

Créer un serveur HTML+Javascript avec Arduino et afficher des données sous forme graphique dans un canvas dans le navigateur client.



Ateliers Arduino

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



Tous droits réservés – 2012.

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

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

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

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

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

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

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

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

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

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

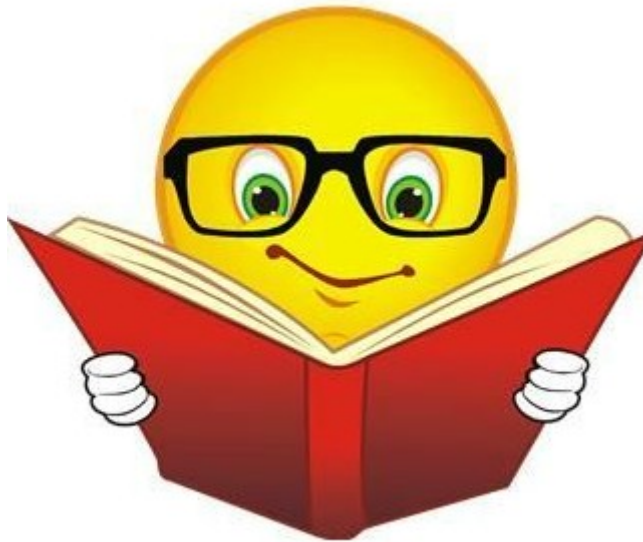
1. Intro

L'objectif ici est :

- d'apprendre à dessiner un simple « vu-mètre » dans un canva
- d'afficher des données numériques sous forme graphique dans un canvas
- de mettre en place une interface graphique analogique simple côté navigateur client

... afin d'être en mesure de créer un serveur Arduino réalisant un affichage graphique évoluées dans le navigateur client

Cet atelier fait suite à un atelier précédent consacré à la présentation du Javascript. Nous utiliserons le même réseau.



Prêt ? C'est parti !

Pratique :

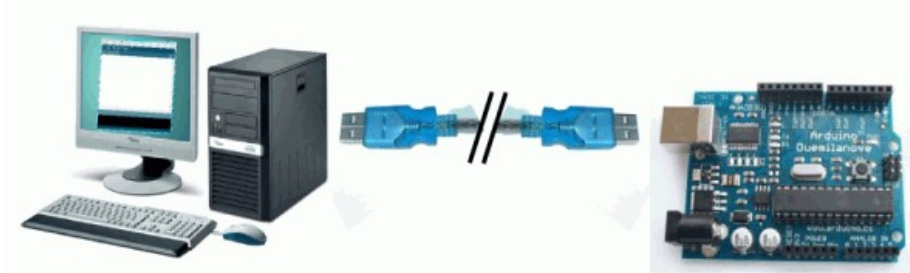
Les codes de cet atelier sont disponibles ici :

http://www.mon-club-elec.fr/mes_downloads/tutos_arduino/12b4.atelier_arduino_ethernet_tcp_javascript_canva_can.tar.gz

2. Matériel nécessaire pour les ateliers Arduino

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

De l'espace de développement Arduino

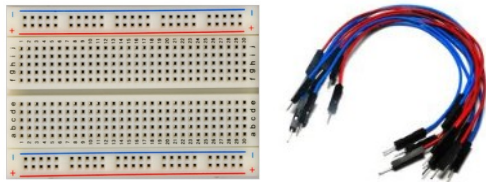


L'espace de développement Arduino associe :

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

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

Du nécessaire pour réaliser des montages sans soudure

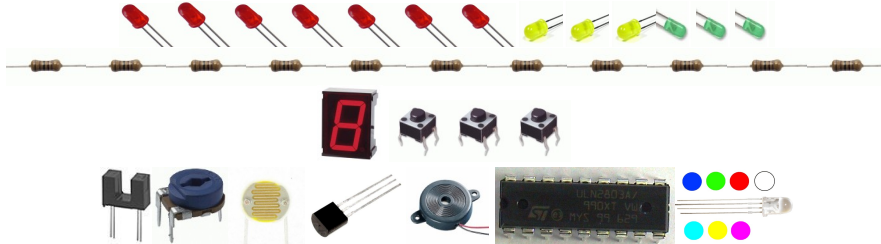


Pour réaliser des montages sans soudure, vous aurez besoin :

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

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

De quelques composants de base



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

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

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

GO TRONIC
ROBOTIQUE ET COMPOSANTS ÉLECTRONIQUES

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

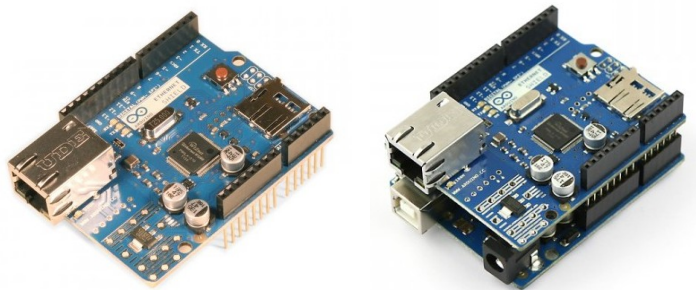
Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

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

3. Matériel spécifique nécessaire pour cet atelier

Pour cet atelier vous aurez besoin également :

D'une carte d'extension (shield) Ethernet



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

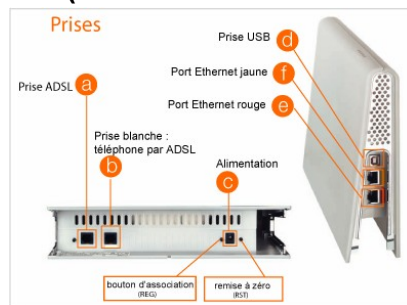
Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 +/- 4) pour communiquer avec Arduino.

