElementById(\"valeurX\");"));   
client.println(F("textInputY= document.getElementById(\"valeurY\");"));   
client.println(F("textInputMin= document.getElementById(\"valeurMin\");"));   
client.println(F("textInputMax= document.getElementById(\"valeurMax\");"));   
client.println(F("textInputValeur= document.getElementById(\"valeur\");"));   
client.println(F("textInputRacine= document.getElementById(\"racine\");"));   
client.println(F("textInputRacineFin= document.getElementById(\"racineFin\");"));   
  
client.println(F("textInputMin.value=\"0\";")); // posMin en degrés   
client.println(F("textInputMax.value=\"180\";")); // posMax en degrés   
client.println(F("textInputValeur.value=(Number(textInputMax.value)-Number(textInputMin.value))/2; "));   
client.println(F("textInputRacine.value=\"servo\";")); // racine debut   
client.println(F("textInputRacineFin.value=\"\";")); // racine fin
```

## Fonction **loop()** : Head : Envoi de la fonction appelée au chargement de la page HTML (1)

- Toujours au sein de la fonction appelée lors du chargement initial de la page HTML, sur l'évènement window.onload :
  - on réalise le **dessin initial du canvas en slider**, positionné à 50 %
  - **et, point crucial ici, on associe l'écoute de l'évènement click (=clie souris) du canvas à une fonction qui sera appelée lors d'un clic sur le canvas.** Bien comprendre ici que la **fonction appelée recevra en paramètre un objet événement** qui donnera accès à la position du curseur :

```
client.println(F("if (canvas.getContext){");

    client.println(F("contextCanvas = canvas.getContext(\"2d\"); "));

    // rect gris de la taille du canvas
    client.println(F("contextCanvas.fillStyle = \"rgb(200,200,200)\"; "));
    client.println(F("contextCanvas.fillRect (0, 0, canvas.width, canvas.height);"));

    // rect vert de la taille moitié du canvas
    client.println(F("contextCanvas.fillStyle = \"rgb(0,255,0)\";"));
    client.println(F("contextCanvas.fillRect (0, 0, canvas.width/2, canvas.height);"));

    client.println(F("canvas.addEventListener('click',mouseClick,false); "));

    client.println(F("} ")); // fin si canvas existe

    client.println(F("else {"));
    client.println(F("alert(\"Canvas non disponible\");"));
    client.println(F("} ")); // fin else

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



## Fonction **loop()** : Head : 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()** : Head : envoi de la fonction Javascript de gestion des données reçues du serveur

- 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.

```
client.println(F("function drawData(stringDataIn) { ");  
  
    // ajoute la réponse au champ texte  
    client.println(F("textarea.value=textarea.value+stringDataIn;")); // ajoute la chaine 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"));  
  
    // -----
```

## Fonction **loop()** (4) : Head (2) : Les fonctions associées aux évènements des éléments HTML et fin de script

- Ensuite, nous insérons la fonction associée au clic sur le canvas. Plusieurs choses sont réalisées :
  - tout d'abord récupération des coordonnées de la souris dans le canvas au moment du clic, mise à jour des champs texte,
  - mise à jour du dessin du slider en se basant sur la valeur X de la souris,
  - ensuite calcul de la valeur correspondante réelle et mise à jour des champs textes
  - à envoyer la chaîne contenue dans le champ texte racine, la valeur et la fin de la chaîne, 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 ) :

```
//--- fonction de gestion du click souris sur le canvas ---
client.println(F("function mouseClicked(e) { "); // fonction recoit evenement obtenu par addEventListener

    client.println(F("var mouseX, mouseY;"));

    // firefox - recupere les coordonnees de la souris dans le canvas
    client.println(F("mouseX = e.layerX;"));
    client.println(F("mouseY = e.layerY;"));

    client.println(F("textInputX.value=mouseX; "));
    client.println(F("textInputY.value=mouseY;"));

    client.println(F("contextCanvas.fillStyle = \"rgb(200,200,200)\";"));
    client.println(F("contextCanvas.fillRect (0, 0, canvas.width, canvas.height);"));
    client.println(F("contextCanvas.fillStyle = \"rgb(0,255,0)\";"));
    client.println(F("contextCanvas.fillRect (0, 0, mouseX, canvas.height);"));

    // map valeur mouseX en valeur reelle
    client.println(F("textInputValeur.value=Number(textInputMin.value)+Math.round((Number(textInputMax.value)-
Number(textInputMin.value))*mouseX/canvas.width);"));
    client.println(F(""));

    // envoi de la requete Ajax
    client.println(F("requeteAjax(\"&\"+textInputRacine.value+textInputValeur.value+textInputRacineFin.value+\"=\"\", drawData);")); //
envoi requete avec &chaîne= et fonction de gestion resultat
    // utilise les champs racine et racineFin pour encadrer la valeur
    client.println(F("}")); // mouseClicked
    // -----

    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 tous les éléments de la page HTML utilisés, à savoir, le canvas, plusieurs champs textes utiles ainsi que la zone de texte utilisée pour afficher les valeurs en provenance du serveur Arduino,
- Suivent les balises de clôture de la page web

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

client.println(F("Serveur Arduino : Test envoi chaine par requete Ajax sur clic Slider Canvas"));
client.println(F("<br/>"));
client.println(F(""));
client.println(F("<canvas id=\"nomCanvas\" width=\"300\" height=\"30\" style=\"position: relative;\"></canvas>"));
client.println(F("<br />"));
client.println(F("X=<input type=\"text\" id=\"valeurX\" />"));
client.println(F("Y=<input type=\"text\" id=\"valeurY\" />"));

client.println(F("<br />"));
client.println(F("Valeur Min=<input type=\"text\" id=\"valeurMin\" size=\"10\"/>"));
client.println(F("Valeur Max=<input type=\"text\" id=\"valeurMax\" size=\"10\"/>"));

client.println(F("<br />"));
client.println(F("Racine : <input type=\"text\" id=\"racine\" size=\"10\"/>"));
client.println(F("Valeur Courante=<input type=\"text\" id=\"valeur\" size=\"10\"/>"));
client.println(F("Fin : <input type=\"text\" id=\"racineFin\" size=\"5\"/>"));

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 clôturée.

