

Créer un serveur HTML+Javascript avec Arduino et afficher des données reçues « en temps réel » par requêtes AJAX sous forme de courbes et de widgets dans des canvas à l'aide d'une librairie javascript dédiée.



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 !

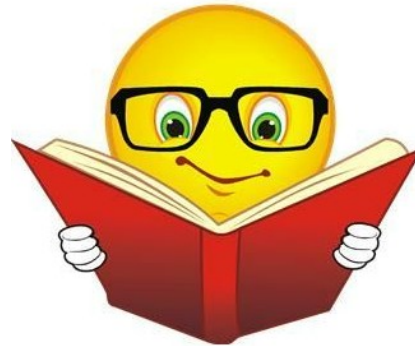
Lancer également le logiciel Arduino et connecter votre carte Arduino afin de pouvoir tester au fur et à mesure les codes d'exemples !

1. Intro

L'objectif ici est :

- d'apprendre à afficher les données reçues par requêtes Ajax en temps réel à l'aide d'une librairie javascript permettant des affichages graphiques élaborés
 - d'apprendre à afficher simultanément une courbe et un widget graphique pour afficher des données reçues par requêtes Ajax
- ... afin d'être en mesure de créer un serveur Arduino réalisant un affichage graphique évoluées dans le navigateur client

Cet atelier fait suite à plusieurs précédent consacrés à l'utilisation de javascript, d'AjAx et de l'objet canvas pour réaliser des affichages graphiques.
Nous utiliserons le même réseau.



Prêt ? C'est parti !



Pratique :

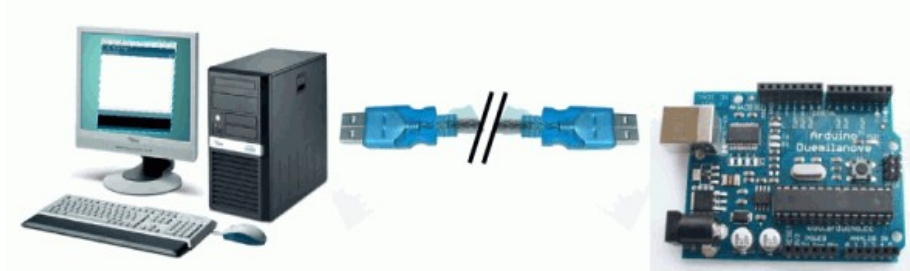
Les codes de cet atelier sont disponibles ici :

http://www.mon-club-elec.fr/mes_downloads/tutos_arduino/12b8.atelier_arduino_ethernet_tcp_javascript_ajax_canva_can_rgraph.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

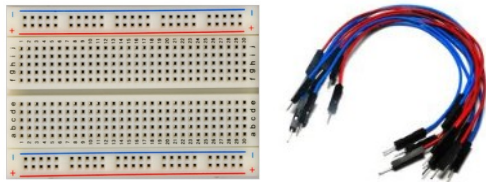


L'espace de développement Arduino associe :

- 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

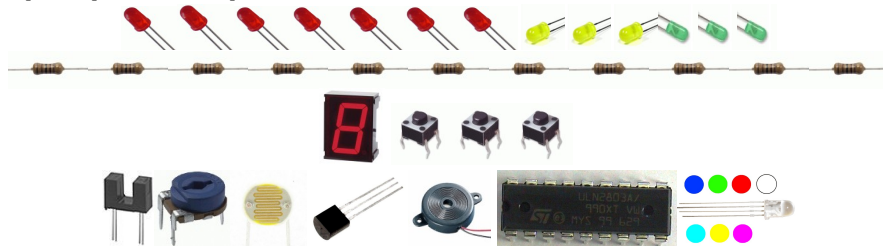


Pour réaliser des montages sans soudure, vous aurez besoin :

- 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

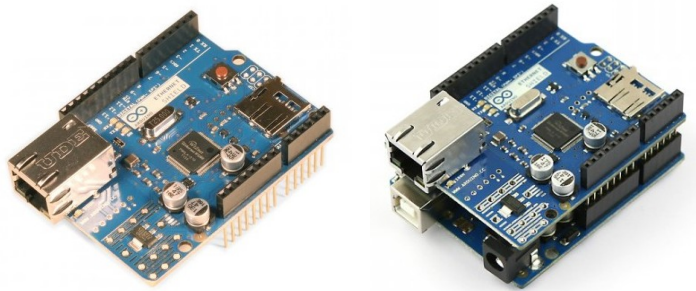
Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- 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.

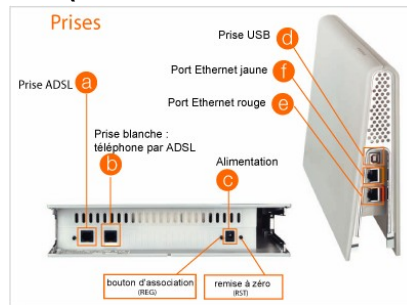
Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 +/- 4) pour communiquer avec Arduino.

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.

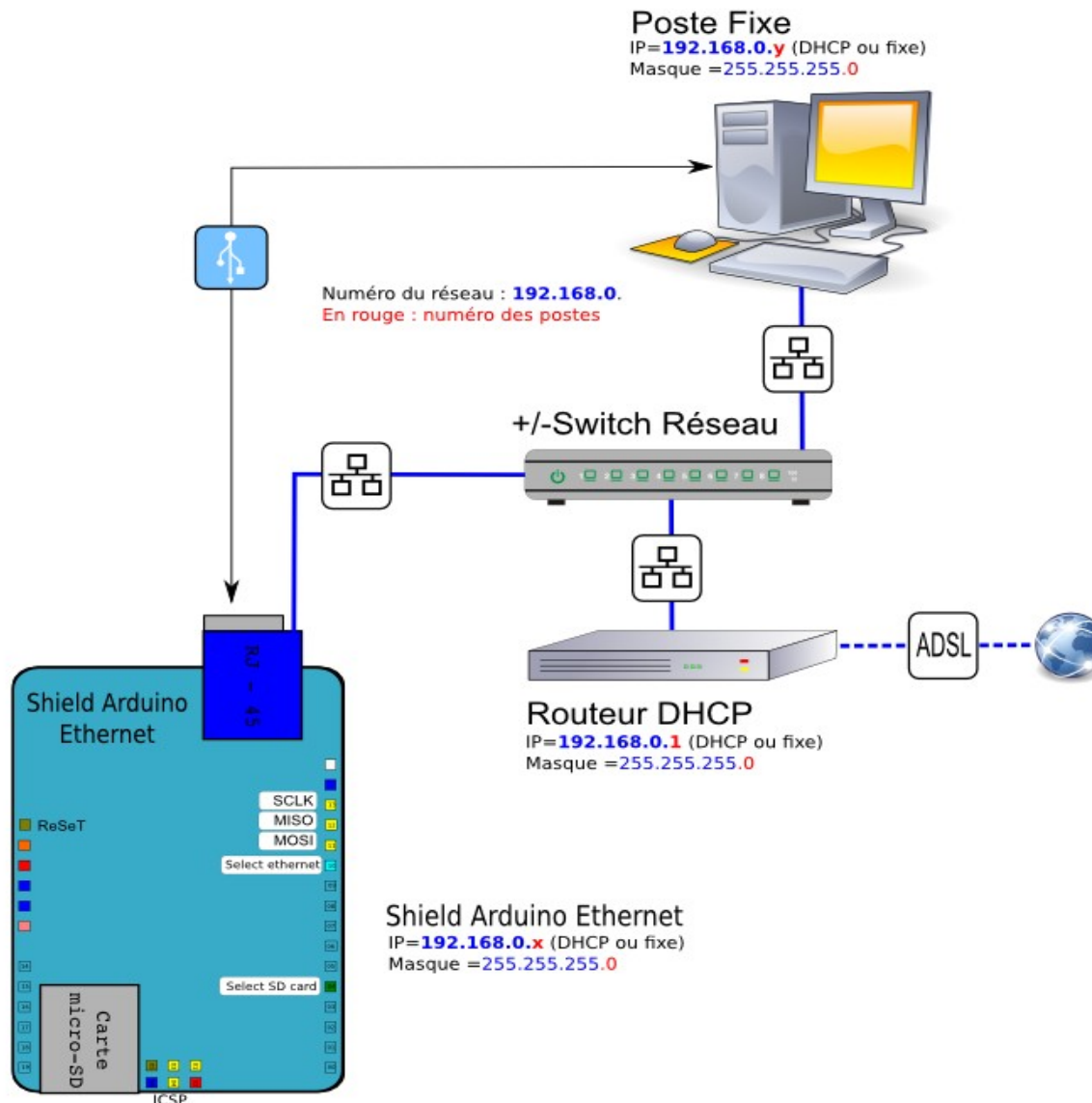
Et d'un poste fixe (PC, Mac, Netbook,...) disposant d'une carte Ethernet !