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

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

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

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



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

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

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

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



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

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

4. Matériel spécifique nécessaire pour cet atelier (suite)

D'un ou 2 câbles réseau RJ45



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

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

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

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

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

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



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

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

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

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

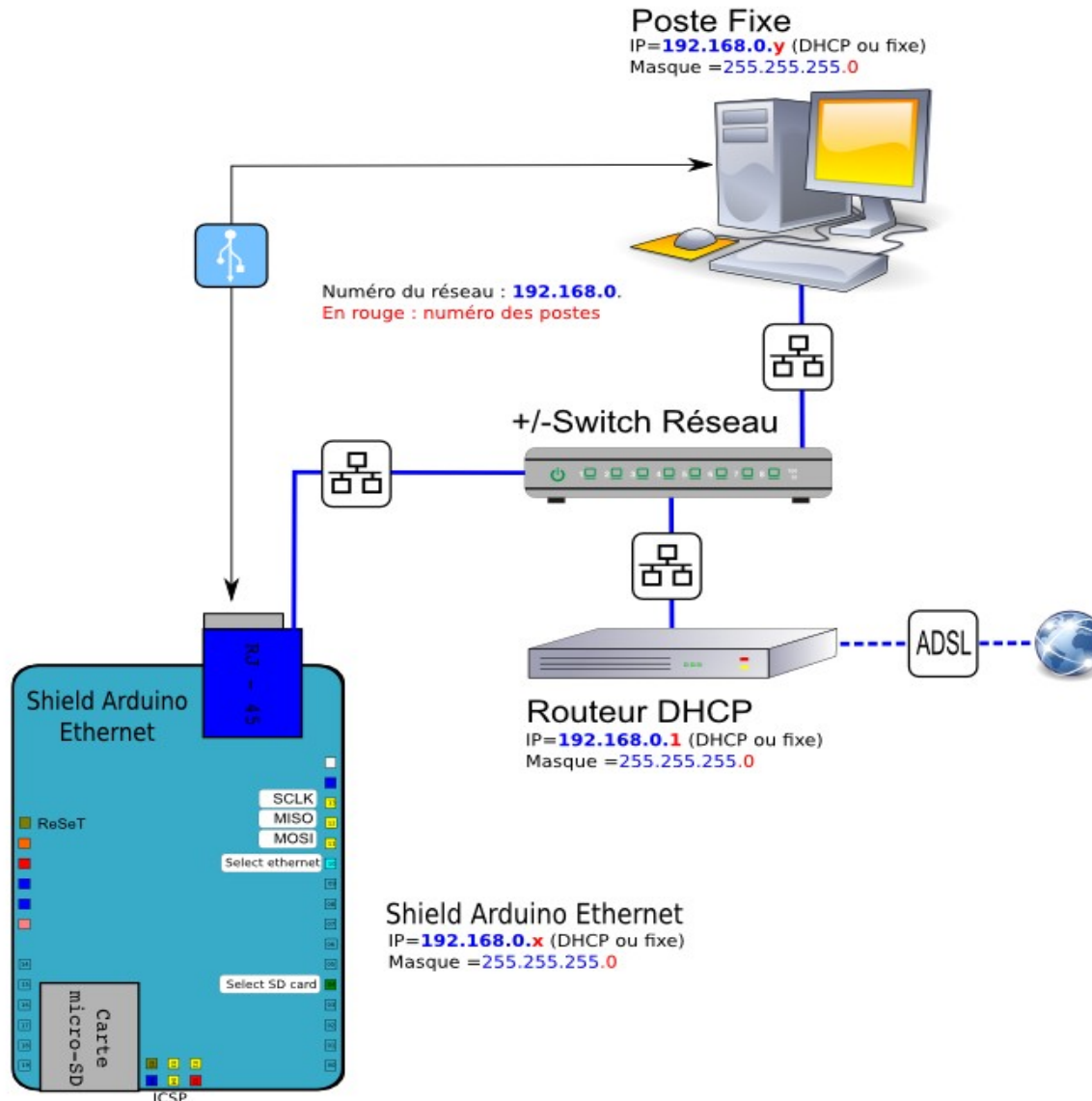
Et d'un poste fixe (PC, Mac, Netbook,...) disposant d'une carte Ethernet !



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

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

5. La structure du réseau que nous allons réaliser



Notre réseau utilisant Arduino va être constitué au minimum :

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

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

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

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

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

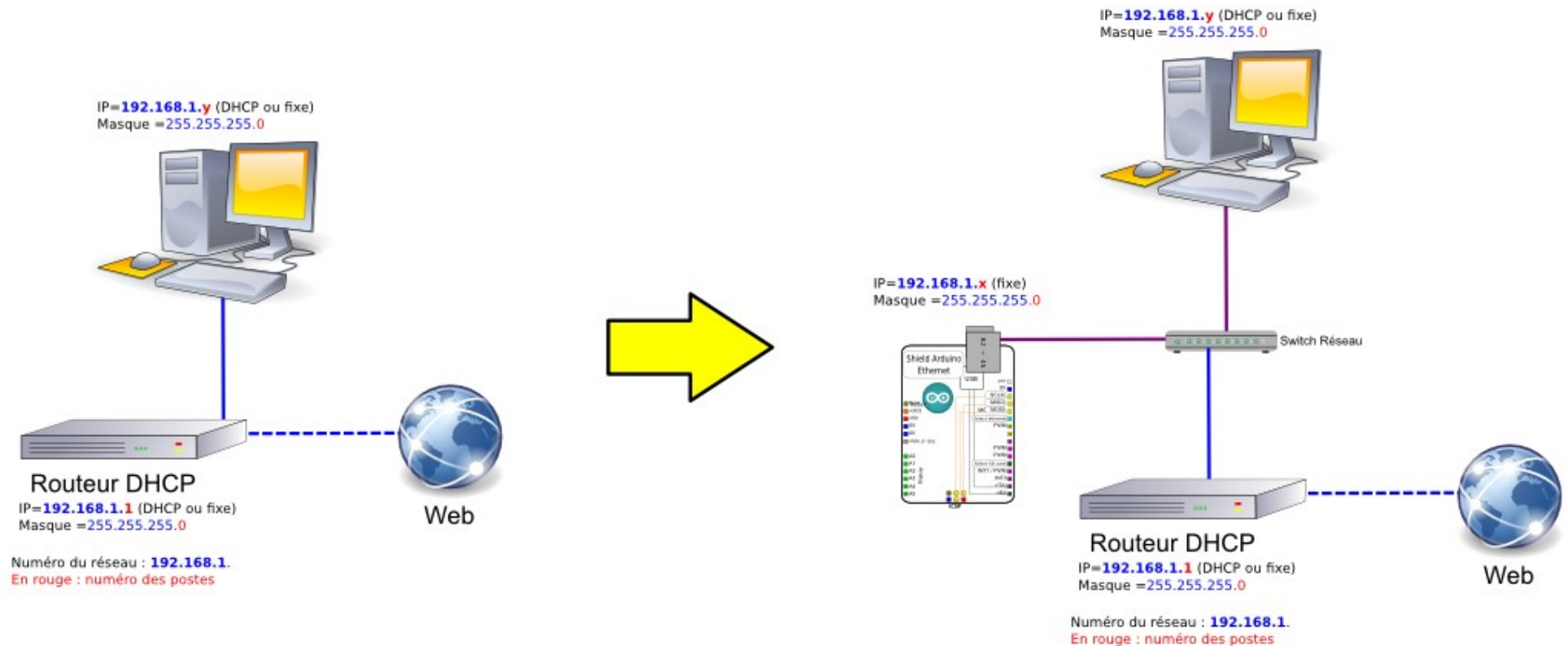
Ceci est très pratique en phase de test et de mise au point, mais une fois la programmation terminée, on pourra bien sûr déconnecter le câble USB.