```
        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()
```

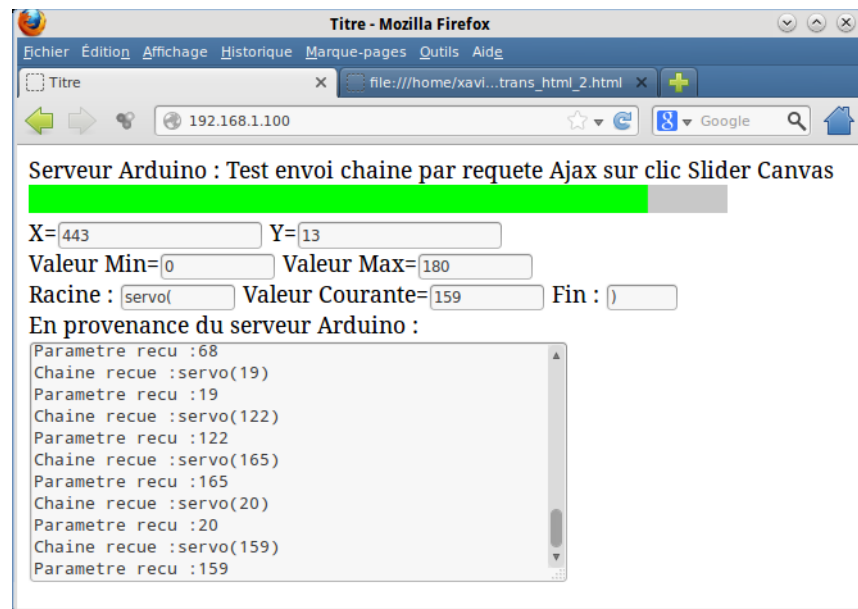
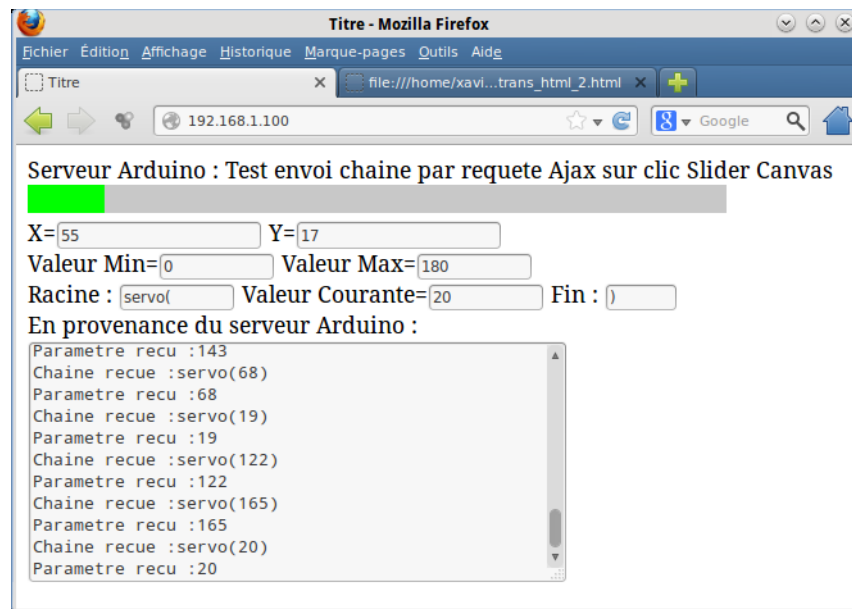
## Fonctions communes : Fonction d'envoi de l'entête http

- 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 racine de la forme `servo(` (et un caractère de fin `)` » de façon à former une chaîne `servo(yyy)` où yyy est une valeur d'angle entre 0 et 180 et cliquez sur le canvas : **le slider est redessiné en conséquence et le servomoteur va se positionner à l'angle voulu... le tout par le réseau !**



**Une fois encore, vous avez les bases pour contrôler graphiquement 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 par simple clic souris, le tout à partir d'un code entièrement écrit côté Arduino et s'exécutant du côté serveur Arduino et du côté client (Javascript)**

La suite ? Faire la même chose mais en utilisant des widgets graphiques plus élaborés à l'aide d'une librairie externe...  
Motivé ? Allez, on continue !



## 22. Rappel : Utilisation d'une librairie Javascript de dessin de widgets analogiques

### Intro

- J'ai pris le temps de détailler par ailleurs l'utilisation d'un fichier javascript externe, car cela va nous être utile à présent.
- Comme vous l'avez vu, il est possible de dessiner facilement dans un canvas... et il est tout à fait possible d'envisager d'écrire soit même le code de ses interfaces graphiques.
- Il existe d'ailleurs plusieurs exemples en ligne tout à fait intéressants à tester et à travailler à votre convenance. Voir ici notamment :
  - <http://geeksretreat.wordpress.com/2012/04/13/making-a-speedometer-using-html5-canvas/>
  - <http://www.splashnology.com/article/how-to-create-a-progress-bar-with-html5-canvas/478/>



### Une librairie à découvrir absolument : Rgraph !

- Si comme moi, vous faites quelques essais, vous verrez que les choses se compliquent vite lorsque l'on code de zéro des widgets analogiques dynamiques en javascript. La vague impression de réinventer la roue...
- C'est pourquoi je vous propose ici de découvrir et tester une librairie absolument incroyable qui propose [des dizaines de possibilités d'affichages analogiques](#) ou graphiques facile à paramétrer en toute simplicité et à afficher dans des... canvas !
- J'ai nommé Rgraph : <http://www.rgraph.net/>
- **La librairie complète est disponible au téléchargement, et est libre d'usage pour une utilisation personnelle ou éducative.**
- Des licences payantes sont proposées pour d'autres usages, commerciaux ou institutionnels, mais cela ne nous concerne pas ici et nous allons pouvoir passer à l'action très simplement.



Purchased by Franck Ourion, franck.ourion@univ-lorraine.fr #6280170

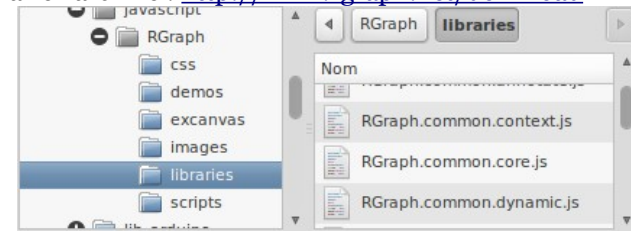
Atelier Arduino : Créer un serveur HTML+Javascript avec Arduino et contrôler Arduino **graphiquement** depuis le navigateur client en utilisant des requêtes Ajax. p.47/78

### Principe général d'utilisation

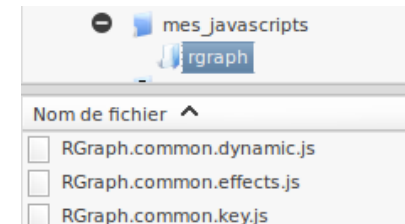
- On commence par installer une fois pour toutes les fichiers javascript de la librairie rgraph dans un répertoire d'un serveur accessible depuis le réseau où se trouve Arduino : ces fichiers contiennent les codes de dessin des éléments graphiques dans un canvas
- Au niveau de la page HTML :
  - on charge les fichiers \*.js nécessaires de la librairie, généralement entre 2 et 3, mais ça peut aller jusqu'à 10 ou plus selon.