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

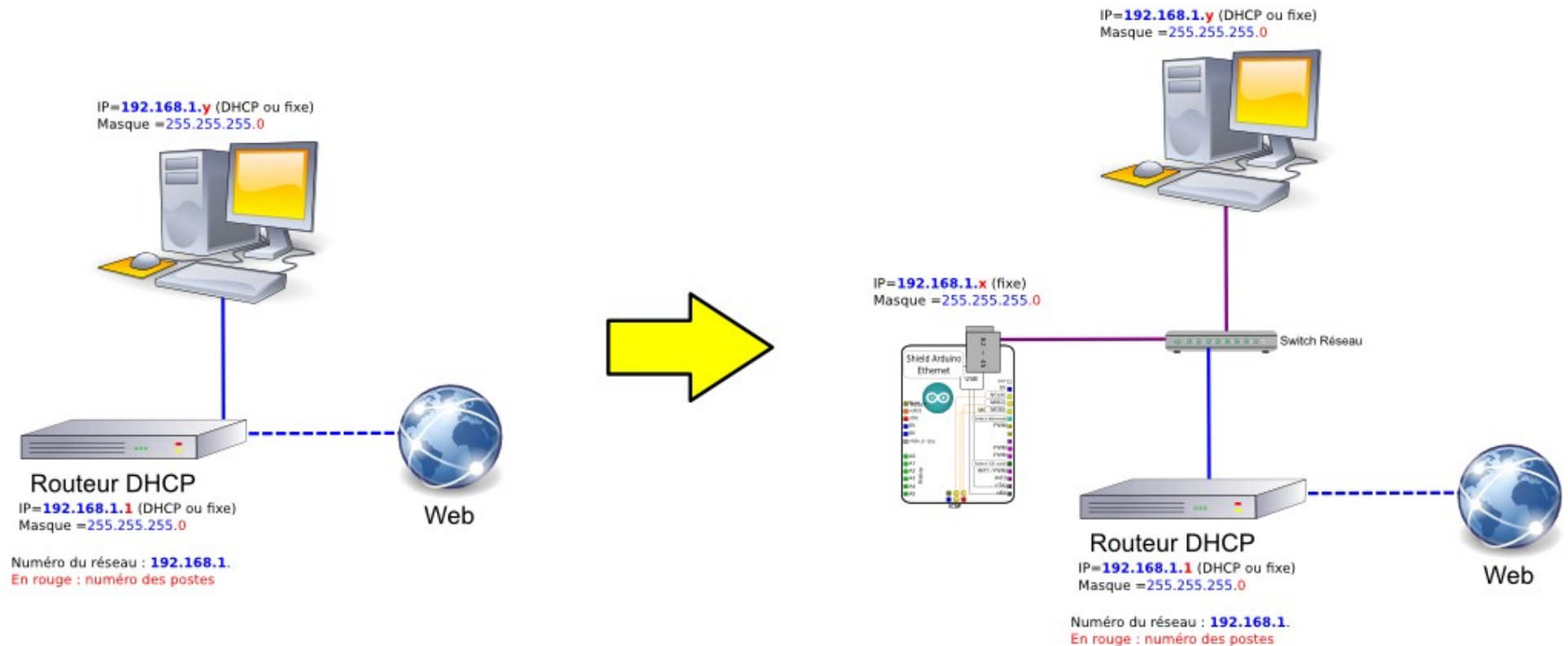
Ceci est très pratique en phase de test et de mise au point, mais une fois la programmation terminée, on pourra bien sûr déconnecter le câble USB.

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. Mémo : 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 : 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.

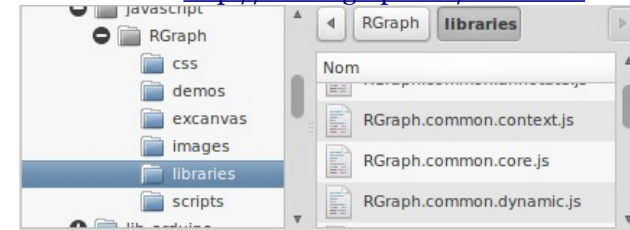


Principe général d'utilisation

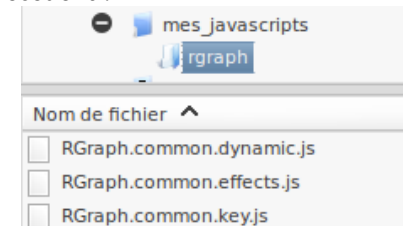
- On commence par installer une fois pour toutes les fichiers javascript de la librairie rgraph dans un répertoire d'un serveur accessible depuis le réseau où se trouve Arduino : ces fichiers contiennent les codes de dessin des éléments graphiques dans un canvas
- Au niveau de la page HTML :
 - on charge 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 :

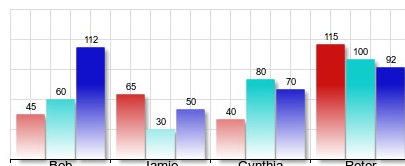
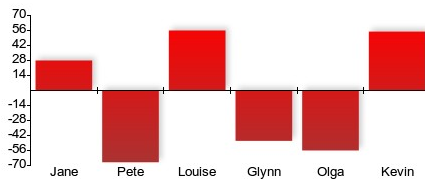
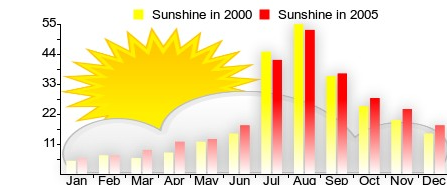


10. Vue d'ensemble des graphiques et éléments analogiques disponibles avec la librairie Javascript utilisée

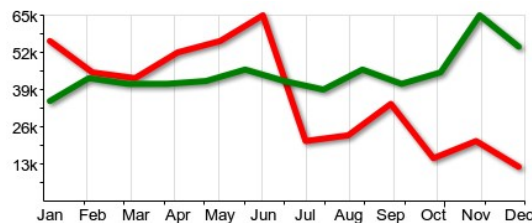
Intro

- 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



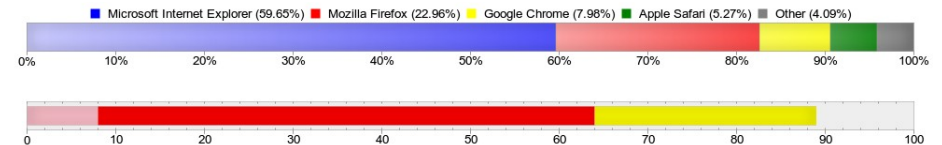
Courbes



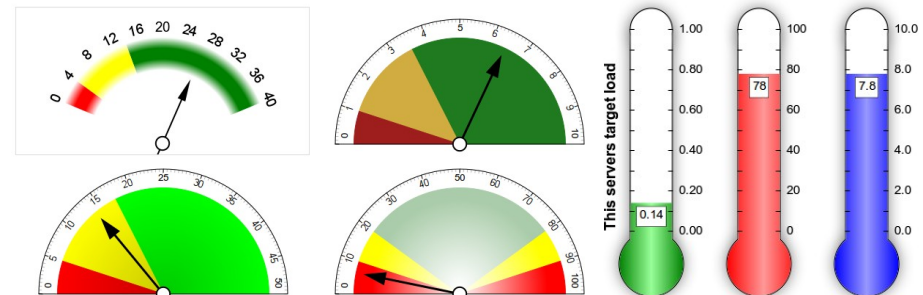
Afficheurs analogiques à aiguille variés



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>

11. Rappel : AJAX : principe et intérêt.

Intro

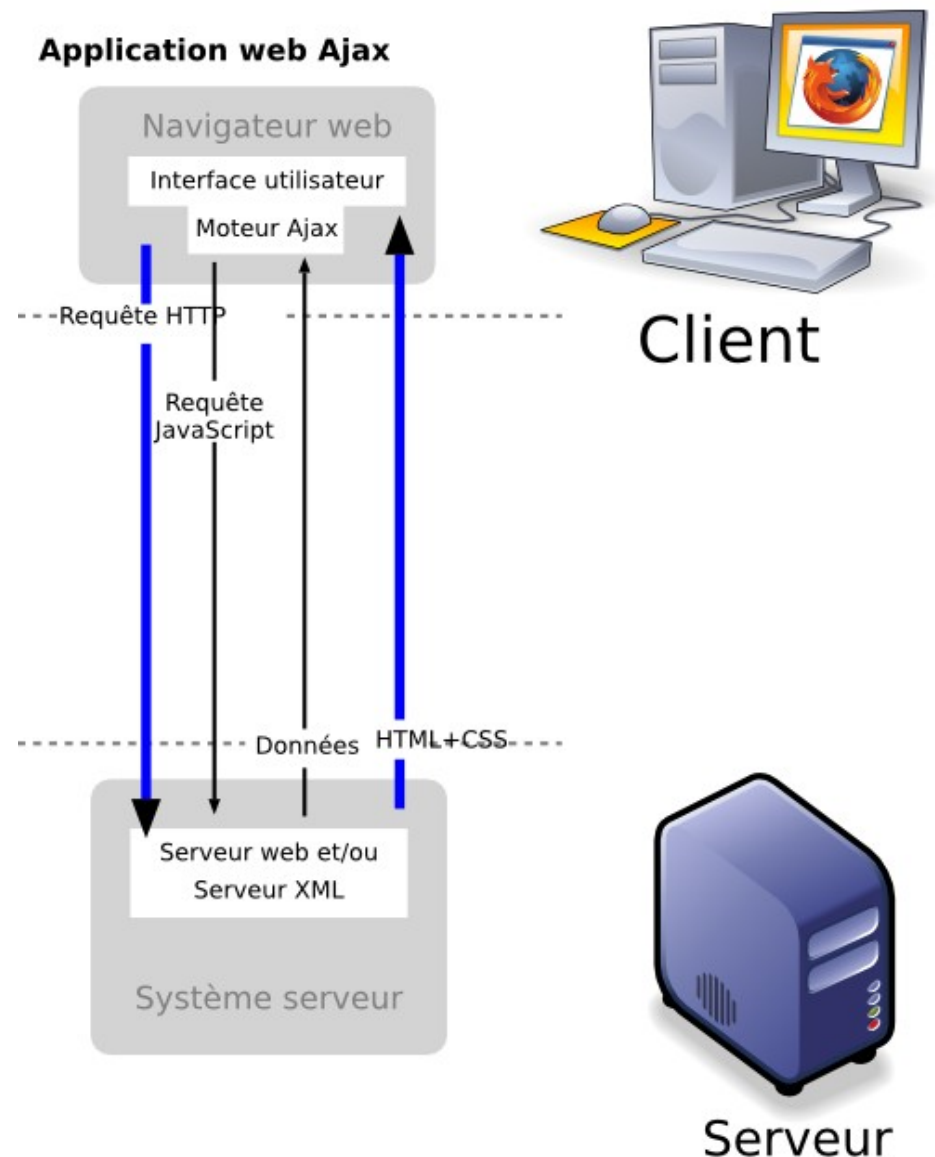
- Dans un tuto précédent, souvenez-vous, nous avons réussi à obtenir un affichage graphique « dynamique » de notre page HTML + Javascript en utilisant la possibilité d'auto-rafraîchissement de la page HTML en ajoutant dans le head une ligne de la forme :

```
<meta http-equiv="refresh" content="3" />
<!-- pour reactualisation 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.



12. 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

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

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

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

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

14. Serveur Arduino : Afficher sous forme graphique à l'aide d'une librairie Javascript en « temps réel » les données en provenance du serveur obtenues par requête Ajax envoyée par le navigateur client.

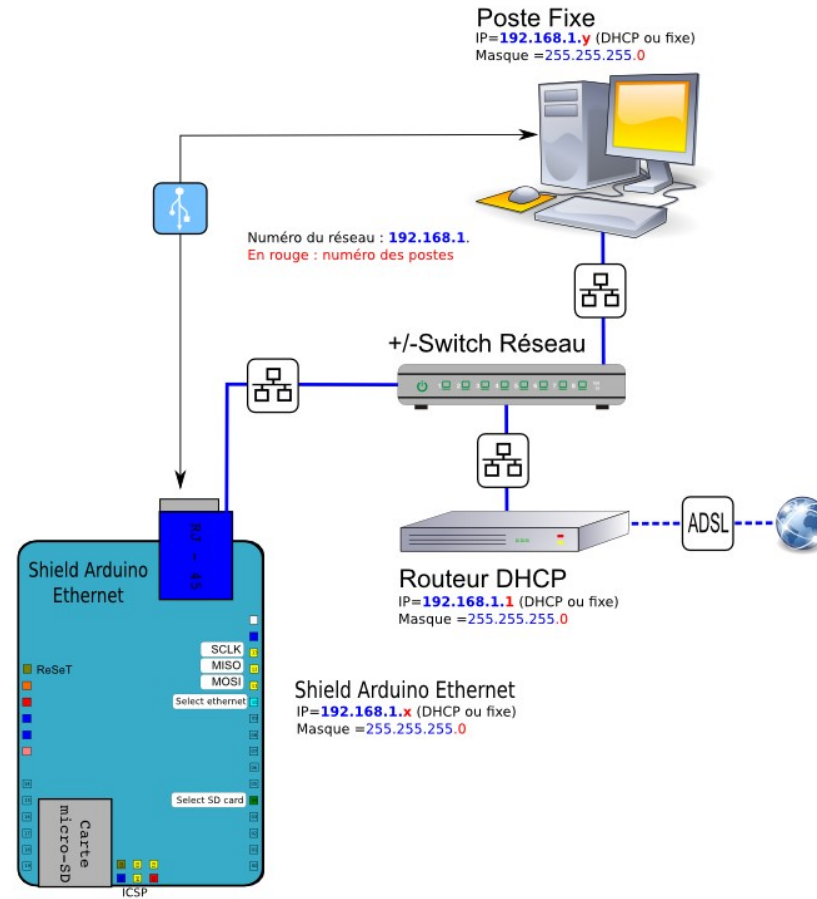
Ce qu'on va faire ici...

- Nous allons reprendre le code du serveur Arduino Ajax déjà vu. Ici, :
 - le serveur enverra une première fois la page complète avec tout le code Javascript gérant les requêtes Ajax, l'affichage graphique, etc...
 - puis, si il reçoit la requête « GET /ajax », il ne renverra qu'un message simple qui sera utilisé côté navigateur client pour modifier la valeur du widget utilisé,
 - la valeur courante reçue sera affichée dans un champ texte.

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 :



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. Reremarquer 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, analyse de la chaîne reçue et envoi de la réponse à la requête Ajax

- Ensuite, tout simplement, on affiche la chaîne reçue
- **La clé de ce programme se trouve ici :**
 - si la requête reçue commence par « GET /ajax », on traitera la requête comme tel et on n'enverra au client qu'une entête http suivie de la chaîne voulue, **ici, le simple résultat de la mesure d'une voie analogique.**

Important : Bien comprendre que la partie « /ajax » de la requête est arbitraire et est définie dans le code Javascript de la page principale : on peut utiliser n'importe quelle chaîne de son choix, en adaptant le code en conséquence. Cela veut dire aussi que l'on peut prévoir autant de requêtes ajax différentes que l'on veut, simplement en modifiant la chaîne utilisée. On pourra donc de la sorte, contrôler des dispositifs côté serveur, etc...

- 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
- Voici donc le contenu de la réponse à une requête « GET /ajax » valide envoyée par le client :

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

//----- analyse si la chaine reçue est une requete GET -----
if (chaîneRecue.startsWith("GET /ajax")) {

    Serial.println (F("Requete GET AJAX valide !"));

    //////////// Réponse HTTP suivie de la chaine de réponse à la requete AJAX //////////

    //-- 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

    int valeur = analogRead(A0); // mesure broche analogique
    client.println(valeur); // envoie la valeur brute seule = réponse à la requete AJAX

    //-- 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(valeur); // envoie la valeur brute seule
} // fin if GET /ajax
```


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

- Si la chaîne reçue n'est pas une requête AJAX, 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 **Connexion: close** indique que la connexion doit être fermée après réception de la réponse.
 - Un saut de ligne précède le message de réponse

```
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 !"));

//----- +/- extraction et analyse de la sous-chaine utile -----

//////////////////// 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("Connexion: 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("Connexion: close"));

Serial.println();
```

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) : Envoi du début et du head (1) de la page HTML de réponse initiale

- 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, nous n'utilisons pas ici le **rafraîchissement automatique de la page** : les données seront reçues lors des requêtes Ajax envoyées par le code Javascript ci-après.

```
//----- 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>Test de requete AJAX </title>")); // titre de la page HTML
```

Fonction **loop()** (4) : Head (2) : Début du code Javascript et chargement des fichiers de la librairie Javascript

- A ce niveau, nous insérons la balise script et nous insérons à l'aide de **println()** successifs le code javascript vu précédemment .
- On commence par inclure le code concernant le chargement de fichiers utiles de la librairie RGraph présentée précédemment. On utilise notamment une fonction qui permet de charger les fichiers en précisant simplement leur et en ayant que le chemin absolu à modifier sur une seule ligne en ce qui concerne le serveur. J'utilise ici la librairie RGraph mise en ligne sur www.mon-club-elec.fr par mesure de simplification. A adapter à votre situation.
- Ici, nous chargeons les fichiers RGraph suivants :
 - common.core qui contient les fonctions de base – obligatoire
 - common.dynamic et common.effect qui vont permettre l'utilisation de l'affichage progressif
 - gauge qui implémente un afficheur à aiguille analogique,
- ce qui nous donne :

```
//===== 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');"));
```

Fonction **loop()** (5) : Head (3) : entête déclarative du code Javascript

- On commence par définir l'ensemble des objets et des variables globales utiles :
 - ici, la variable délai qui fixe le délai entre 2 requêtes Ajax en millisecondes.
 - ainsi que les objets canvas, gauge et champ texte utilisés
 - et une variable de mémorisation de la valeur à afficher,
- ce qui nous donne :

```
//-----> Entete déclarative Javascript <-----  
  
client.println(F("// variables / objets globaux"));  
  
client.println(F("var canvasRGraph= null; "));  
client.println(F("var contextCanvasRGraph = null;"));  
  
//---- objets RGraph --  
client.println(F("var gauge= null; "));  
  
client.println(F("var textInputRGraph=null; "));  
client.println(F("var delai=50;"));  
client.print(F("var val=0;")); // variable pour gauge
```

Fonction **loop()** (6) : Head (4) : Envoi de la fonction javascript appelée au chargement de la page HTML

- Ensuite, suit la fonction appelée lors du chargement initial de la page HTML, sur l'évènement window.onload :
 - on déclare un élément RGraph appelé gauge (afficheur à aiguille) et on le paramètre pour un affichage 0-1023
 - on déclare le canvas et le champs texte à l'aide de la fonction `getElementById()`
 - on initialise également l'objet context du canvas utilisé avec RGraph pour pouvoir accéder aux fonctions de dessin au besoin.
 - ici, on fixe le délai avec la fonction `setTimeout` pour le premier appel de la fonction d'envoi de requête Ajax (appelée ici `requeteAjax`), en passant en paramètre la fonction de gestion de la réponse Ajax (fonction de « callback », appelée ici `drawData`) (voir précédemment pour les explications) :

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

//----- éléments RGraph -----
client.println(F("gauge = new RGraph.Gauge('cvs', 0, 1023, val);")); // création gauge

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

//----- les éléments de la page HTML -----
client.println(F("canvasRGraph = document.getElementById(\"cvs\");")); // canvas RGraph
client.println(F("textInputRGraph= document.getElementById(\"valeurRGraph\");"));
client.println(F("textInputRGraph.value=val;"));

// ----- 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("setTimeout(function () {requeteAjax(drawData);}, delai);"));

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

Voici le dessin de l'afficheur obtenu par défaut :



Bonus : Personnaliser l'afficheur graphique à aiguille

- La librairie RGraph permet de personnaliser l'afficheur utilisé : voici un exemple de code de personnalisation à insérer entre la déclaration de l'objet gauge et l'appel de la fonction Draw(). Je ne fais que reprendre ici le code utilisé dans la démo gauge04 fournie avec la librairie en l'adaptant un peu :

```
// -- tracé des gauges analogiques --
client.println(F("gauge = new RGraph.Gauge('cvs', 0, 1023, val);")); // création gauge
//----- personnalisation de la Gauge -----
client.println(F("gauge.Set('chart.title.top', 'T°');"));
client.println(F("gauge.Set('chart.title.top.size', 'Italic 22');"));
client.println(F("gauge.Set('chart.title.top.font', 'Impact');"));
client.println(F("gauge.Set('chart.title.top.color', 'white');"));
client.println(F("gauge.Set('chart.title.top', 'Temp');"));
client.println(F("gauge.Set('chart.title.bottom.size', 'Italic 14');"));
client.println(F("gauge.Set('chart.title.bottom.font', 'Impact');"));
client.println(F("gauge.Set('chart.title.bottom.color', '#ccc');"));
client.println(F("gauge.Set('chart.title.bottom', String.fromCharCode(0xBA)+'C');"));
client.println(F("gauge.Set('chart.title.bottom.pos', 0.4);"));
client.println(F("gauge.Set('chart.background.color', 'black');"));
client.println(F("gauge.Set('chart.background.gradient', true);"));
client.println(F("gauge.Set('chart.centerpin.color', '#666');"));
client.println(F("gauge.Set('chart.centerpin.color', '#666');"));
client.println(F("gauge.Set('chart.needle.colors', [RGraph.RadialGradient(gauge, 125, 125, 0, 125, 125, 25, 'transparent',
'white'),RGraph.RadialGradient(gauge, 125, 125, 0, 125, 125, 25, 'transparent', '#d66')]);"));
client.println(F("gauge.Set('chart.needle.size', [null, 50]);"));
client.println(F("gauge.Set('chart.text.color', 'white');"));
client.println(F("gauge.Set('chart.tickmarks.big.color', 'white');"));
client.println(F("gauge.Set('chart.tickmarks.medium.color', 'white');"));
client.println(F("gauge.Set('chart.tickmarks.small.color', 'white');"));
client.println(F("gauge.Set('chart.border.outer', '#666');"));
client.println(F("gauge.Set('chart.border.inner', '#333');"));
client.println(F("gauge.Set('chart.colors.ranges', []);"));

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

L'aspect de l'afficheur devient alors :



Fonction **loop()** (5) : Head (4) : envoi de la fonction Javascript de requete Ajax

- La suite du code Javascript de la page est constitué par la définition de la fonction de requête 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 « /ajax » de la requête qui sera reconnue par le code Arduino comme expliqué précédemment.**
 - 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(callback) { "));  
  
client.println(F("var xhr = XMLHttpRequest(); "));  
  
client.println(F("xhr.open(\"GET\", \"/ajax\", true);"));  
client.println(F("xhr.send(null);"));  
  
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"));  
  
client.println(F("} // fin fonction requeteAjax"));
```

Fonction **loop()** (6) : Head (5) : envoi de la fonction Javascript de gestion des données reçues du serveur et fin de script

- Ensuite on envoie la fameuse fonction de « callback » qui sera appelée une fois la réponse à la requête Ajax reçue : [cette fonction reçoit en paramètre la chaîne reçue du serveur](#).
- Cette fonction est le cœur de notre programme : c'est à ce niveau que l'on utilise la valeur reçue en provenance du serveur pour réaliser l'affichage graphique en réglant la position de l'aiguille. On a ici 2 possibilités :
 - soit un affichage direct avec effacement préalable du canvas
 - soit un affichage progressif, qui nécessite d'utiliser un délai de requête long (au moins 1000ms) mais qui donne un mouvement doux assez sympa. Se reporter au code pour les détails.
- Puis, avant de terminer cette fonction de gestion de la réponse, ne pas oublier de réactiver le délai avec la fonction `setTimeout()` pour un nouvel envoi d'une requête Ajax au serveur

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

client.println(F("textInputRGraph.value=Number(stringDataIn);"));

client.println(F("gauge.value = Number(stringDataIn);")); // fixe valeur gauge

//-- si mise à jour directe --
client.println(F("contextCanvasRGraph.fillStyle = \"rgb(255,255,200)\";"));
client.println(F("contextCanvasRGraph.fillRect (0, 0, canvasRGraph.width, canvasRGraph.height);"));
client.println(F("gauge.Draw();")); // MAJ la gauge

//-- si mise à jour progressive --
//client.println(F("RGraph.Effects.Gauge.Grow(gauge);")); // fixe valeur gauge progressivement - utiliser delai long

// réinitialise delai

//client.println(F("alert(stringDataIn);"));
client.println(F("setTimeout(function () {requeteAjax(drawData);}, delai);"));

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

// -----

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

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

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

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

- De la même façon, par un jeu de **println()**, on envoie la suite du code HTML, c'est à dire ici le body de la page HTML.
- Ici, on inclut :
 - le canvas utilisé avec RGraph
 - un champ texte
 - et un message texte simple
- Suivent les balises de clôture de la page web

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

client.println(F("<canvas id=\"cvs\" width=\"250\" height=\"250\">[No canvas support]</canvas>")); // Canvas gauge RGraph
client.println(F("<br/>"));

client.println(F("Valeur (0-1023) =<input type=\"text\" id=\"valeurRGraph\" />"));
client.println(F("<br/>"));
client.println(F("Serveur Arduino : Test RGraph avec reponse de requete Ajax "));
client.println(F("<br/>"));

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

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

} // fin if GET
```

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

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

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

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

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

} // --- fin if client connected

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

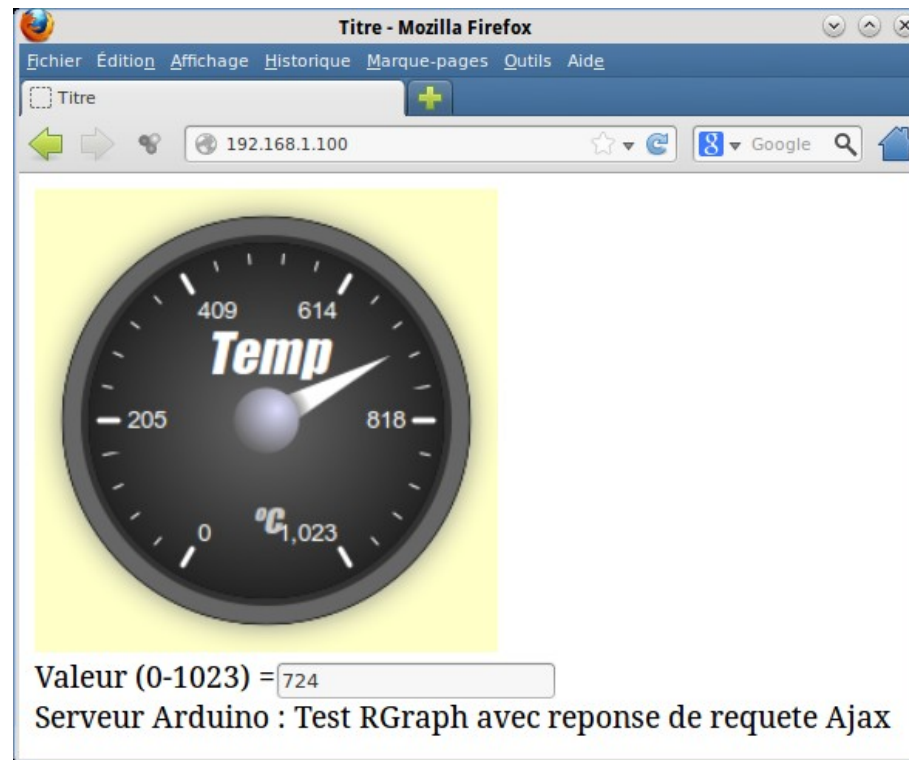
} // fin de la fonction loop()
```



Ici, au final, il s'agit essentiellement une adaptation de codes vus précédemment...
Si vous êtes perdus, il vaut mieux retravailler un peu les tutos précédents consacrés à RGraph, Javascript et Ajax...

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, l'afficheur à aiguille s'afficher et varier au gré des variations de la tension sur la broche Ao de la carte Arduino. La valeur s'affiche également dans le champ texte associé.



Cool non ? Et remarquer que la page ne se rafraîchit pas : **les échanges serveur Arduino ↔ navigateur client se passent en arrière plan.**

Cette fois, ça commence sérieusement à ressembler à quelque chose d'opérationnel, d'autant que **la position de l'aiguille est rafraîchie plusieurs fois par seconde.**

Essayer d'accéder au serveur Arduino avec une tablette sous Android : vous verrez que ça fonctionne parfaitement bien !!

15. 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>

16. 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

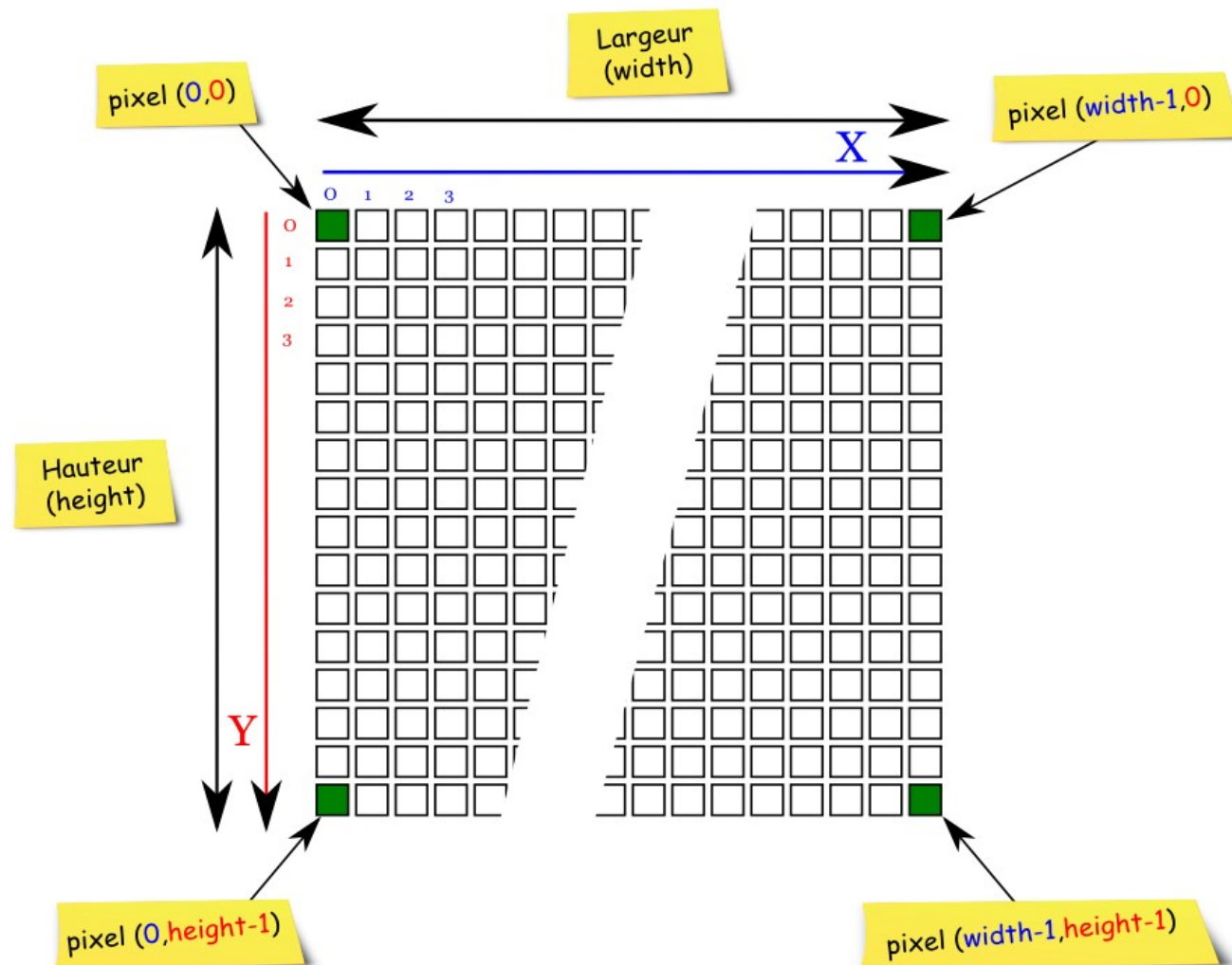
- Une fois que l'on dispose de l'objet Context d'accès aux fonctions de dessin du Canvas, on va pouvoir passer à l'action : **en fait, à ce stade, c'est tout un nouveau monde de possibilités qui s'ouvre à vous !** Un peu à la façon processing pour ceux qui connaissent...
- Tout d'abord, on peut modifier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

17. Objet Canvas : système de coordonnées

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



18. Serveur Arduino : Afficher sous forme graphique à l'aide d'une librairie Javascript et sous forme de courbe en « temps réel » les données en provenance du serveur obtenues par requête Ajax envoyée par le navigateur client.



Bon, je considère ici que vous êtes à l'aise : je continue avec un mix du programme de courbe « temps-réel » déjà vu dans le tuto précédent, combiné à l'utilisation d'un widget RGraph comme nous venons de le faire, le tout avec Ajax !

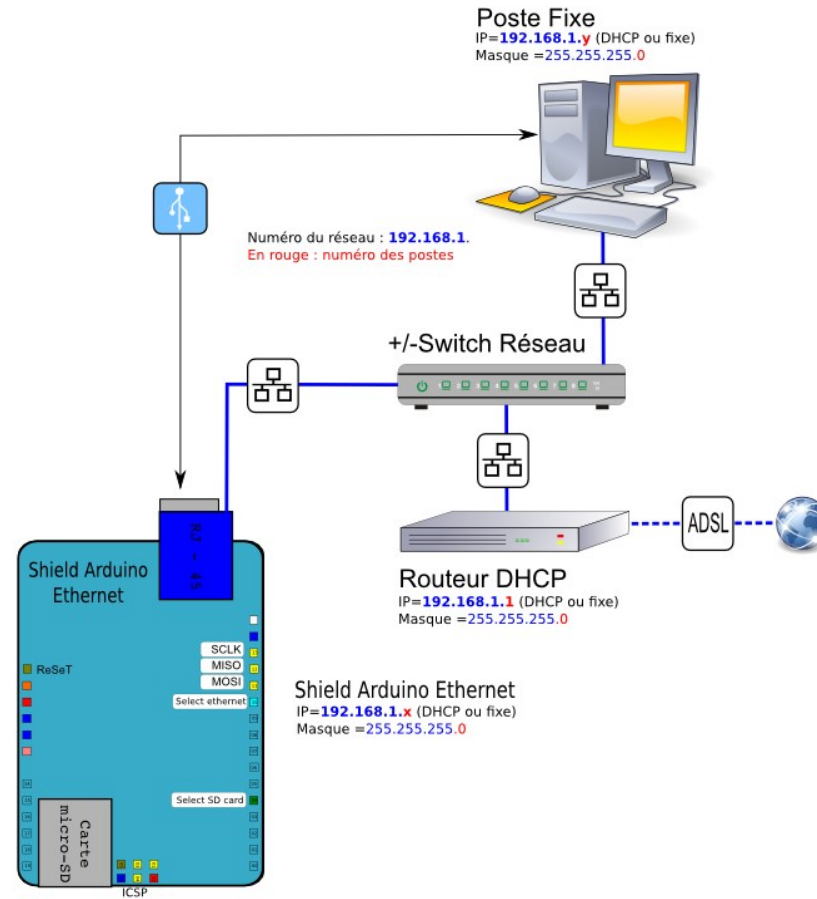
Ce qu'on va faire ici...

- Nous allons reprendre un mix des codes du serveur Arduino Ajax déjà vu, à la fois utilisation de RGraph et d'une courbe « temps réel ». Ici, :
 - le serveur enverra une première fois la page complète avec tout le code Javascript gérant les requêtes Ajax, l'affichage graphique, etc...
 - puis, si il reçoit la requête « GET /ajax », il ne renverra qu'un message simple qui sera utilisé côté navigateur client à la fois :
 - pour modifier la valeur du widget utilisé,
 - et ajouter un nouveau point à la courbe
 - la valeur courante reçue sera affichée dans un champ texte, ainsi que les coordonnées du point courant de la courbe.

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 :



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, analyse de la chaîne reçue et envoi de la réponse à la requête Ajax

- Ensuite, tout simplement, on affiche la chaîne reçue
- **La clé de ce programme se trouve ici :**
 - si la requête reçue commence par « GET /ajax », on traitera la requête comme tel et on n'enverra au client qu'une entête http suivie de la chaîne voulue, **ici, le simple résultat de la mesure d'une voie analogique.**

Important : Bien comprendre que la partie « /ajax » de la requête est arbitraire et est définie dans le code Javascript de la page principale : on peut utiliser n'importe quelle chaîne de son choix, en adaptant le code en conséquence. Cela veut dire aussi que l'on peut prévoir autant de requêtes ajax différentes que l'on veut, simplement en modifiant la chaîne utilisée. On pourra donc de la sorte, contrôler des dispositifs côté serveur, etc...

- 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
- Voici donc le contenu de la réponse à une requête « GET /ajax » valide envoyée par le client :

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

//----- analyse si la chaine reçue est une requete GET -----
if (chaîneRecue.startsWith("GET /ajax")) {

    Serial.println (F("Requete GET AJAX valide !"));

    //////////// Réponse HTTP suivie de la chaine de réponse à la requete AJAX //////////

    //-- 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

    int valeur = analogRead(A0); // mesure broche analogique
    client.println(valeur); // envoie la valeur brute seule = réponse à la requete AJAX

    //-- 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(valeur); // envoie la valeur brute seule
} // fin if GET /ajax
```

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

- Si la chaîne reçue n'est pas une requête AJAX, 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 **Connexion: close** indique que la connexion doit être fermée après réception de la réponse.
 - Un saut de ligne précède le message de réponse

```
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 !"));

//----- +/- extraction et analyse de la sous-chaine utile -----

//////////////////// 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("Connexion: 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("Connexion: close"));

Serial.println();
```

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) : Envoi du début et du head (1) de la page HTML de réponse initiale

- 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, nous n'utilisons pas ici le **rafraîchissement automatique de la page** : les données seront reçues lors des requêtes Ajax envoyées par le code Javascript ci-après.

```
//----- 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>Test de requete AJAX </title>")); // titre de la page HTML
```

Fonction **loop()** (4) : Head (2) : Début du code Javascript et chargement des fichiers de la librairie Javascript

- A ce niveau, nous insérons la balise script et nous insérons à l'aide de **println()** successifs le code javascript vu précédemment .
- On commence par inclure le code concernant le chargement de fichiers utiles de la librairie RGraph présentée précédemment. On utilise notamment une fonction qui permet de charger les fichiers en précisant simplement leur et en ayant que le chemin absolu à modifier sur une seule ligne en ce qui concerne le serveur. J'utilise ici la librairie RGraph mise en ligne sur www.mon-club-elec.fr par mesure de simplification. A adapter à votre situation.
- Ici, nous chargeons les fichiers RGraph suivants :
 - common.core qui contient les fonctions de base – obligatoire
 - common.dynamic et common.effect qui vont permettre l'utilisation de l'affichage progressif
 - gauge qui implémente un afficheur à aiguille analogique,
- ce qui nous donne :

```
//===== 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');"));
```

Fonction **loop()** (5) : Head (3) : entête déclarative du code Javascript

- On commence par définir l'ensemble des objets et des variables globales utiles :
 - ici, la variable délai qui fixe le délai entre 2 requêtes Ajax en millisecondes.
 - ainsi que les objets canvas, gauge et champ texte utilisés
 - et une variable de mémorisation de la valeur à afficher,
 - et les variables utiles pour le tracé de la courbe
- ce qui nous donne :

```
//-----> Entete déclarative Javascript <-----  
  
client.println(F("// variables / objets globaux"));  
client.println(F("var canvas= null; "));  
client.println(F("var contextCanvas = null;"));  
  
client.println(F("var canvasRGraph= null; "));  
client.println(F("var contextCanvasRGraph = null;"));  
  
//---- objets RGraph --  
client.println(F("var gauge= null; "));  
  
client.println(F("var textInputX=null; "));  
client.println(F("var textInputY=null; "));  
client.println(F("var textInputRGraph=null; "));  
  
client.println(F("var delai=50;"));  
//client.println(F("var compt=0;"));  
client.println(F("var x=0;"));  
client.println(F("var y=0;"));  
client.println(F("var xo=0;"));  
client.println(F("var yo=0;"));  
  
client.print(F("var val=0;")); // variable pour gauge
```

Il est important de déclarer tous les objets utilisés ensuite avant les fonctions pour qu'ils aient une portée globale dans le reste du code.

Fonction **loop()** (6) : Head (4) : Envoi de la fonction javascript appelée au chargement de la page HTML : la gauge analogique

- Ensuite, suit la fonction appelée lors du chargement initial de la page HTML, sur l'évènement window.onload :
 - on déclare un élément RGraph appelé gauge (afficheur à aiguille) et on le paramètre pour un affichage 0-1023
 - on personnalise au besoin l'aspect de l'afficheur à aiguille utilisé :

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("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  
  
//----- personnalisation de la Gauge -----  
client.println(F("gauge.Set('chart.title.top', 'T°');"));  
client.println(F("gauge.Set('chart.title.top.size', 'Italic 22');"));  
client.println(F("gauge.Set('chart.title.top.font', 'Impact');"));  
client.println(F("gauge.Set('chart.title.top.color', 'white');"));  
client.println(F("gauge.Set('chart.title.top', 'Temp');"));  
client.println(F("gauge.Set('chart.title.bottom.size', 'Italic 14');"));  
client.println(F("gauge.Set('chart.title.bottom.font', 'Impact');"));  
client.println(F("gauge.Set('chart.title.bottom.color', '#ccc');"));  
client.println(F("gauge.Set('chart.title.bottom', String.fromCharCode(0xBA)+'C');"));  
client.println(F("gauge.Set('chart.title.bottom.pos', 0.4);"));  
client.println(F("gauge.Set('chart.background.color', 'black');"));  
client.println(F("gauge.Set('chart.background.gradient', true);"));  
client.println(F("gauge.Set('chart.centerpin.color', '#666');"));  
client.println(F("gauge.Set('chart.centerpin.color', '#666');"));  
client.println(F("gauge.Set('chart.needle.colors', [RGraph.RadialGradient(gauge, 125, 125, 0, 125, 125, 25, 'transparent',  
'white'),RGraph.RadialGradient(gauge, 125, 125, 0, 125, 125, 25, 'transparent', '#d66')]);"));  
client.println(F("gauge.Set('chart.needle.size', [null, 50]);"));  
client.println(F("gauge.Set('chart.text.color', 'white');"));  
client.println(F("gauge.Set('chart.tickmarks.big.color', 'white');"));  
client.println(F("gauge.Set('chart.tickmarks.medium.color', 'white');"));  
client.println(F("gauge.Set('chart.tickmarks.small.color', 'white');"));  
client.println(F("gauge.Set('chart.border.outer', '#666');"));  
client.println(F("gauge.Set('chart.border.inner', '#333');"));  
client.println(F("gauge.Set('chart.colors.ranges', []);"));  
  
//--- dessin de la gauge ---  
client.println(F("gauge.Draw();"));
```

Fonction **loop()** (6) : Head (4) : Envoi de la fonction javascript appelée au chargement de la page HTML : la courbe

- Ensuite,
 - on déclare les canvas utilisés et les champs texte à l'aide de la fonction `getElementById()`
 - on initialise également l'objet context de chaque canvas utilisé pour pouvoir accéder aux fonctions de dessin au besoin.
 - on configure les paramètres graphiques de la courbe,
 - enfin, on fixe le délai avec la fonction `setTimeout` pour le premier appel de la fonction d'envoi de requête Ajax (appelée ici `requeteAjax`), en passant en paramètre la fonction de gestion de la réponse Ajax (fonction de « callback », appelée ici `drawData`) (voir précédemment pour les explications) :

```
//----- canvas courbe -----
client.println(F("canvas = document.getElementById(\"nomCanvas\");"));
client.println(F("canvas.width = 360;"));
client.println(F("canvas.height = 300;"));

//----- canva RGraph ---
client.println(F("canvasRGraph = document.getElementById(\"cvs\");")); // canvas RGraph

//---- champs texte ----
client.println(F("textInputX= document.getElementById(\"valeurX\");"));
client.println(F("textInputY= document.getElementById(\"valeurY\");"));
client.println(F("textInputRGraph= document.getElementById(\"valeurRGraph\"); "));

client.println(F("textInputX.value=x; "));
client.println(F("textInputY.value=y;"));
client.println(F("textInputRGraph.value=val;"));

// ----- initialisation canvas RGraph ----
client.println(F("if (canvasRGraph.getContext){"));
client.println(F("contextCanvasRGraph = canvasRGraph.getContext(\"2d\");"));
client.println(F("} // fin si canvas existe"));

// ----- initialisation canvas courbe -----
client.println(F("if (canvas.getContext){"));
client.println(F("contextCanvas = canvas.getContext(\"2d\");"));

// carre de la taille du canvas
client.println(F("contextCanvas.fillStyle = \"rgb(255,255,200)\";"));
client.println(F("contextCanvas.fillRect (0, 0, canvas.width, canvas.height);"));

// position initiale - dessine 1 pixel
client.println(F("contextCanvas.fillStyle = \"rgb(0,0,255)\";"));
client.println(F("contextCanvas.fillRect (x,canvas.height-y-1, 1,1);"));

// parametres graphique
client.println(F("contextCanvas.lineWidth=1;"));
client.println(F("contextCanvas.strokeStyle = \"rgb(0,0,255)\";"));

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

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

// intervalle de rafraichissement commun
client.println(F("setTimeout(function () {requeteAjax(drawData);}, delai);"));

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


Fonction **loop()** (5) : Head (4) : envoi de la fonction Javascript de requete Ajax

- La suite du code Javascript de la page est constitué par la définition de la fonction de requête 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 « /ajax » de la requête qui sera reconnue par le code Arduino comme expliqué précédemment.**
 - 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(callback) { ");  
  
  client.println(F("var xhr = XMLHttpRequest();"));  
  
  client.println(F("xhr.open(\"GET\", \"/ajax\", true);"));  
  client.println(F("xhr.send(null);"));  
  
  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"));  
  
client.println(F("} // fin fonction requeteAjax"));
```

Fonction **loop()** (6) : Head (5) : envoi de la fonction Javascript de gestion des données reçues : dessin de la courbe

- 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](#).
- Cette fonction est le cœur de notre programme :
 - dans un premier temps, nous réalisons l'ajout d'un point à courbe « temp-réel » selon un script présenté précédemment,
 - le canvas est réinitialisé si on dépasse la largeur du canvas :

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

// --- dessin courbe ---
client.println(F("if (contextCanvas!=null) {"));

// -- coordonnees x,y courantes
client.println(F("x=x+5;")); // pas de 5 pixels à la fois
client.println(F("if (x>canvas.width)x=0;"));

client.println(F("y=Number(stringDataIn)")); // récupère la valeur chiffrée à partir chaîne reçue
client.println(F("y=y/1023*(canvas.height-1)")); // y mappée sur la hauteur canvas

// -- trace courbe --
// RAZ courbe si x==0
client.println(F("if (x==0) {"));
  client.println(F("contextCanvas.moveTo(x,y);"));
  client.println(F("contextCanvas.fillStyle = \"rgb(255,255,200)\";"));
  client.println(F("contextCanvas.fillRect (0, 0, canvas.width, canvas.height);"));
  client.println(F("x=0;"));
client.println(F("} // fin if x==0 "));

// sinon trace ligne point courant
client.println(F("else {"));
  client.println(F("contextCanvas.beginPath(); "));
  client.println(F("contextCanvas.moveTo(xo,canvas.height-yo-1);"));
  client.println(F("contextCanvas.lineTo(x,canvas.height-y-1);"));
  client.println(F("contextCanvas.closePath(); "));
  client.println(F("contextCanvas.stroke();"));

  client.println(F("textInputX.value=x;"));
  client.println(F("textInputY.value=Math.round(y);"));

  client.println(F("xo=x;"));
  client.println(F("yo=y;"));

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

client.println(F("} // fin if context !=null "));
// -- fin dessin courbe --
```

Fonction **loop()** (6) : Head (5) : fonction Javascript de gestion des données reçues : affichage de la gauge et fin de script

- Toujours au sein de la fameuse fonction de « callback » qui est appelée une fois la réponse à la requête Ajax reçue , nous mettons à jour l'afficheur à aiguille en utilisant la valeur reçue en provenance du serveur . On a ici 2 possibilités :
 - soit un affichage direct avec effacement préalable du canvas
 - soit un affichage progressif, qui nécessite d'utiliser un délai de requête long (au moins 1000ms) mais qui donne un mouvement doux assez sympa. Se reporter au code pour les détails.
- Puis, avant de terminer cette fonction de gestion de la réponse, ne pas oublier de réactiver le délai avec la fonction **setTimeout()** pour un nouvel envoi d'une requête Ajax au serveur

```
//----- positionne gauge RGraph ----
client.println(F("textInputRGraph.value=Number(stringDataIn);"));

client.println(F("gauge.value = Number(stringDataIn);")); // fixe valeur gauge

//-- si mise à jour directe --
client.println(F("contextCanvasRGraph.fillStyle = \"rgb(255,255,255)\";"));
client.println(F("contextCanvasRGraph.fillRect (0, 0, canvasRGraph.width, canvasRGraph.height);")); // efface le canva avant retrace
client.println(F("gauge.Draw();")); // MAJ la gauge

//-- si mise à jour progressive --
//client.println(F("RGraph.Effects.Gauge.Grow(gauge);")); // fixe valeur gauge progressivement - utiliser delai long 1000ms ou plus

// réinitialise delai

//client.println(F("alert(stringDataIn);"));
client.println(F("setTimeout(function () {requeteAjax(drawData);}, delai);"));

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

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

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

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

- De la même façon, par un jeu de **println()**, on envoie la suite du code HTML, c'est à dire ici le body de la page HTML.
- Ici, on inclut :
 - le canvas utilisé avec RGraph et le canvas utilisé pour la courbe
 - deux champs textes associés à la courbe et un champ texte associé au widget graphique
 - et un message texte simple
- 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("<canvas id=\"cvs\" width=\"250\" height=\"250\">[No canvas support]</canvas>")); // Canvas gauge RGraph
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("Valeur (0-1023) =<input type=\"text\" id=\"valeurRGraph\" />"));
client.println(F("<br/>"));
client.println(F("Serveur Arduino : Test courbe et gauge RGraph avec reponse de requete Ajax "));
client.println(F("<br/>"));

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

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

} // fin if GET
```

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

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

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

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

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

} // --- fin if client connected

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

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

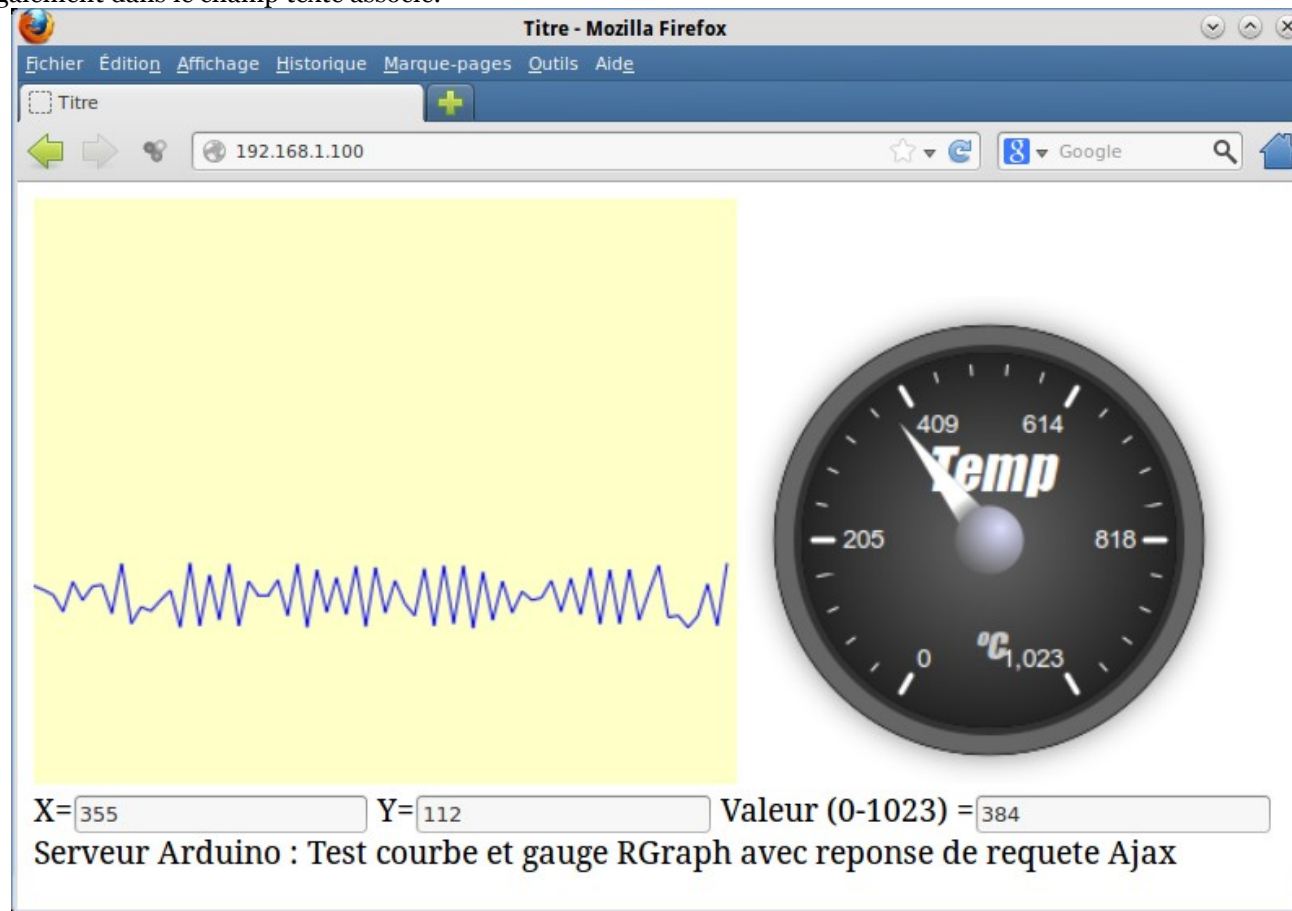


Si vous êtes arrivés jusqu'ici, c'est que vous êtes désormais à l'aise avec Http, Html, Javascript, Ajax, RGraph et Arduino ! Bravo... !!
Vous venez quand même d'avaler un code de quelques centaines de ligne... donc c'est pas mal !
C'est désormais tout un petit univers qui s'ouvre à vous : celui des interfaces temps réel sur réseau de type web !

Au fait, elle donne quoi notre interface ??

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, :
 - la courbe se trace progressivement, reflétant l'état de la broche analogique mesurée et les champs texte de la courbe indiquent les coordonnées du point courant de la courbe,
 - l'afficheur à aiguille doit s'afficher, la position de l'aiguille variant au gré des variations de la tension sur la broche Ao de la carte Arduino. La valeur s'affiche également dans le champ texte associé.



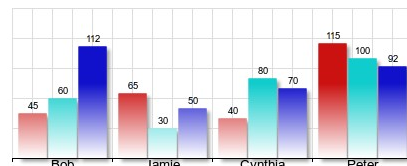
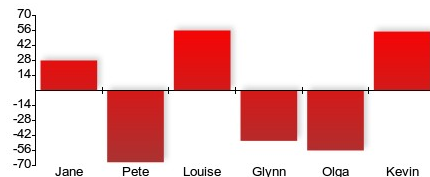
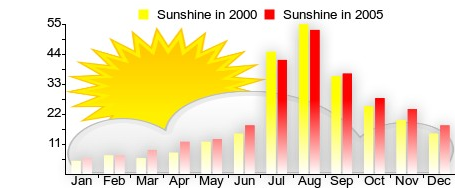
Cool non ? Maintenant, à vous de jouer : mixez les widgets RGraph, utilisez plusieurs voies en même temps... !

Et je vous rappelle les nombreuses possibilités de la librairie RGraph qui vous permettront d'adapter simplement l'exemple proposé ici !

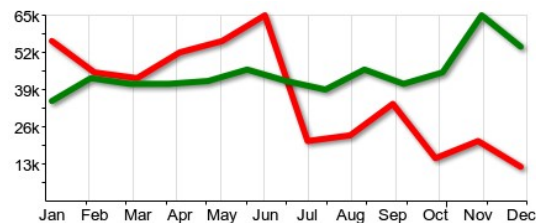
La librairie RGraph

- 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



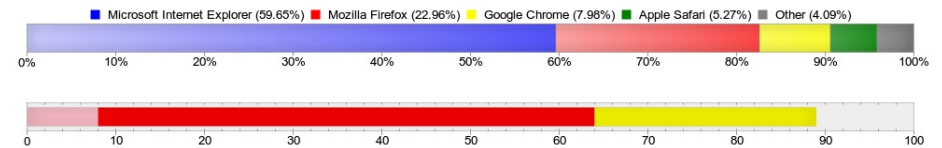
Courbes



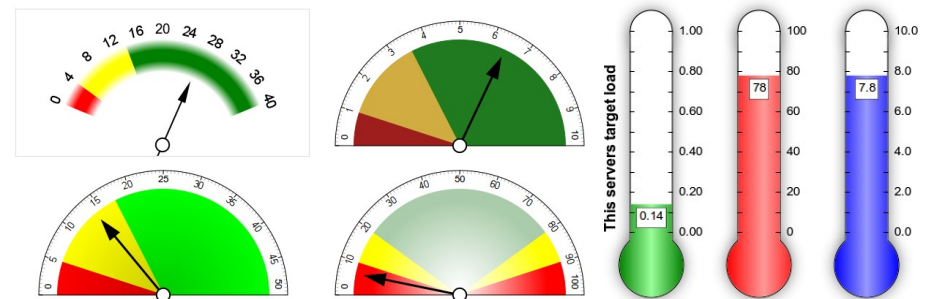
Afficheurs analogiques à aiguille variés



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>

19. 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

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

- d'utiliser une librairie javascript permettant des affichages graphiques élaborés
- de mettre en place une interface graphique analogique élaborées côté navigateur client

Table des matières

Créer un serveur HTML+Javascript avec Arduino et afficher des données reçues « en temps réel » par requêtes AJAX sous forme de courbes et de widgets dans des canvas à l'aide d'une librairie javascript dédiée.

- 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 |
- Mémo : Syntaxe de base du langage Javascript |
- Rappel : Ecrire un script Javascript intégré dans une page HTML |
- 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 |
- Rappel : AJAX : principe et intérêt. |
- Rappel : Javascript : L'objet XMLHttpRequest et son utilisation |
- Rappel : Javascript : Code type de gestion d'une requête par XMLHttpRequest |
- Serveur Arduino : Afficher sous forme graphique à l'aide d'une librairie Javascript en « temps réel » les données en provenance du serveur obtenues par requête Ajax envoyée par le navigateur client. |
- Rappel : HTML : Présentation de l'objet canvas |
- Javascript : Rappel : Les fonctions essentielles de l'objet canvas |
- Objet Canvas : système de coordonnées |
- Serveur Arduino : Afficher sous forme graphique à l'aide d'une librairie Javascript et sous forme de courbe en « temps réel » les données en provenance du serveur obtenues par requête Ajax envoyée par 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