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

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

Remarque :

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



7. Rappel : Syntaxe de base du langage Javascript

Intro

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

Structure

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

Variables

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

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

Tableaux

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

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

Condition if

- Identique à Arduino :

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

Boucle For

- Idem Arduino :

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

Boucle While

- Idem Arduino :

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

Fonctions

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

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

    return expression;
}
```

Déclarer un objet

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

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

Fonctions de l'objet window

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

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

Pour aller plus loin

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

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

De quoi avez-vous besoin ?

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

Pour info : différentes façon d'utiliser Javascript

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

Balise HTML d'insertion d'un code javascript

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

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

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

-->
</script>
```

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

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

Head ou Body ?

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

Fonction onload()

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

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

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

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

    } // fin onload

-->
</script>
```

Votre première page HTML + Javascript

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

```
<!DOCTYPE HTML>

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

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

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

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

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

9. Rappel : 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 nom
 - 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 300 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
- Une petite page qui permet facilement de tester les possibilités du canvas :
<http://jm.davalan.org/lang/jsc/js09.html>

10. Javascript : Rappel : 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

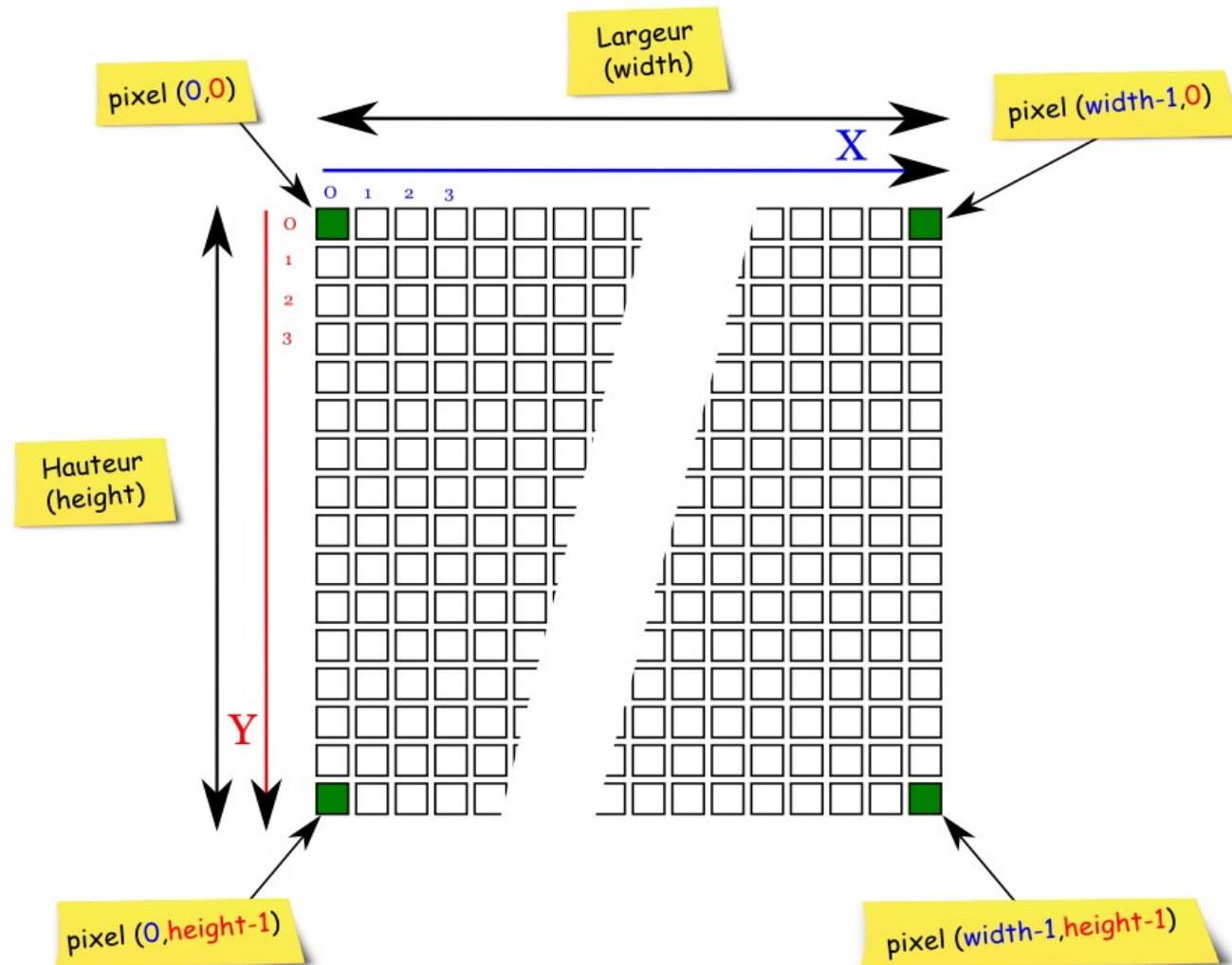
- Une fois que l'on dispose de l'objet Context d'accès aux fonctions de dessin du Canvas, on va pouvoir passer à l'action : **en fait, à ce stade, c'est tout un nouveau monde de possibilités qui s'ouvre à vous !** Un peu à la façon processing pour ceux qui connaissent...
- Tout d'abord, on peut modifier 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
- 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
```

11. Objet Canvas : système de coordonnées

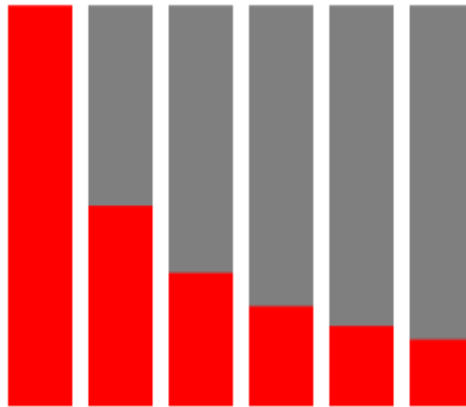
- Le système de coordonnées d'un canvas de largeur width et de hauteur height est le suivant :



12. HTML+ Javascript : dessiner une interface « vu-mètre » dans un canvas

Ce que l'on va faire ici

- Je vous propose ici quelques chose d'assez basique d'un point de vue graphique, mais facile à coder : réaliser une interface graphique type « vu-mètre » dans un canvas. Ici, nous posons les bases pour afficher sous forme graphique la mesure analogique des 6 voies analogiques d'Arduino dans le navigateur client. Pour l'instant, nous allons le coder sur le poste fixe.
- Le dessin de notre interface est très simple et consiste à :
 - tracer 6 rectangles allongés verticalement de couleur grise
 - sur lesquels seront dessinés 6 autres rectangles allongés de même largeur et de même position mais dont la couleur sera rouge et la hauteur variable
- L'ensemble abouti à un aspect de 6 vumètres basiques :



Le code Javascript

Structure générale

- On commence par encadrer le code au sein d'une fonction appelée lors de la survenue de l'évènement **onload** de l'objet window (qui représente la fenêtre du navigateur où est chargée la page) pour que le code javascript ne soit appelé que lorsque tous les éléments de la pages HTML ont été chargés.
- On déclare ensuite l'objet canvas que l'on récupère (à partir de son nom utilisé dans le code HTML) à l'aide de la fonction **getElementById()**
- On peut au besoin reparamétrer la taille du canvas, mais ça n'est pas obligatoire
- Ensuite, on teste l'existence du canvas à l'aide de la propriété **getContext** qui renvoie True si le canvas existe : on place le code graphique au sein de cette condition.

Code graphique

- On commence le code graphique à proprement parler (placé au sein de la condition vérifiant l'existence du canvas) en créant l'objet Context qui va donner accès aux fonctions graphiques du canvas : ceci se fait à l'aide de la fonction **getContext()** (« 2d ») de l'objet canvas,
- Ensuite, on commence par tracer 6 rectangles pleins régulièrement espacés et de taille identique en fixant la couleur rgb grise au préalable
- Puis on trace par dessus les 6 rectangles pleins de couleur rouge, également régulièrement espacés :