  - ensuite, on insère un code javascript réduit dans la page HTML, ce qui va permettre de créer les éléments graphiques voulus et de les afficher avec les effets voulus.

### Installer la librairie RGraph

- La première chose à faire est donc d'installer la librairie RGraph sur un serveur http opérationnel. On commence par télécharger et extraire l'archive : <http://www.rgraph.net/download>



- On obtient un répertoire avec plusieurs sous-répertoires : repérer le **répertoire libraries** qui contient tous les fichiers javascript de la librairie.
- Ce sont ces fichiers qu'il faut **copier sur un serveur http opérationnel**. Le serveur peut-être un poste fixe de votre réseau avec un serveur Apache opérationnel (serveur LAMP sous Gnu/Linux), votre site perso, etc... ou même un raspberryPi.
- Au niveau du serveur choisi, se connecter par FTP, créer un répertoire appelé **rgraph**, lui-même éventuellement placé dans répertoire dédié aux fichiers javascripts et y copier tous les fichiers \*.js du répertoire **libraries** précédent :

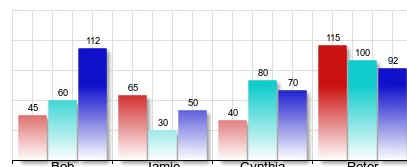
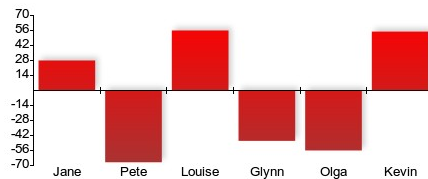
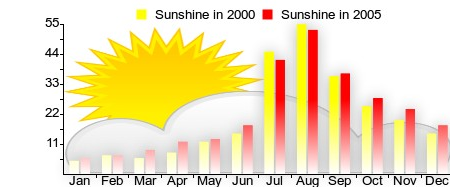


## 23. Vue d'ensemble des graphiques et éléments analogiques disponibles avec la librairie Javascript utilisée

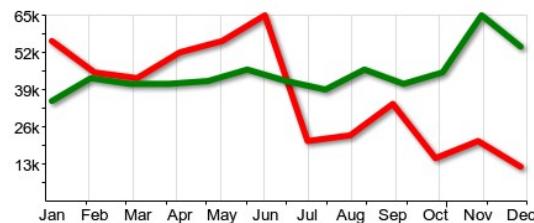
### Intro

- Les types de graphiques et d'effets visuels disponibles avec la librairie Rgraph, affichables dans un simple canvas, sont très variés et très intéressants en ce qui nous concerne, c'est à dire dans le cas d'un serveur Arduino.
- Les effets visuels disponibles sont également bluffant au vu de la simplicité de mise en œuvre.
- Voici un petit panorama en images, à partir d'exemples que vous retrouverez ici : <http://www.rgraph.net/examples/index.html>

### Histogrammes



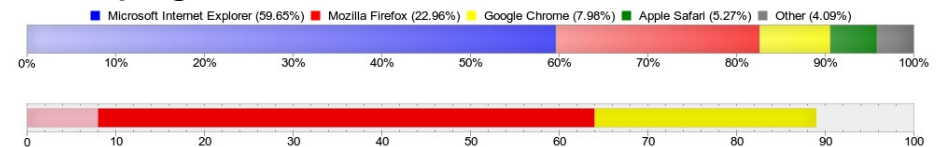
### Courbes



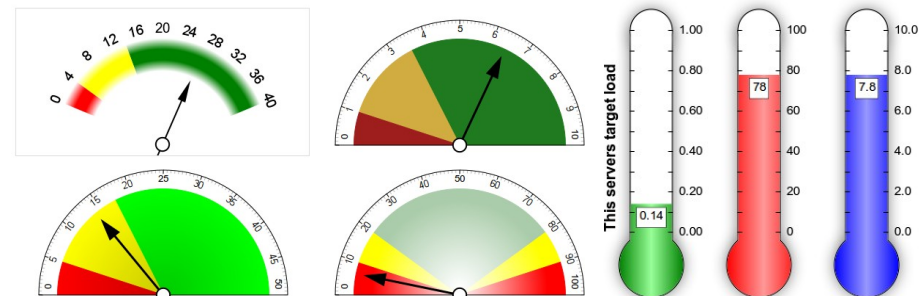
### Afficheurs analogiques à aiguille variés



### Barres progressives



### Multimètres et Thermomètres



### Sympa non ?

Tous ces éléments graphiques sont totalement paramétrables, peuvent subir des effets visuels avancés (mouvement progressif...) et disposent d'une documentation complète : <http://www.rgraph.net/docs/charts-index.html>

## 24. HTML + Javascript : Capturer les événements de widgets graphiques d'une librairie Javascript

### Ce que l'on va faire ici

- Ce que nous allons faire ici va nous servir de préparation pour la suite.
- Nous avons déjà vu le principe d'utilisation des widgets de la librairie RGraph pour réaliser des affichages. Ici, nous allons utiliser une autre fonctionnalité très intéressante des widgets de la librairie RGraph : ils peuvent être utilisés en tant que « dispositif d'entrée », autrement dit, par un simple clic sur un widget, il va être possible de régler sa valeur courante !
- Ceci est rendu possible grâce à l'utilisation de capture d'événements qui sont implémentés pour tous les widgets de la librairie : nous allons voir ici comment les utiliser.

### Le code Javascript (1) : Inclusion fichiers utiles et entête déclarative

- On commence par insérer la fonction permettant de charger facilement les fichiers de la librairie RGraph à partir du chemin absolu,
- Puis on insère les fichiers utiles, à savoir ici :
  - common.js obligatoire,
  - dynamic.js nécessaire pour capture événement souris,
  - effect.js pour l'effet mouvement progressif
  - puis les fichiers des widgets utilisés, ici meter.js et/ou gauge.js
- Enfin, on définit les objets globaux utiles, laissés à null ici :

```
function path(jsFileNameIn) { // fonction pour fixer chemin absolu
    var js = document.createElement("script");
    js.type = "text/javascript";
    js.src = " http://www.mon-club-elec.fr/mes_javascripts/rgraph/"+jsFileNameIn; // <=== modifier ici chemin ++
    document.head.appendChild(js);
    //alert(js.src); // debug
}

//---- fichiers a charger ---
path('RGraph.common.core.js');
path('RGraph.common.dynamic.js');
path('RGraph.common.effects.js');
path('RGraph.gauge.js');
path('RGraph.meter.js');

// variables / objets globaux - a declarer avant les fonctions pour eviter problemes de portee
var canvas= null; // pour objet Canvas
var contextCanvas = null; // pour objet context Canvas
var textInputValeur=null;
```

## Le code Javascript (2) : Fonction initiale : Le tracé du widget et gestion de l'évènement onclick

- Ensuite vient la désormais classique fonction initiale chargée sur l'évènement window.onload
- On commence par déclarer les objets utilisés le canvas, le champ texte
- Puis on déclare et on paramètre le widget graphique. Les propriétés sont décrites en détail dans la documentation de RGraph. On dessine enfin le widget.
- Vient ensuite la fonction clé de notre code :
  - on définit la fonction à exécuter sur l'évènement `canvas.onclick_rgraph` qui reçoit l'objet événement.
  - Cette fonction permet de modifier la valeur de l'objet déclencheur à partir de la valeur obtenue par le click souris.
  - On utilise ensuite la valeur pour mettre à jour le champ texte :
    - soit par un effet progressif (fonction `Effects.Gauge.Grow` )
    - soit par dessin direct (fonction `Draw()` )

```
window.onload = function () { // fonction executee apres chargement de la page HTML

    canvas = document.getElementById("nomCanvas"); // declare objet canvas a partir id = nom
    textInputValeur= document.getElementById("valeur"); // declare objet champ text a partir id = nom

    //--- code graphique ---

    // parametres sont : nom du canva, min, max, valeur courante
    //var gauge = new RGraph.Gauge('nomCanvas', 0, 180, 0); // declare widget graphique
    var gauge = new RGraph.Meter('nomCanvas', 0,180,90);
    gauge.Set('chart.angles.start', PI ); // angle debut
    gauge.Set('chart.angles.end', TWOPI ); // angle fin
    gauge.Set('segment.radius.start', 145); // blanc interne
    gauge.Set('chart.tickmarks.big.num', 10); // nombre marques principales
    gauge.Set('chart.tickmarks.small.num', 100); // nombre marques unitaires
    // il y a 1 label chiffre tous les 1/10 eme de l'echelle globale independemment des marques

    gauge.Draw(); // dessine le widget graphique dans le canvas

    textInputValeur.value=gauge.value; // valeur courante par default

    //--- gestion de l'évènement on click --- necessite dynamic.js
    gauge.canvas.onclick_rgraph = function (e){

        var obj    = e.target.__object__;
        var value = obj.getValue(e);

        obj.value = value;
        textInputValeur.value=Math.round(obj.value); // MAJ valeur courante avec nouvelle valeur

        RGraph.Effects.Gauge.Grow(obj); // necessite effect.js
        //obj.Draw(); // redessine widget

    } // fin onclick_rgraph
} // fin fonction onload
```

## La page HTML + Javascript complète :

- Par mesure de simplicité, le code Javascript n'est pas indiqué ici. Le code Javascript précédent est à intégrer par copier/coller au niveau du head.
- Au niveau du body de la page HTML, on intègre le canvas ainsi que le champ texte,
- ce qui donne :

```
<!DOCTYPE html >
<html>
  <head>

    <meta http-equiv="content-type" content="text/html; charset=UTF-8" /> <!-- Encodage de la page -->

    <script language="javascript" type="text/javascript">
      <!--
        // code Javascript ici
      -->
    </script>

    <title>Test RGraph simple</title>

  </head>

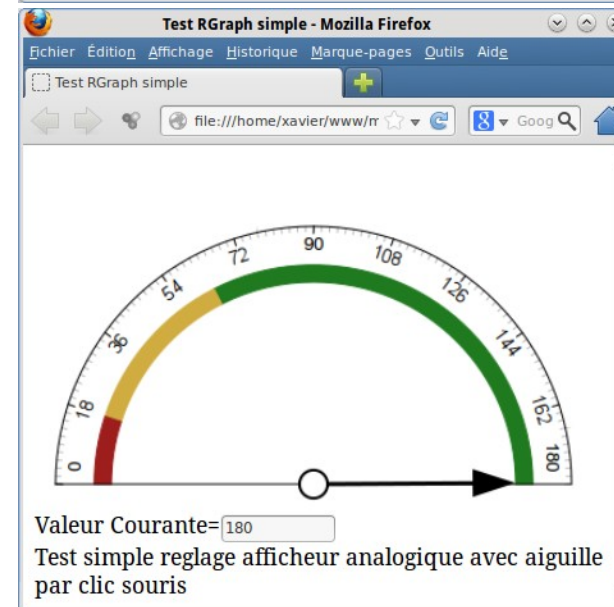
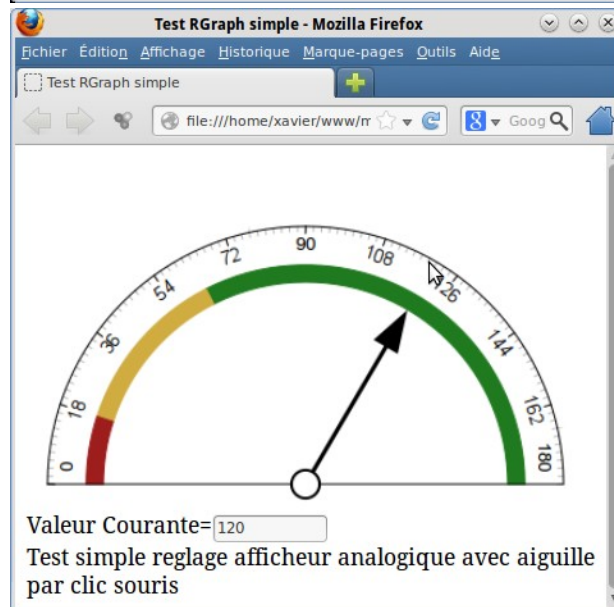
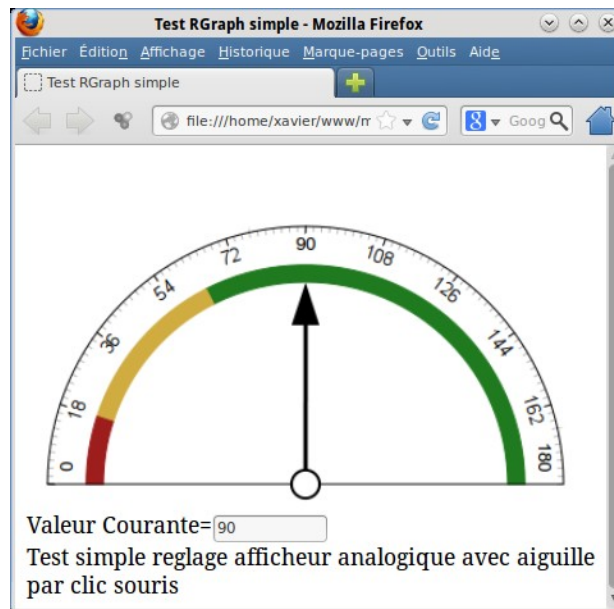
  <body>