```
window.onload = function() {  
  
    var canvas = document.getElementById("nomCanvas"); // declare objet canvas à partir id  
  
    canvas.width = 300; // largeur canva  
    canvas.height = 300; // hauteur canva  
  
    if (canvas.getContext){ // la fonction getContext() renvoie True si canva accessible  
  
        var ctx = canvas.getContext("2d"); // objet contexte permettant acces aux fonctions de dessin  
  
        ctx.fillStyle = "rgb(1000,1000,1000)"; // couleur de remplissage  
        ctx.fillRect (0, 0, canvas.width, canvas.height); // rectangle de la taille du canva  
  
        //-- variables utiles  
        var largeur=40;  
        var hauteur=250;  
        var espacement=10;  
  
        // le code graphique ci-dessous  
        for (var i=0; i<6; i++) {  
  
            // vu-metre 1  
            ctx.fillStyle = "rgb(127,127,127)"; // couleur remplissage - gris  
            ctx.fillRect (10+(i*largeur)+(i*espacement), canvas.height-10, largeur, -hauteur); // 6 rectangles gris  
  
            ctx.fillStyle = "rgb(255,0,0)"; // couleur remplissage - rouge  
            ctx.fillRect (10+(i*largeur)+(i*espacement), canvas.height-10, largeur, -hauteur/(i+1)); // 6 rectangles rouges  
  
        } // fin for  
  
    }  
    else {  
        // si trace non supporte  
    }  
} // fin window.onload
```


Le code HTML + Javascript complet (1) : le head

- On intègre ce code au niveau du Head de la 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">
      <!--
        // code javascript par X. HINAULT - www.mon-club-elec.fr - tous droits réservés - GPL v3
        window.onload = function() {

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

          canvas.width = 300; // largeur canva
          canvas.height = 300; // hauteur canva

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

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

            ctx.fillStyle = "rgb(1000,1000,1000)"; // couleur de remplissage
            ctx.fillRect (0, 0, canvas.width, canvas.height); // rectangle de la taille du canva

            //-- variables utiles
            var largeur=40;
            var hauteur=250;
            var espacement=10;

            // le code graphique ci-dessous
            for (var i=0; i<6; i++) { // boucle pour 6 rectangles

              // vu-metre
              ctx.fillStyle = "rgb(127,127,127)"; // couleur remplissage - gris
              ctx.fillRect (10+(i*largeur)+(i*espacement), canvas.height-10, largeur, -hauteur); // 6 rectangles gris

              ctx.fillStyle = "rgb(255,0,0)"; // couleur remplissage - rouge
              ctx.fillRect (10+(i*largeur)+(i*espacement), canvas.height-10, largeur, -hauteur/(i+1)); // 6 rectangles rouges

            } // fin for

          } // fin if getContext
          else {

            // si trace non supporte
          } // fin else

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

Le code HTML + Javascript complet (2) : le body

- Au niveau du body, les choses sont simples : on insère simplement une balise canvas avec les paramètres définissant le canvas, notamment :
 - le nom (**le même nom que celui utilisé avec la fonction getElementById() dans le code javascript +++**)
 - une largeur et une hauteur en pixels
- le reste de la page HTML, ici limitée à un simple message texte,
- ce qui nous donne :

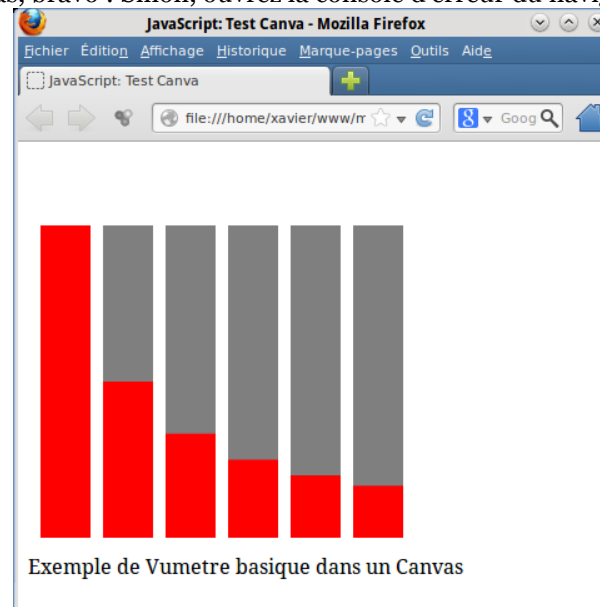
```
<!-- Debut Corps de page -->
<body >
    <canvas id="nomCanvas" width="300" height="300"></canvas>
    <br />
    Exemple de Vumetre basique dans un Canvas