    <canvas id="nomCanvas" width="400" height="250">[Canvas non disponible]</canvas>
    <br />
    Valeur Courante=<input type="text" id="valeur" size="10"/>
    <br />
    Test simple reglage afficheur analogique avec aiguille par clic souris

  </body>
</html>
```

## Résultat obtenu

- On obtient un « beau » widget à aiguille allant de 0 à 180 à aiguille : il suffit alors de cliquer sur diverses positions de la graduation pour que l'aiguille se positionne graduellement. La nouvelle valeur courante s'affiche dans le champ texte. [Super sympa je trouve !](#)





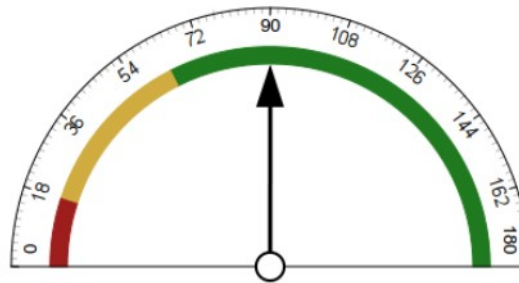
## 25. Serveur Arduino : Contrôler un servomoteur par envoi d'une requête Ajax avec paramètre numérique sur un clic souris sur un widget graphique obtenu à l'aide d'une librairie Javascript

### Ce qu'on va faire ici...

- Ici, nous allons pour l'essentiel reprendre le programme précédent du « slider » en l'adaptant de façon à utiliser un widget graphique :
  - lors d'un clic sur le widget graphique**, déclencher l'envoi d'une requête AJAX intégrant une chaîne « racine » saisie dans un champ texte (ici « **servo**(» ), suivie d'une valeur numérique correspondant à la valeur courante du widget et d'un caractère de fermeture (ici « **)** »).
  - 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.

Serveur Arduino : Test envoi chaine par requete Ajax sur clic  
Widget RGraph



Racine :  Valeur Courante= Fin :

En provenance du serveur Arduino :

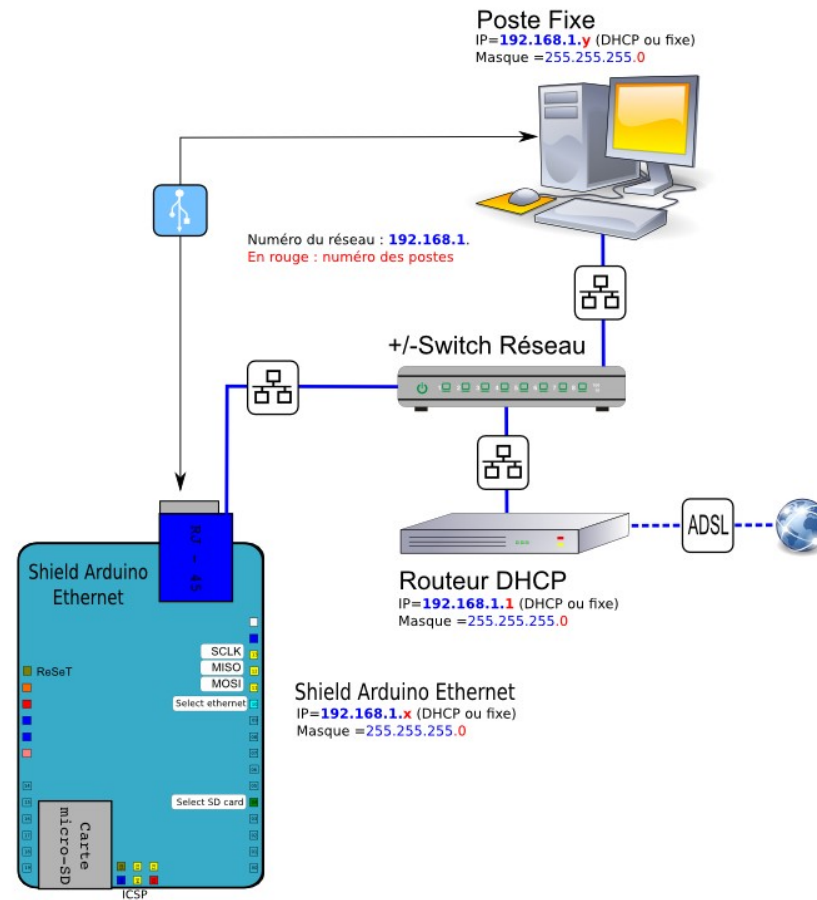
L'interface que nous allons mettre en place ici...

Une véritable « mini-application » graphique contrôlant Arduino via le réseau depuis le navigateur client !



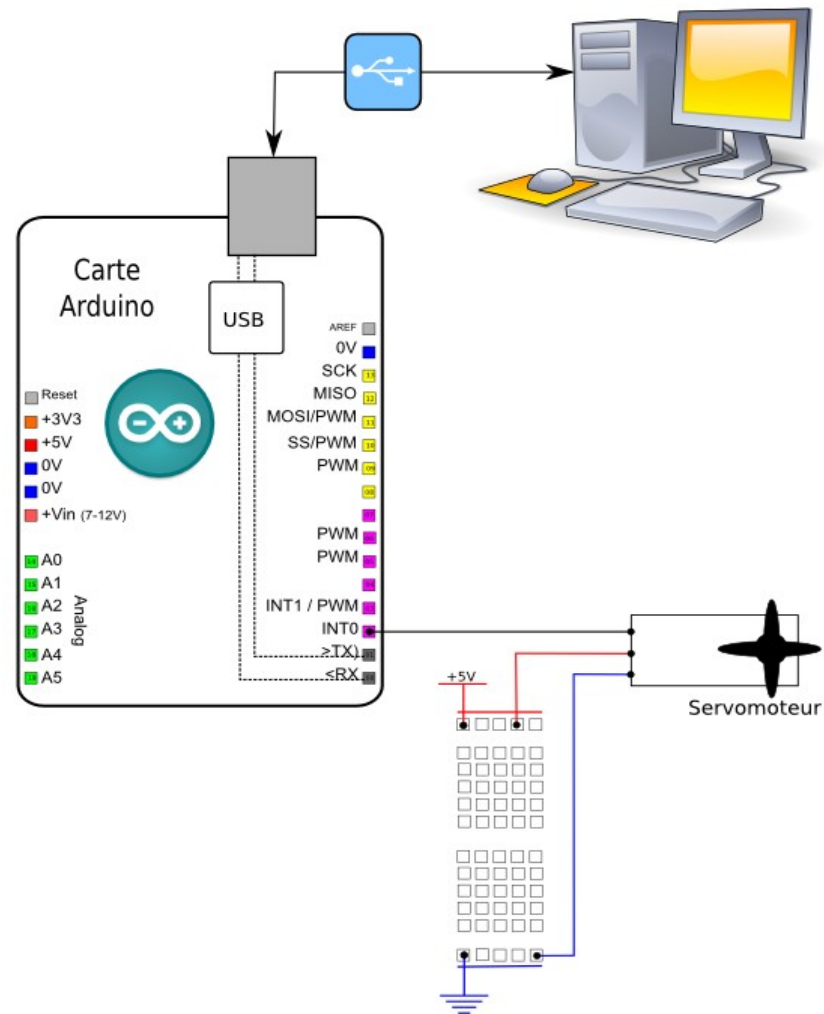
## Le schéma du réseau utilisé

- 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 librairies utiles

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

```
// --- Inclusion des librairies ---  
  
#include <Utils.h> // inclusion de la librairie  
#include <Servo.h> // inclut la librairie Servo  
#include <SPI.h> // librairie SPI - obligatoire avec librairie Ethernet  
#include <Ethernet.h> // librairie 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()** : 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()** : Affichage, analyse de la chaîne reçue et envoi de la réponse aux requêtes Ajax

- Ensuite, tout simplement, on affiche la chaîne reçue
- **Une des clés de ce programme Arduino 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
    }
}
```



## 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(yyy) avec yyy, un angle entre 0 et 180"));
    //-- une fois la réponse Ajax terminée, la fonction de callback drawData est exécutée

} // fin else

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

} // fin if GET /&
```

## Fonction **loop()** : 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()** : Head : 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 commence par charger les fichiers de la librairie Javascript RGraph utiles, à savoir ici :
  - common.js obligatoire,
  - dynamic.js nécessaire pour capture événement souris,
  - effect.js pour l'effet mouvement progressif
  - puis les fichiers des widgets utilisés, ici meter.js et/ou gauge.js
- On définit les variables globales et objets utiles (laissés à null ici) :
  - **ici, l'objet zone de texte et les objets champs texte utilisés,**
  - **l'objet canvas et l'objet context associé**
  - **l'objet Rgraph utilisé, appelé ici gauge**

```
//===== bloc de code javascript =====
client.println(F("<!-- Début du code Javascript -->"));
client.println(F("<script language=\"javascript\" type=\"text/javascript\">"));
client.println(F("<!--      "));

//----- > fonction pour chargement des fichiers <-----
client.println(F("function path(jsFileNameIn) { // fonction pour fixer chemin absolu "}));
client.println(F("var js = document.createElement(\"script\");"));
client.println(F("js.type = \"text/javascript\";"));
client.println(F("js.src = \" http://www.mon-club-elec.fr/mes_javascripts/rgraph/\"+jsFileNameIn; // <=== modifier ici chemin ++ "));
client.println(F("document.head.appendChild(js);"));
client.println(F("//alert(js.src); // debug"));
client.println(F("} "));

client.println(F("//---- fichiers a charger ---"));
client.println(F("path('RGraph.common.core.js');"));
client.println(F("path('RGraph.common.dynamic.js');"));
client.println(F("path('RGraph.common.effects.js');"));
client.println(F("path('RGraph.gauge.js');"));
client.println(F("path('RGraph.meter.js');"));

// variables / objets globaux - a declarer avant les fonctions pour eviter problemes de portee

client.println(F("var canvasRGraph= null; "));
client.println(F("var contextCanvasRGraph = null;"));
client.println(F("var textInputValeur=null;"));
client.println(F("var textarea=null;"));

client.println(F(""));

//---- objets RGraph --
client.println(F("var gauge= null; "));
```

## Fonction **loop()** : Head : Envoi de la fonction appelée au chargement de la page HTML (1)

- Ensuite, on inclut la fonction appelée lors du chargement initial de la page HTML, sur l'évènement `window.onload` :
  - dans un premier temps, on trace le widget RGraph utilisé :
    - on commence par le déclarer avec le mot clé **new** et en précisant le canvas utilisé, la valeur min et max, la valeur courante,
    - ensuite, on personnalise le widget à l'aide de la fonction **.Set** qui permet de fixer les propriétés du widget : ici, on utilise un format demi-cercle de 180°. Se reporter à la documentation de la librairie RGraph pour les détails : <http://www.rgraph.net/docs/charts-index.html>
    - puis on trace le widget initial avec la fonction **Draw**
  - ... de façon à ce que ces objets soient directement accessibles dans le reste du code javascript.

```
client.println(F("window.onload = function () { // au chargement de la page});

//----- éléments RGraph -----

// -- tracé des gauges analogiques --

//client.println(F("gauge = new RGraph.Gauge('cvs', 0, 1023, val);")); // création gauge
client.println(F("var gauge = new RGraph.Meter('cvs', 0,180,90);"));

//----- personnalisation de la Gauge ----
client.println(F("gauge.Set('chart.angles.start', PI );")); // angle debut
client.println(F("gauge.Set('chart.angles.end', TWOPI );")); // angle fin
client.println(F("gauge.Set('segment.radius.start', 145);")); // blanc interne
client.println(F("gauge.Set('chart.tickmarks.big.num', 10);")); // nombre marques principales
client.println(F("gauge.Set('chart.tickmarks.small.num', 100);")); // nombre marques unitaires
client.println(F(""));

//--- dessin de la gauge ---
client.println(F("gauge.Draw();"));
```