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

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

Résultat

- Ensuite, il suffit de lancer votre page dans le navigateur (je conseille Firefox) :
 - soit clic sur le bouton dédié de votre éditeur (le « globe » dans bluefish)
 - soit clic droit sur le nom du fichier > ouvrir avec Firefox
 - soit ouvrir le navigateur et menu Fichier > ouvrir
- Vous devez obtenir les 6 vu-mètres : si c'est le cas, bravo ! Sinon, ouvrez la console d'erreur du navigateur et vérifiez ce qui ne vas pas.



13. Serveur Arduino : Envoyer une page HTML + Javascript et afficher des valeurs numériques sous forme graphique dans un canvas dans le navigateur client

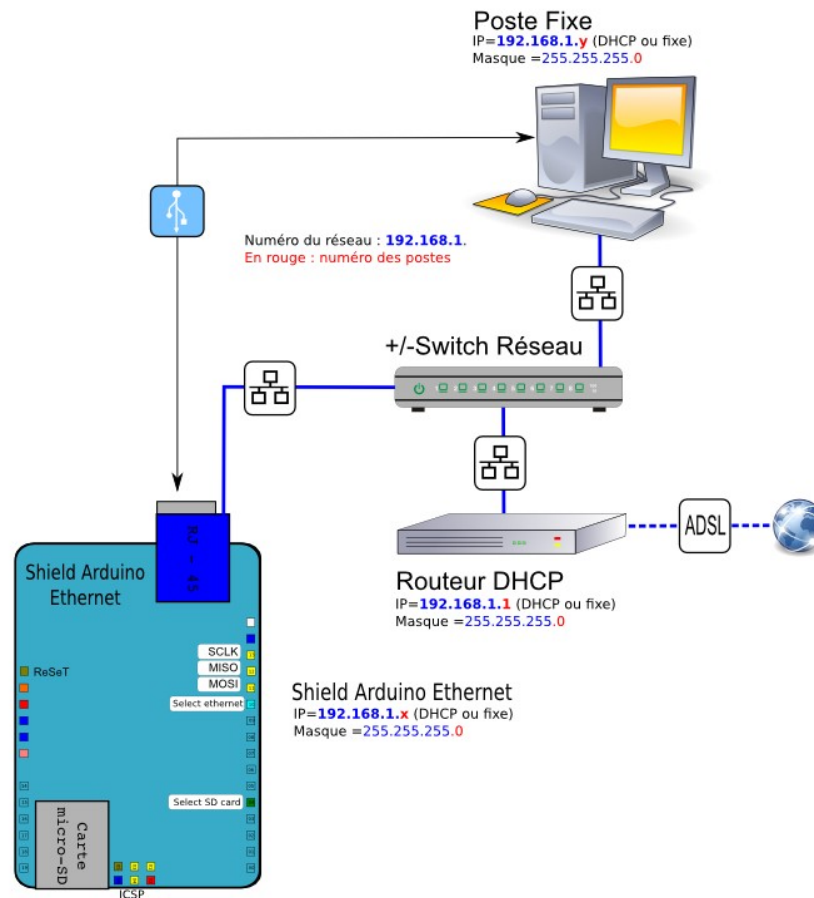
Ce qu'on va faire ici...

- Bien, à ce stade, nous allons une nouvelle fois pouvoir intégrer notre script à notre serveur Arduino, mais cette fois, la hauteur des rectangles rouges sera le reflet de la mesure des voies analogiques de l'Arduino !

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

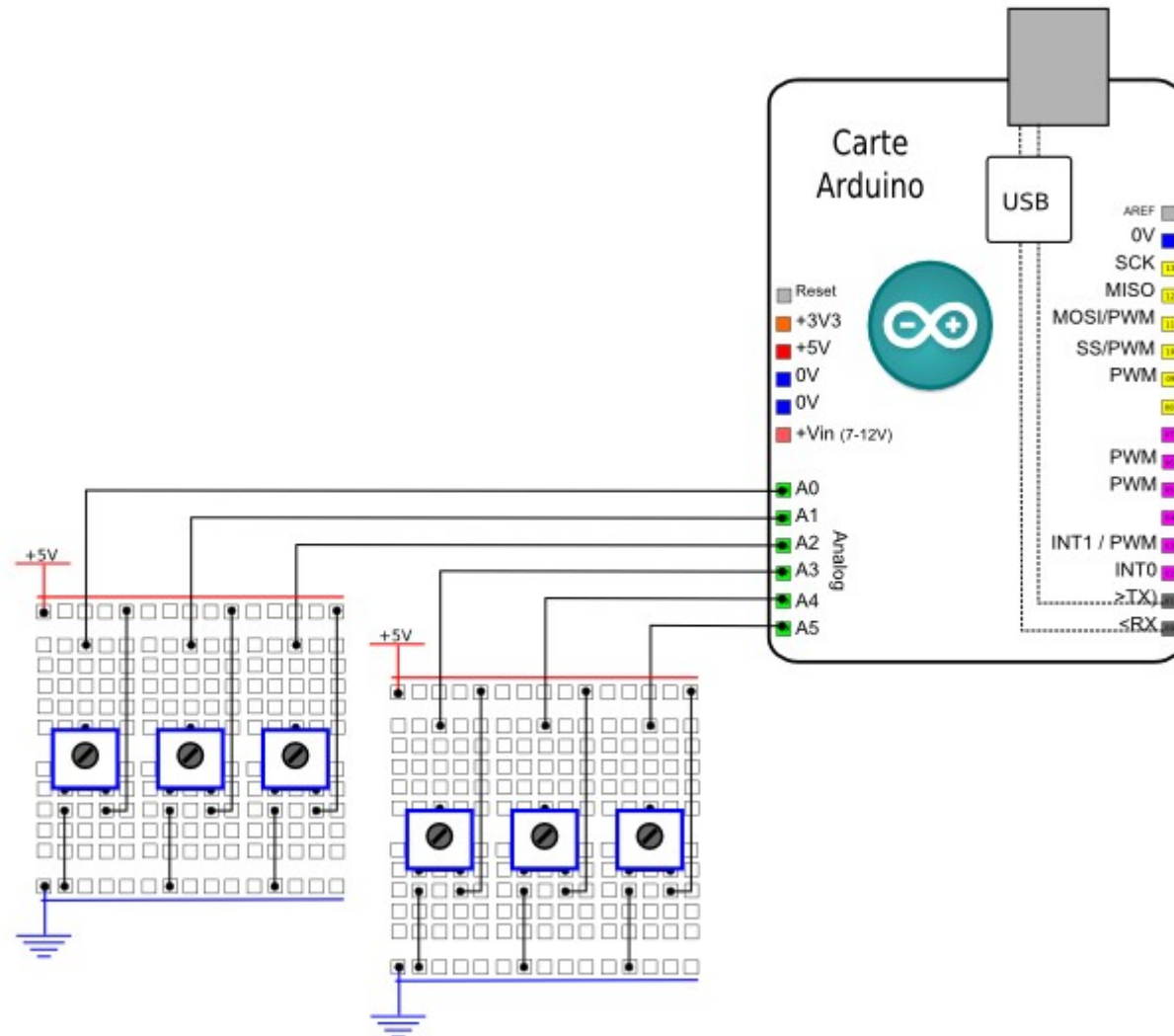
Le schéma du réseau utilisé

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



Le montage Arduino associé

- Ici, nous allons réaliser une mesure analogique sur les 6 voies analogiques de la carte Arduino. Il est possible de les laisser non-connectées pour un simple test : leur valeur variera de façon aléatoire.
- Mais on pourra bien sûr y connecter jusqu'à 6 capteurs analogiques représentés ici par 6 résistances variables (le shield Ethernet et le réseau ne sont pas représentés ici par souci de simplification...) :



Entête déclarative

Inclusion des bibliothèques utiles

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

Configuration du shield Ethernet

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

Variables utiles

- On déclare enfin des variables utiles pour la réception de la chaîne sur le réseau.

```
// --- Inclusion des bibliothèques ---  
  
#include <SPI.h> // bibliothèque SPI - obligatoire avec bibliothèque Ethernet  
#include <Ethernet.h> // bibliothèque Ethernet  
  
// --- Déclaration des variables globales ---  
  
//--- l'adresse mac = identifiant unique du shield  
// à fixer arbitrairement ou en utilisant l'adresse imprimée sur l'étiquette du shield  
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x1A, 0x71 };  
  
//----- l'adresse IP fixe à utiliser pour le shield Ethernet ---  
IPAddress ipLocal(192,168,1,100); // l'adresse IP locale du shield Ethernet  
// ATTENTION : il faut utiliser une adresse hors de la plage d'adresses du routeur DHCP  
// pour connaître la plage d'adresse du routeur : s'y connecter depuis un navigateur à l'adresse xxx.xxx.xxx.1  
// par exemple : sur livebox : plage adresses DHCP entre .10 et .50 => on peut utiliser .100 pour le shield ethernet  
  
// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---  
  
//--- création de l'objet serveur ---  
EthernetServer serveurHTTP(80); // crée un objet serveur utilisant le port 80 = port HTTP  
  
String chaineRecue=""; // déclare un string vide global pour réception chaîne requête  
int comptChar=0; // variable de comptage des caractères reçus
```

Fonction **setup()**

Initialisation série

- On initialise la connexion série

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

//---- 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 et analyse de la chaîne reçue

- Ensuite, tout simplement, on affiche la chaîne reçue
- Puis on teste si la chaîne commence bien l'entête GET attendue dans le cas de la réception d'une requête HTTP valide :

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

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

//----- analyse si la chaîne reçue est une requete GET -----
if (chaineRecue.startsWith("GET")) { // si la chaîne recue commence par GET

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

//----- +/- extraction et analyse de la sous-chaîne utile -----
```

Fonction **loop()** (3) : Envoi de la réponse Http

- on envoie ensuite une réponse HTTP valide, affichée également en copie dans le terminal série :
 - **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

```
////////// Réponse HTTP suivie de la Page HTML de réponse //////////  
  
//-- envoi de la réponse HTTP ---  
client.println(F("HTTP/1.1 200 OK")); // entete de la réponse : protocole HTTP 1.1 et exécution requete réussie  
client.println(F("Content-Type: text/html")); // précise le type de contenu de la réponse qui suit  
client.println(F("Connection: close")); // précise que la connexion se ferme après la réponse  
client.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"));  
Serial.println();
```

Remarque technique :

Noter l'utilisation abondante de la forme `println(F(« chaine »))` 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) : Envoi du début et du head (1) de la page HTML de réponse

- 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
- Au niveau du head, **il est possible d'insérer une balise meta pour assurer un rafraîchissement automatique de la page** : nous allons faire cela ici de façon à obtenir un effet « temps réel » de l'affichage, en fixant un rafraîchissement toute les secondes. « Qui peut le plus, peut le moins » donc il sera évidemment possible d'utiliser des délais de rafraîchissement automatique plus courts.

Techniquement, avec le rafraîchissement automatique, le navigateur va adresser une requête Http « GET » au serveur sans que l'utilisateur n'ait besoin de cliquer sur le bouton de rafraîchissement de la page. Résultat : la page s'actualise automatiquement.

```
//----- 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("<meta http-equiv=\"refresh\" content=\"1\" /> <!-- pour actualisation auto toutes les x secondes -->"));  
client.println(F("<title>Titre</title>")); // titre de la page HTML
```

Fonction **loop()** (4) : Head (2) : Envoi du code Javascript

- A ce niveau, nous insérons la balise script et nous insérons à l'aide de **println()** successifs le code javascript vu précédemment .
- A l'aide d'une boucle, nous allons par ailleurs passer en paramètres au sein du code Javascript les valeurs issues de la mesure analogique des voies analogiques. Ces valeurs seront ensuite utilisées dans le code Javascript pour fixer la hauteur des vu-mètres :

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

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

client.println(F("var canvas = document.getElementById(\"nomCanvas\"); // declare objet canvas a partir id = nom "));
client.println(F("canvas.width = 300; // largeur canvas"));
client.println(F("canvas.height = 300; // hauteur canvas"));

client.println(F("if (canvas.getContext){ // la fonction getContext() renvoie True si canvas accessible"));

//-----> Le Code graphique <-----
client.println(F("var ctx = canvas.getContext(\"2d\"); // objet contexte permettant acces aux fonctions de dessin"));
client.println(F("ctx.fillStyle = \"rgb(255,255,255)\"; // couleur de remplissage"));
client.println(F("ctx.fillRect (0, 0, canvas.width, canvas.height); // rectangle de la taille du canva"));

client.println(F("//-- variables utiles "));
client.println(F("var largeur=40;"));
client.println(F("var hauteur=255; "));
client.println(F("var espacement=10;"));

//-- envoi des 6 valeurs analogique sous la forme var valeur=[val0,val1,val2,val3,val4,val5];
client.print(F("var valeur="));
Serial.print(F("var valeur=")); // debug

for (int i=0; i<6; i++) { // for
  int valeur=analogRead(i); // lit la voie analogique i
  valeur=map(valeur, 0,1023,0,255); // calcule la hauteur du rectangle
  client.print(valeur);
  Serial.print(valeur); // debug
  if (i<5) client.print(",");
  if (i<5) Serial.print(",");// debug
} // fin for

client.println(F("; // tableau de 6 valeurs numeriques"));
Serial.println(F("; // tableau de 6 valeurs numeriques"));

// -- tracé des rectangles --
client.println(F("for (var i=0; i<6; i++) { // boucle pour 6 rectangles"));
client.println(F("ctx.fillStyle = \"rgb(127,127,127)\"; // couleur remplissage - gris"));
client.println(F("ctx.fillRect (10+(i*largeur)+(i*espacement), canvas.height-10, largeur, -hauteur); // 6 rectangles gris"));
client.println(F("ctx.fillStyle = \"rgb(255,0,0)\"; // couleur remplissage - rouge"));
client.println(F("ctx.fillRect (10+(i*largeur)+(i*espacement), canvas.height-10, largeur, -valeur[i]); // 6 rectangles rouges"));
client.println(F("} // fin for"));

//-----> Fin du Code graphique <-----
client.println(F("} // fin si canvas existe"));

client.println(F("else {}"));
client.println(F("// code si canvas non disponible "));
client.println(F("} // fin else"));

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

client.println(F("//--"));
client.println(F("</script>"));
client.println(F("<!-- Fin du code Javascript --> "));
//===== fin du bloc de code javascript =====

client.println(F("</head>"));
```

Fonction **loop()** (5) : 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 au besoin.
- **Ici, point essentiel, on insère la balise canvas dans le body de la page HTML.**
- Suivent les balises de clôture de la page web

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

// affiche chaines caractères simples
client.println(F("<CENTER>")); //pour centrer la suite de la page
client.println(F("<canvas id=\"nomCanvas\" width=\"300\" height=\"300\"></canvas>"));
client.println(F("<br/>"));
client.println(F("<br/>"));
client.println(F("Serveur Arduino : Test affichage des 6 voies analogiques dans un canvas"));
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()** (6) : Fermeture de la connexion avec le client

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

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

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

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

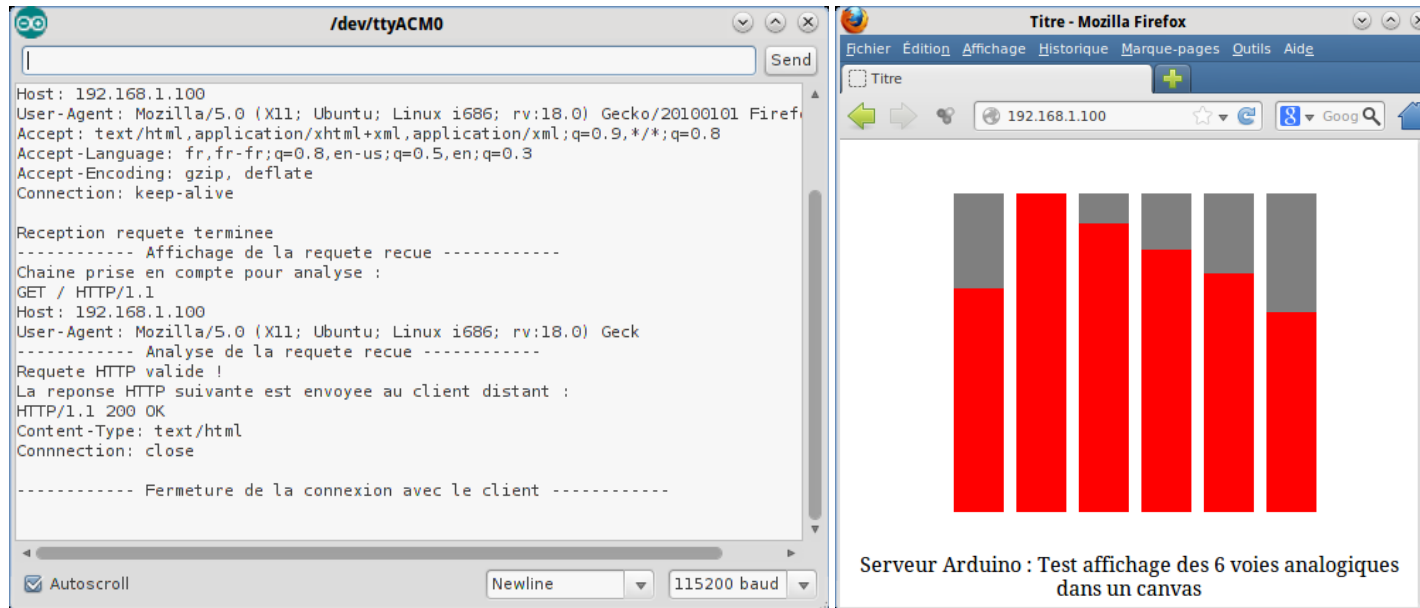
    } // --- fin if client connected

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

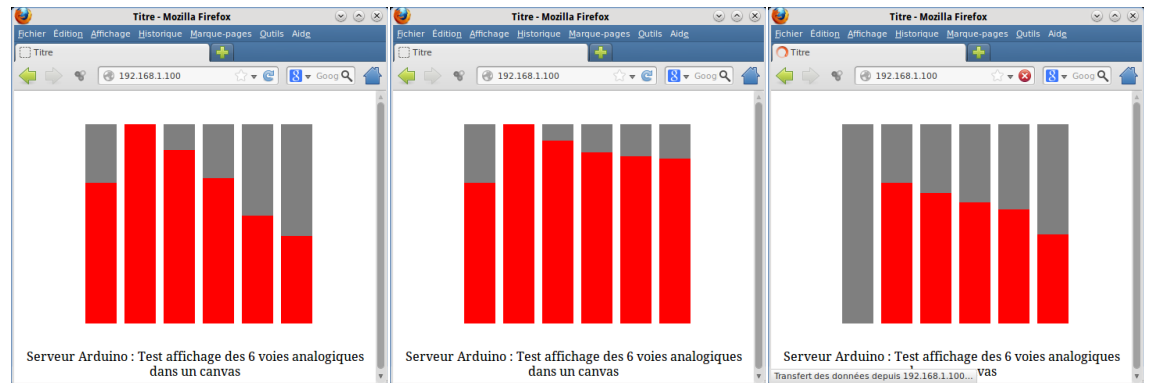
} // fin de la fonction loop()
```

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 notre canvas s'afficher avec nos 6 vu-mètres.



Les vumètres s'affichent dans le navigateur ? Bravo ! Vous avez écrit votre première interface graphique envoyée par votre serveur Arduino en utilisant un canvas et un code Javascript !



Cerise sur le gâteau : la page s'actualise toutes les secondes, permettant un monitoring « temps-réel » des sorties analogiques de la carte Arduino sur le réseau !

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

Atelier Arduino : Créer un serveur HTML+Javascript avec Arduino et afficher des données sous forme graphique dans un canvas dans le navigateur client. p.28/33

Synthèse technique

Ce code est quand même une petite prouesse : en 17Ko on réalise un serveur Arduino générant une page HTML dynamique incluant des valeurs de mesures analogiques (ça pourrait être n'importe quoi d'autre...) et réalisant un affichage graphique dans un canvas généré par un code Javascript exécuté côté client.

Noter au passage l'intérêt de rester en IP fixe côté Arduino, l'utilisation du DHCP ajoutant plusieurs Ko et limitant en conséquence l'espace restant.

Certains me diront cependant que l'affichage est très basique ! Certes, mais le but ici est de montrer le principe, poser les bases. Ensuite, dans la mesure où l'on utilise un canvas, tout est possible et l'implémentation d'affichages beaucoup plus élaborés est possible comme nous allons le voir.

Une petite page qui permet facilement de tester les possibilités du canvas : <http://jm.davalan.org/lang/jsc/js09.html>

Et ici pour quelques affichages analogiques sympa : <http://geeksretreat.wordpress.com/2012/04/13/making-a-speedometer-using-html5s-canvas/>

D'autre part, ici nous insérons le code Javascript directement dans la page HTML : tant que le code Javascript n'est pas trop conséquent, c'est jouable. Mais si on utilise un code Javascript important, il est beaucoup plus pratique de l'intégrer dans un fichier séparé. Avec Arduino, 2 solutions sont possibles : soit mettre le fichier sur une SD Card, soit le lire sur un serveur externe, ce qui suppose d'être connecté à internet via le routeur du réseau.

Dernier point : ici nous réalisons un rafraîchissement automatique de la page à intervalle régulier. Mais il est possible de faire encore mieux : récupérer les données en temps réel sur le serveur Arduino pour mise à jour de la page sans la recharger. C'est possible avec la technique AJAX que nous verrons dans un tuto suivant.

La suite ?

**Encore plus fort : des affichages analogiques évolués dans la navigateur client !
Rendez-vous dans le tuto suivant...**

14. 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

15. A présent, vous devriez être capable :

- de dessiner un simple « vu-mètre » dans un canva
- d'afficher des données numériques sous forme graphique dans un canvas
- de mettre en place une interface graphique analogique simple côté navigateur client

Table des matières

Créer un serveur HTML+Javascript avec Arduino et afficher des données sous forme graphique dans un canvas dans le navigateur client.

Intro |

Matériel nécessaire pour les ateliers Arduino |

Matériel spécifique nécessaire pour cet atelier |

Matériel spécifique nécessaire pour cet atelier (suite) |

La structure du réseau que nous allons réaliser |

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

Rappel : Syntaxe de base du langage Javascript |

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

Rappel : HTML : Présentation de l'objet canvas |

Javascript : Rappel : Les fonctions essentielles de l'objet canvas |

Objet Canvas : système de coordonnées |

HTML+ Javascript : dessiner une interface « vu-mètre » dans un canvas |

Serveur Arduino : Envoyer une page HTML + Javascript et afficher des valeurs numériques sous forme graphique dans un canvas dans le navigateur client |

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