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

- Ensuite, toujours dans la fonction appelée lors du chargement initial de la page HTML, sur l'évènement `window.onload` :
  - on déclare et initialise les différents éléments de la page HTML au niveau du code Javascript, le canvas, les champs textes, la zone de texte
  - ... de façon à ce que ces objets soient directement accessibles dans le reste du code javascript.
- Noter ici que l'on déclare l'objet context du Canvas utilisé pour le tracé du widget RGraph : ce n'est pas strictement nécessaire, mais dans la mesure où cela peut-être utile pour effacer le canvas ou autre, je le fais de principe...

```
//----- les éléments de la page HTML -----  
  
//----- canva RGraph ---  
client.println(F("canvasRGraph = document.getElementById(\"cvs\");")); // canvas RGraph  
  
//----- zone texte ----  
client.println(F("textarea = document.getElementById(\"textarea\"); // declare objet canvas a partir id = nom "));  
client.println(F("textarea.value=\"\";")); // efface le contenu  
  
//---- champs texte ----  
client.println(F("textInputValeur= document.getElementById(\"valeur\");"));  
client.println(F("textInputRacine= document.getElementById(\"racine\");"));  
client.println(F("textInputRacineFin= document.getElementById(\"racineFin\");"));  
  
client.println(F("textInputRacine.value=\"servo(\";")); // racine debut  
client.println(F("textInputRacineFin.value=\")\";")); // racine fin  
// valeur courante par default  
client.println(F("textInputValeur.value=gauge.value;"));  
  
// ----- initialisation canvas RGraph ----  
client.println(F("if (canvasRGraph.getContext){"));  
client.println(F("contextCanvasRGraph = canvasRGraph.getContext(\"2d\");"));  
client.println(F("} // fin si canvas existe"));  
  
client.println(F("else {"));  
client.println(F("alert(\"Canvas non disponible\");"));  
client.println(F("} ")); // fin else
```

## Fonction **loop()** : Head : Envoi de la fonction appelée au chargement de la page HTML (3)

- Enfin, toujours au sein de la fonction appelée lors du chargement initial de la page HTML, sur l'évènement `window.onload` : on intègre la fonction de gestion de l'évènement onclick du widget RGraph. **Cette fonction est ici l'un des points clé du code Javascript exécuté côté navigateur client.**
- A la différence du code précédent, on utilise ici la forme « directe » d'association de l'évènement à une fonction qui va recevoir l'objet événement provoqué par le clic souris :
  - on déclare un objet représentant le widget concerné par l'évènement
  - ainsi qu'une variable contenant la valeur de widget correspondant au clic souris à l'aide de la fonction `getValue` de l'objet, qui reçoit l'objet événement
  - une fois fait, les choses sont assez simples :
    - on met à jour la valeur du widget sur la nouvelle valeur, ainsi que le champ texte contenant la valeur courante
    - puis on envoie la requête Ajax en y intégrant la chaîne sous la forme `/&racine(valeur)=`, où valeur est la nouvelle valeur du widget

**Noter la relative simplicité permettant d'obtenir la nouvelle valeur courante à partir du clic souris !**

D'autre par, noter qu'il est nécessaire de placer cette fonction dans la fonction initiale appelée sur `window.onload` car il faut que le widget existe avant d'y associer l'évènement onclick : il s'agit ici d'un événement interne à la librairie RGraph, pas un événement natif HTML...

```
//-- gestion de l'évènement on click sur widget RGraph-- necessite dynamic.js
//-- Doit etre placée dans windows.onload pour que gauge existe quand chargée
client.println(F("gauge.canvas.onclick_rgraph = function (e){");

    client.println(F("var obj  = e.target.__object__;"));
    client.println(F("var value = obj.getValue(e);"));

    client.println(F("obj.value = value;"));

    client.println(F("textInputValeur.value=Math.round(obj.value); ")); // MAJ valeur courante avec nouvelle valeur

    client.println(F("RGraph.Effects.Gauge.Grow(obj);")); // necessite effect.js
    //client.println(F("obj.Draw();")); // redessine widget
    client.println(F(""));

    // envoi de la requete Ajax
    client.println(F("requeteAjax(\"&\"+textInputRacine.value+textInputValeur.value+textInputRacineFin.value+\"=\", drawData);"));
    // envoi requete avec &chaîne= - utilise les champs racine et racineFin pour encadrer la valeur
    // la fonction drawData est appelée une fois la requete exécutée

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

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

## Fonction **loop()** : Head : 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()** : Head : envoi de la fonction Javascript de gestion des données reçues du serveur

- 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.

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

    // ajoute la réponse au champ texte
    client.println(F("textarea.value=textarea.value+stringDataIn;")); // ajoute la chaine 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"));

    // -----
```



## 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 tous les éléments de la page HTML utilisés, à savoir, le canvas, plusieurs champs textes utiles ainsi que la zone de texte utilisée pour afficher les valeurs en provenance du serveur Arduino,
- Suivent les balises de clôture de la page web

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

client.println(F("Serveur Arduino : Test envoi chaine par requete Ajax sur clic Widget RGraph"));
client.println(F("<br/>"));
client.println(F(""));
client.println(F("<canvas id=\"cvs\" width=\"400\" height=\"250\"></canvas>"));
client.println(F("<br />"));

client.println(F("<br />"));
client.println(F("Racine : <input type=\"text\" id=\"racine\" size=\"10\"/>"));
client.println(F("Valeur Courante=<input type=\"text\" id=\"valeur\" size=\"10\"/>"));
client.println(F("Fin : <input type=\"text\" id=\"racineFin\" size=\"5\"/>"));

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 clôturée.

```
        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()
```

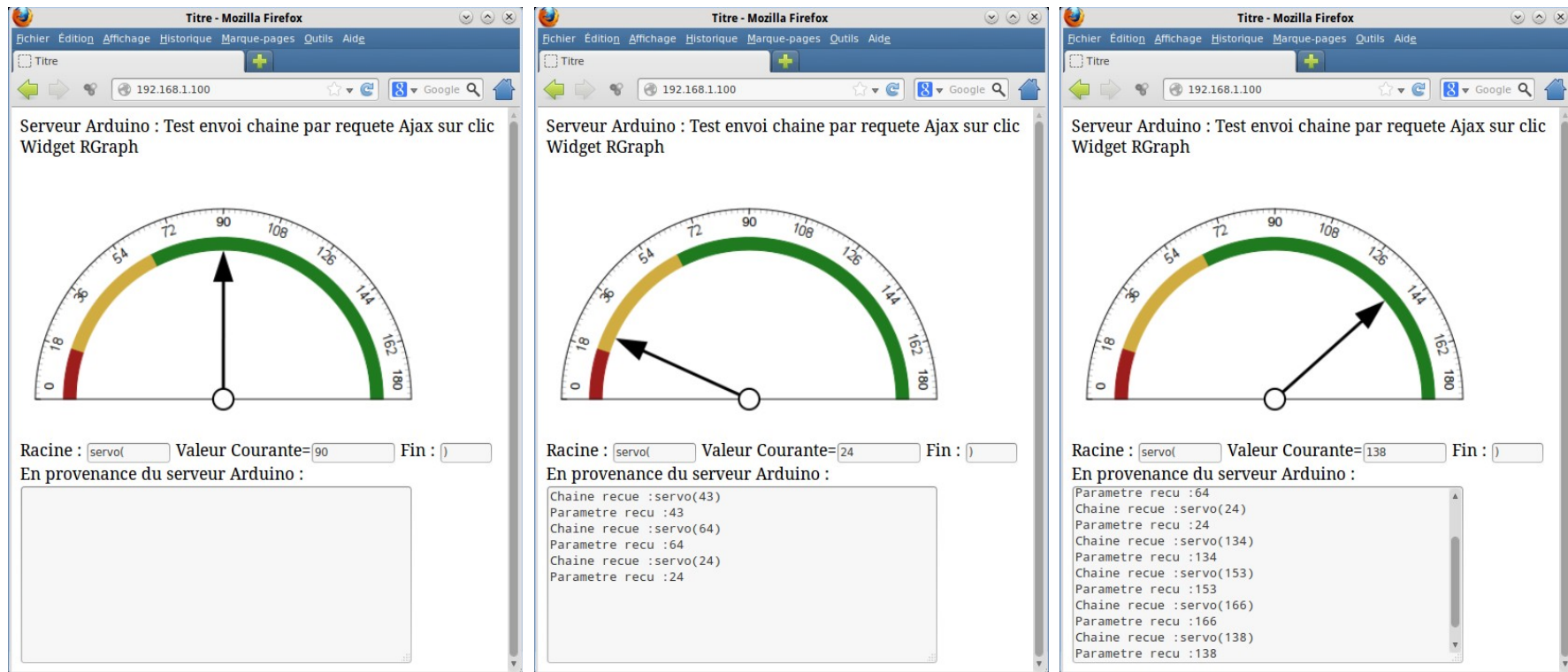
## Fonctions communes : Fonction d'envoi de l'entête http

- 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 racine de la forme servo( et un caractère de fin « ) » de façon à former une chaîne servo(xxx) où xxx est une valeur d'angle entre 0 et 180
- Cliquez sur le widget graphique : **l'aiguille se positionne dynamiquement et le servomoteur va se positionner à l'angle voulu... le tout par le réseau !**



**A ce stade, vous disposez d'une interface graphique dynamique qui permet de contrôler en « temps-réel » par simple clic souris !!**  
le paramètre numérique est envoyé par le réseau vers Arduino qui renvoie des messages attestant la bonne réception de la valeur.  
Et une fois de plus, tout le code se trouve côté Arduino, s'exécute sur le serveur Arduino et dans le navigateur client (Javascript) !

### **Synthèse technique**

A présent, vous savez recevoir des données en temps réel depuis le réseau en provenance du serveur Arduino, vous savez les affichez sous forme graphique dans le navigateur client.

Vous savez également envoyer des données en temps réel vers Arduino vers le serveur Arduino, à partir d'une interface graphique dans le navigateur client.

A ce stade, vous êtes à même de mettre en place de véritables petites applications graphiques interactives s'exécutant dans le navigateur client, et communiquant en temps réel avec Arduino, par requêtes Ajax.

**Vous avez toutes les cartes en main : à vous de jouer !**



## 26. 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](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.ReferenceMaxi)

## **27. A présent, vous devriez être capable :**

- d'utiliser un canvas pour réaliser un simple slider (widget linéaire réglable)
- de déclencher l'envoi de chaînes vers le serveur par requête Ajax à partir d'un clic souris dans un canvas
- de détecter des événements associés à des widgets graphiques d'une librairie javascript dédiée
- de contrôler des dispositifs à l'aide de widgets graphiques par requête Ajax

# Table des matières

Créer un serveur HTML+ Javascript + Ajax avec Arduino et contrôler Arduino graphiquement ou à l'aide de widgets 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) |  
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 : Javascript : le DOM, l'accès aux éléments d'une page HTML et les fonctions de l'objet window |  
Rappel : Javascript : Associer une fonction à la survenue d'un événement attaché à un élément du DOM |  
Rappel : HTML : Présentation de l'objet canvas |  
Rappel : Javascript : Les fonctions essentielles de l'objet canvas |  
Rappel : Objet Canvas : système de coordonnées |  
HTML + Javascript : Utiliser un canvas comme un « slider » |  
Rappel : AJAX : principe et intérêt. |  
Rappel : Javascript : L'objet XMLHttpRequest et son utilisation |  
Rappel : Fonction de requête Ajax modifiée pour passer une chaîne texte |  
Synthèse : Evènement DOM appelant fonction Javascript envoyant requête AJAX vers Arduino |  
Arduino : 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 sur un clic souris dans un canvas utilisé en « slider » |  
Rappel : Utilisation d'une librairie Javascript de dessin de widgets analogiques |  
Vue d'ensemble des graphiques et éléments analogiques disponibles avec la librairie Javascript utilisée |  
HTML + Javascript : Capturer les événements de widgets graphiques d'une librairie Javascript |  
Serveur Arduino : Contrôler un servomoteur par envoi d'une requête Ajax avec paramètre numérique sur un clic souris sur un widget graphique obtenu à l'aide d'une librairie Javascript |  
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](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS)