

Utiliser la carte d'extension (shield) Ethernet avec Arduino et créer un serveur web Arduino sur réseau local.



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 :

- de monter un réseau local utilisant Arduino
- d'apprendre les rudiments du protocole HTTP et d'écrire un programme Arduino utilisant ce protocole
- d'apprendre les rudiments du protocole HTML et d'écrire un programme Arduino générant une page HTML valide
- d'apprendre à accéder au serveur Arduino depuis un poste fixe sur le réseau local

... afin d'être en mesure de créer un serveur web basique avec le couple Arduino + shield Ethernet.

Remarque

Dans cet atelier les notions abordées vont être nombreuses, notamment en ce qui concerne le protocole de communication HTTP, l'écriture d'une page web en HTML. Le sujet est passionnant mais aussi potentiellement déroutant pour le néophyte. Je vais tenter de vous présenter tout ça le plus simplement possible, mais soyez prévenus : il va falloir prendre son temps pour tout bien comprendre, surtout si tout cela est nouveau pour vous. La bonne nouvelle : à la fin de cet atelier, vous serez capable d'utiliser Arduino pour créer un simple serveur web local ! Une fois que vous saurez le faire en local, vous pourrez facilement le rendre accessible depuis internet.

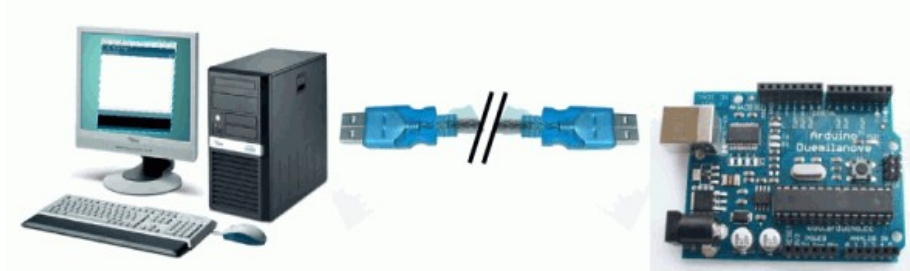


Prêt ? C'est parti !

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

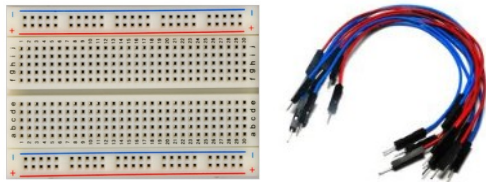


L'espace de développement Arduino associe :

- 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

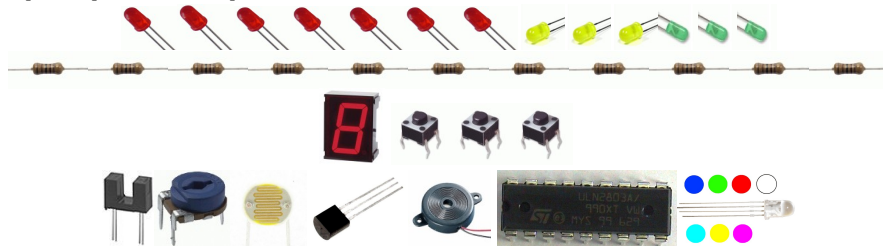


Pour réaliser des montages sans soudure, vous aurez besoin :

- 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

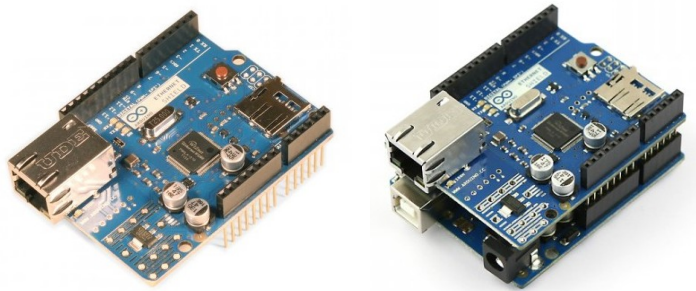
Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- 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.

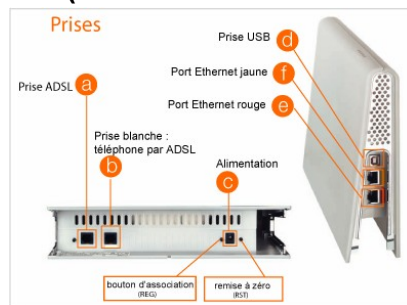
Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 +/- 4) pour communiquer avec Arduino.

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 « mult prises » RJ45.

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

4. Matériel spécifique nécessaire pour cet atelier (suite)

D'un ou 2 câbles réseau RJ45



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

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

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

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

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

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



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

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

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

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

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



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

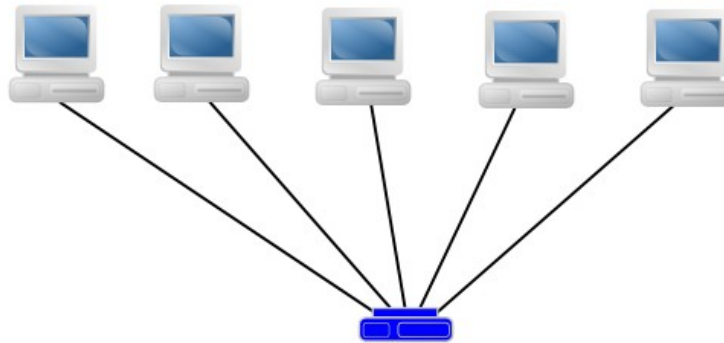
Votre poste peut-être sous Windows, Mac OS X ou Gnu/Linux, peu importe. Vous pouvez utiliser indifféremment un PC de bureau, un netbook ou un portable.

5. Rappel : structure d'un réseau local et matériel nécessaire

Structure générale

Techniquement, un réseau local type va avoir la même structure quelque soit la technique de connexion utilisée (filaire ou wifi) :

- l'un des postes va être le « meneur du jeu » : on l'appelle le routeur (en bleu sur le schéma). C'est lui qui :
 - fixe le numéro du réseau (couleur du maillot)
 - attribue les numéros individuels des postes (les numéros des joueurs), (rôle de serveur DHCP)
 - voit tous les postes
 - et distribue les messages (rôle de commutateur / switch sur le réseau local et/ou rôle de routeur si connexion au réseau extérieur)
- les joueurs sont l'ensemble des postes du réseau qui sont connectés au routeur.
- Les numéros des joueurs et du meneur sont appelés « adresse IP » et sont attribués par le routeur (mode automatique dit « DHCP »).



Le matériel de base nécessaire

- Pour constituer un réseau local filaire (le plus simple au début), on a donc besoin :
 - d'un **routeur** (ethernet ou wifi ou mixte) ou d'une box (qui est un routeur + modem)
 - de **plusieurs câbles RJ-45** pour connecter les éléments entre eux
 - d'un ou plusieurs **postes** à connecter sur le réseau
 - +/- d'un « multiprise » réseau, appelé switch, si le routeur n'a pas assez de connecteurs RJ-45 (Attention : un switch n'est pas un routeur... mais le routeur est un switch sur le réseau local !)



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

Atelier Arduino : Utiliser la carte d'extension (shield) Ethernet avec Arduino et créer un serveur web Arduino sur réseau local.

6. Un peu de vocabulaire pour avoir les idées claires

Avant de poursuivre, voici quelques définitions pour vous aider à avoir les idées claires :

Réseau :

Un ensemble de postes informatiques / électroniques reliés entre eux et capables de communiquer entre eux.

Réseau local :

Réseau constitué par les postes d'un même réseau (dans la maison ou le bureau). Le réseau local « pur » n'utilise pas d'accès vers l'extérieur, mais seulement une connexion de type Ethernet. Pour reprendre l'image du jeu, un réseau est la table de jeu avec le meneur et les joueurs autour. Comme nous allons le voir, chaque « table de jeu » va avoir un numéro qui permet d'identifier un réseau local donné.

Réseau internet ou web :

Réseau qui relie par le réseau téléphonique, et/ou fibre optique, des postes individuels et des réseaux locaux entre eux.

Ethernet :

Réseau où les postes sont reliés entre eux par fil à l'aide d'un câble spécial appelé RJ45.

Wifi :

Réseau où les postes sont reliés entre eux sans fil, par ondes radios.

Modem :

Appareil permettant de transmettre des données informatiques à l'aide du réseau téléphonique. C'est la fonction « modem » qui permet de se connecter à internet. La technologie utilisée actuellement par les modems est dite « ADSL ». Ici, cette fonction modem du routeur ne sera pas indispensable, mais pourra être utile pour certains programmes.

Routeur :

Appareil permettant aux différents postes d'un réseau Ethernet local de communiquer entre eux. Dans le jeu précédent, le routeur est le meneur de jeu. Le routeur va à la fois fixer le numéro de la table (= numéro du réseau) et le numéro de chaque joueur sur le réseau.

Noter qu'un routeur peut fonctionner soit en ethernet (fil) soit wifi (sans fil) soit les 2 (cas le plus courant des box internet actuelles)

Box internet

Appareil électronique et informatique qui permet de se connecter à internet et à créer un réseau local domestique. Bien comprendre qu'une box est à la fois un modem (connexion à internet par le réseau téléphonique) et un routeur (qui permet aux ordinateurs d'un même réseau de communiquer entre eux).

Carte Réseau (Ethernet) :

Carte d'interface électronique intégrée à un ordinateur ou un dispositif et qui permet de communiquer sur un réseau filaire Ethernet (local). Le shield Ethernet est une mini carte Réseau.

Carte wifi :

Carte d'interface électronique intégrée à un ordinateur ou un dispositif et qui permet de communiquer sur un réseau wifi sans fil (local). Le shield Ethernet est une mini carte Réseau.

Adresse IP :

Le numéro qui est attribué à chaque poste sur le réseau.

DHCP :

Fonction du routeur qui permet d'attribuer automatiquement les numéros (ou adresse IP) aux postes du réseau (les joueurs de la table...). Retenir que le routeur utilise une plage d'adresses prédéfinies, propre à chaque routeur.

IP fixe :

Se dit d'un poste du réseau dont l'adresse IP est fixée manuellement au lieu d'être attribuée par le routeur.

Serveur :

Se dit d'un poste du réseau qui répond à des requêtes en provenance d'un client. Un site internet par exemple est en fait un serveur réseau.

Client :

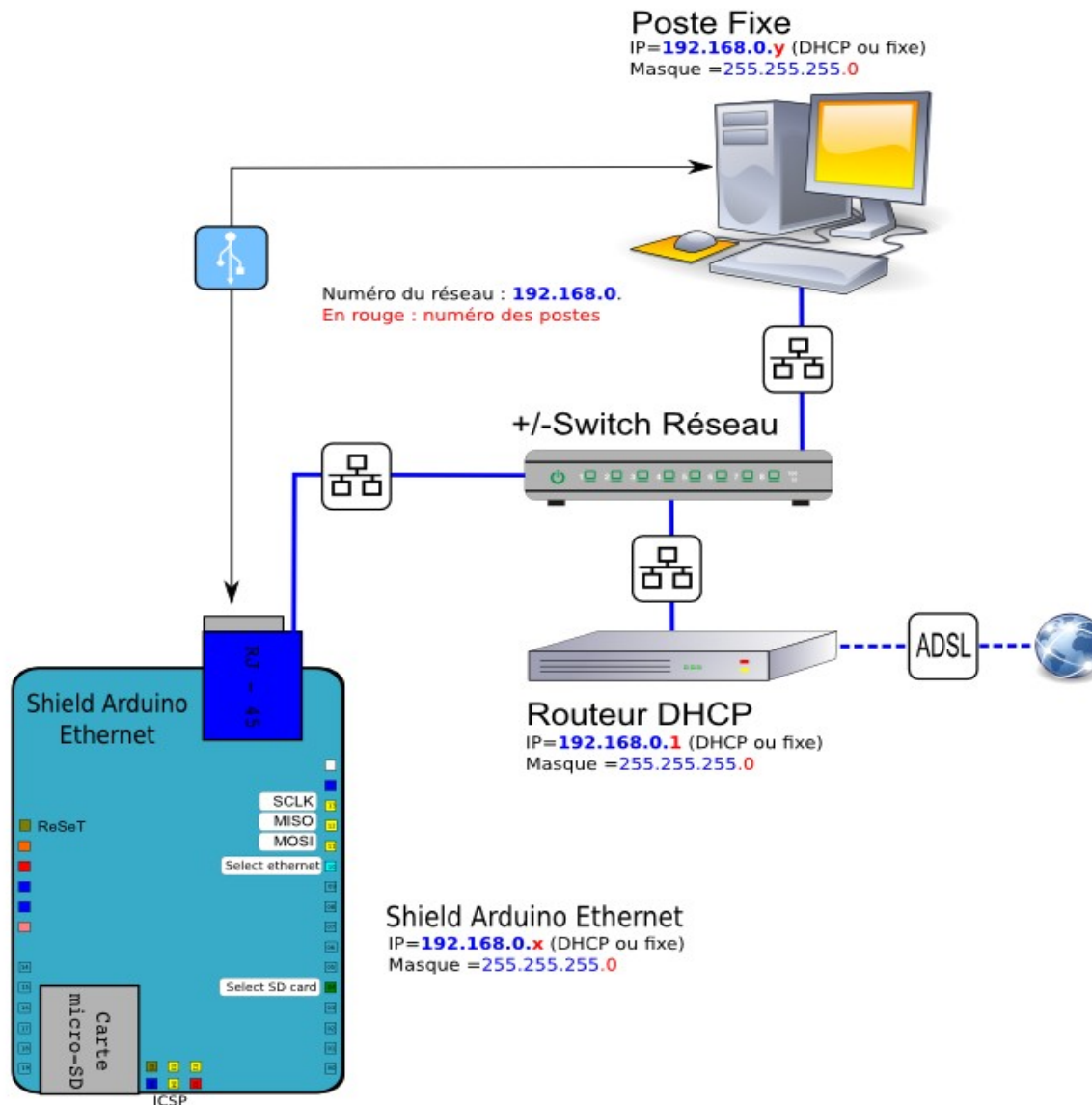
Se dit d'un poste du réseau qui envoie des requêtes (des demandes) à un serveur. Un navigateur sur un poste fixe constitue un client réseau.

Protocoles TCP/IP et UDP :

Les protocoles TCP/IP et UDP correspondent à la façon dont les postes d'un réseau communiquent entre eux : TCP/IP est le protocole de l'internet, UDP un protocole plus simple.

Si vous ne retenez pas tout pour le moment, ce n'est pas très grave. Les choses vont s'éclaircir progressivement et vous pourrez revenir sur cette page si besoin.

7. 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.

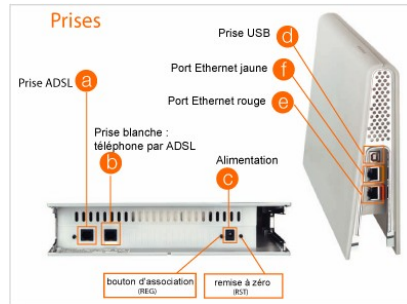
La signification des numéros (adresses IP) indiqués sur ce schéma seront expliqués par la suite.

8. Les éléments du réseau local que nous allons utiliser

Bien, à présent, trêve de « blabla », passons aux choses concrètes. Voyons à quoi va ressembler notre réseau.

Le premier élément du réseau : le routeur (ou une box).

Je me place ici dans le cas de figure courant de nos jours où vous disposez d'une box internet qui assure la double fonction de routeur et de modem. Les box intègrent la fonction DHCP d'attribution automatique des adresses IP. Votre box est le premier élément de votre réseau.



Si vous n'avez pas de box, vous devrez au moins disposer d'un routeur ethernet qui supporte la fonction DHCP, ce qui est le cas de la plupart des routeurs récents.

Noter que si vous utilisez un routeur ou une box non connecté à internet, ce n'est pas grave : vous n'aurez pas besoin de l'accès à internet pour la majorité des programmes que je vous propose.

Sur votre routeur, vous devez disposer d'au moins une prise ethernet RJ45, et idéalement 2.

+/- Élément complémentaire du routeur : le switch Ethernet

Si vous ne disposez que d'une prise Ethernet sur le routeur (c'est parfois le cas sur les box internet), vous devrez également utiliser un switch réseau qui est une sorte de multi-prises RJ45.



Le second élément de votre réseau : le poste fixe où vous travaillez disposant d'une carte réseau

Le second élément du réseau est le poste fixe où vous travaillez. Ce poste doit disposer d'une interface Ethernet filaire (ou carte réseau), L'autre possibilité est que le poste fixe se connecte au routeur par wifi, mais ce cas de figure n'est pas idéal ici car cela pourrait entraîner certains problèmes de connexion. Donc dans un premier temps au moins, utiliser de préférence une connexion Ethernet pour votre réseau.

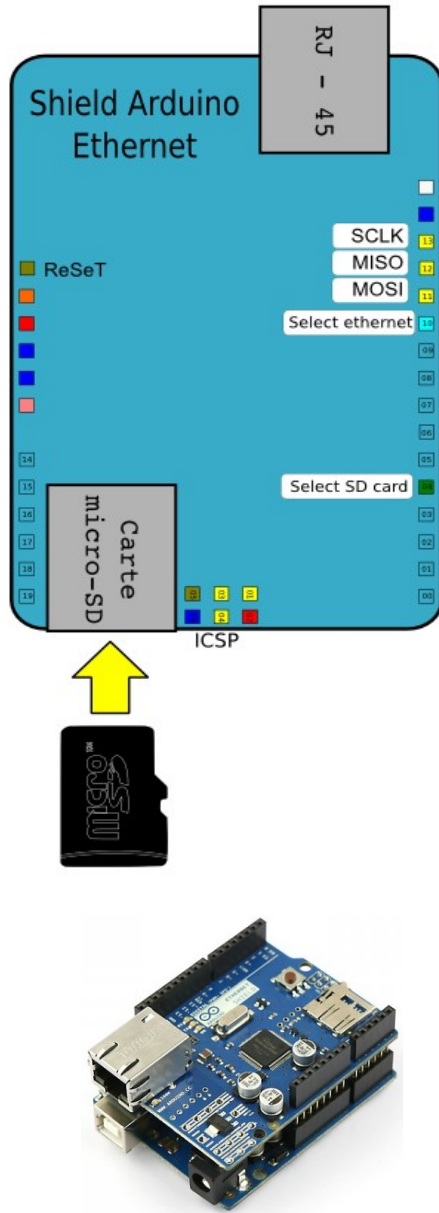


Le troisième élément de votre réseau : le couple « carte Arduino + shield Ethernet »

Le troisième élément du réseau est bien évidemment le shield Ethernet qui sera tout simplement enfiché broche à broche sur la carte Arduino



9. Le shield Ethernet : description et principe d'utilisation



Description

- Le module Ethernet Arduino permet à une carte Arduino de se connecter à un réseau Ethernet filaire ou même à internet.
- Ce module intègre également un emplacement pour une carte mémoire micro-SD, pouvant stocker plusieurs Go de données !
- Ce module est basé sur le circuit intégré [Wiznet W5100](#). Le Wiznet W5100 fournit une pile réseau (IP) capable à la fois de **TCP et UDP**. Il supporte jusqu'à quatre connexions simultanées.
- Il suffit d'utiliser la **bibliothèque Ethernet** pour écrire des programmes qui se connectent à un réseau ou à internet en utilisant ce module.

Brochage

- Ce shield communique avec la carte Arduino en utilisant une communication SPI (voir atelier dédié pour plus de détails). Cette communication utilise les 3 broches suivantes 11 (MOSI), 12 (MISO) et 13 (CLK). *Noter que la connexion de ces broches se fait par le connecteur ICSP de la carte Arduino.*
- La sélection de l'étage utilisé (carte SD ou Ethernet) se fait à l'aide de 2 broches de sélection :
 - la broche 10 pour sélectionner l'étage Ethernet
 - la broche 4 pour sélectionner la carte mémoire SD

La gestion des broches sera assurée automatiquement par les bibliothèques.

- Toutes les autres broches de la carte Arduino restent disponibles pour d'autres utilisations.

Principe d'utilisation

- Le module ethernet se connecte « broche à broche » sur une carte Arduino grâce à ses longues broches qui dépassent du circuit imprimé. On dit que le shield est « stackable » !
- Ainsi le brochage de la carte Arduino n'est pas modifié et permet d'enfiler un autre module par dessus et laisse l'accès aux broches de la carte Arduino.

10. Monter le réseau utilisant le shield Ethernet Arduino

- A faire hors tension dans la mesure du possible... Connecter le câble RJ 45 à la carte réseau du poste fixe :



- Connecter le câble RJ45 au routeur (ou à la box) :



- Mettez le shield Ethernet en place sur la carte Arduino :



- Puis connecter le shield Arduino au routeur à l'aide d'un câble RJ-45 :



- Connecter enfin le câble USB entre l'ordinateur et la carte Arduino.



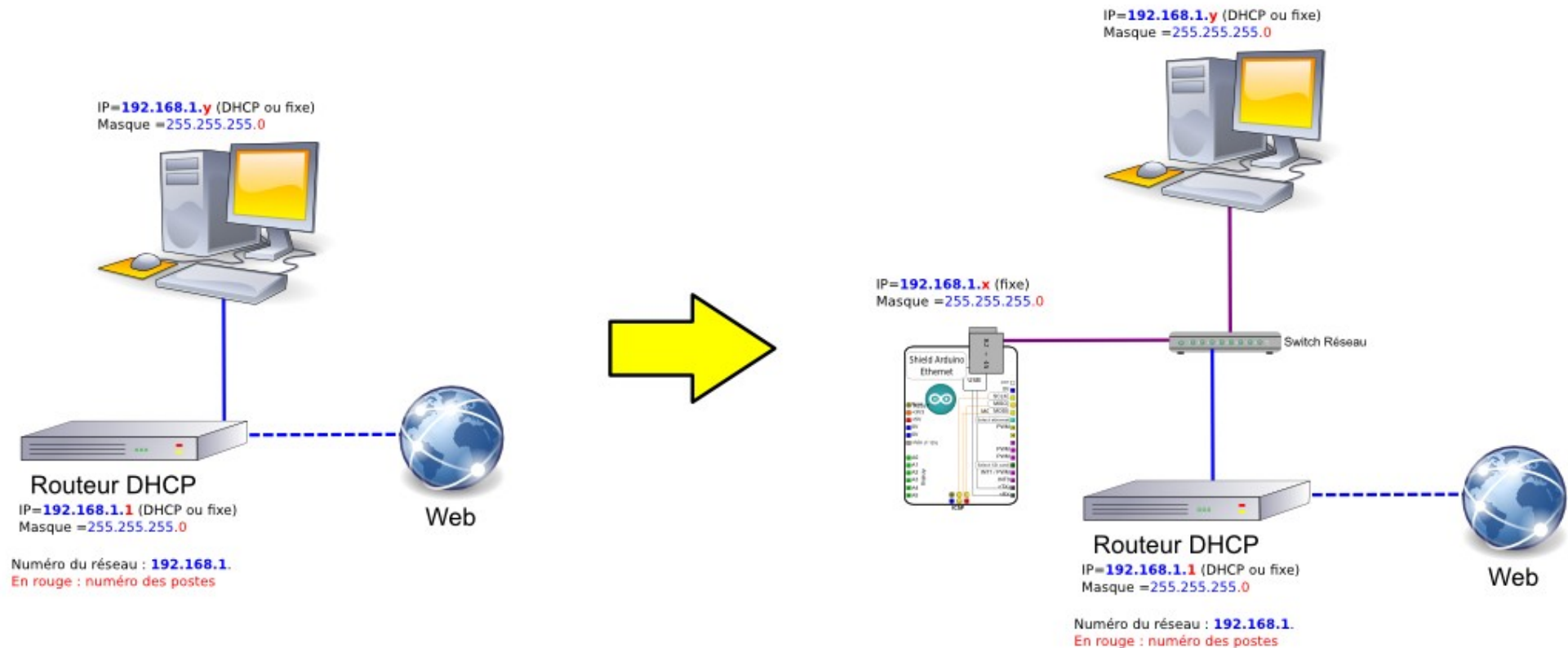
Dans notre cas, on connecte le couple Arduino + shield Ethernet à la fois en USB et en Ethernet au poste fixe, ceci uniquement à des fins didactiques et pour faciliter les développements.

En situation réelle, évidemment, le couple Arduino + shield Ethernet pourra être utilisé à distance du poste fixe, pourvu qu'il soit connecté au réseau Ethernet.

11. 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.



12. Technique : Réseau : Notion de serveur / client, notion de port

Intro

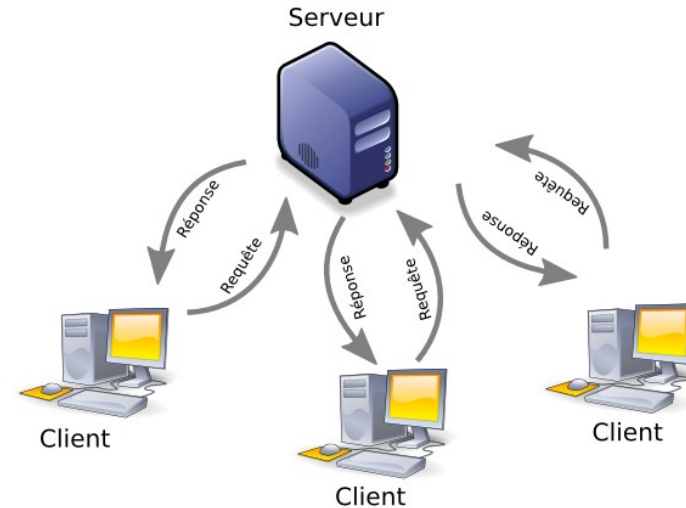
- Avant de passer à la suite, nous devons présenter deux notions à avoir en tête pour bien comprendre ce que l'on va faire.
- Si on résume ce que l'on a déjà vu au sujet des réseaux :
 - un **réseau** est un ensemble de postes qui communiquent entre eux
 - chaque poste est identifié sur le réseau par son **adresse IP**
 - l'adresse IP contient à la fois le **numéro du réseau** et le **numéro du poste** sur le réseau
 - le **routeur a le numéro 1** sur le réseau.
- Dans le programme précédent nous nous sommes contenté de connecter le shield Ethernet au réseau simplement en tant que poste réseau, sans fonction particulière.

Notion de serveur / client

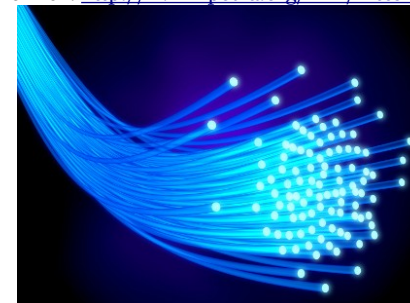
- Pour faire simple, lorsque 2 postes communiquent sur un réseau :
 - l'un des postes sera appelé le serveur
 - l'autre poste sera appelé le client
- Le **client** :
 - est **le poste qui a l'initiative de la connexion** : il se connecte au serveur
 - **envoie des requêtes**, des demandes au serveur
 - sur un réseau typiquement il existe de nombreux clients qui vont se connecter à un serveur. L'exemple type est internet où tous les postes qui se connectent à un site se connectent en fait à un serveur et sont donc tous des clients
- Le **serveur** :
 - est **le poste qui est à l'écoute sur le réseau** et attend les connexions entrantes
 - reçoit les requêtes ou demandes en provenance des clients et **envoie une réponse**,
 - un réseau comporte typiquement 1 serveur auquel se connectent plusieurs clients
- Une image pour retenir tout ça : celle du serveur qui sert les clients à la terrasse du café. Un serveur, plusieurs clients qui lèvent le doigt pour interpeller le serveur et faire leur commande. Le serveur en retour apporte au client sa commande.

Notion de port

- Un dernier point à comprendre avant de passer à la suite : la notion de port.



- L'image la plus simple pour comprendre la notion de « port réseau » est celle d'un câble téléphonique multibrins ou de fibre optique :
 - la **connexion** réseau une fois établie entre un client et un serveur d'un réseau est en quelque sorte la **gaine** autour du câble,
 - et à l'intérieur de cette « connexion », il existe en quelque sorte des « **fils virtuels** », plus de 65000 potentiellement, que les 2 postes ont à leur disposition pour communiquer entre eux.
 - **Chacun de ces « fils virtuels » est appelé un « port »** et s'apparente à un fil du câble téléphonique ou à une fibre optique.
 - **Pour communiquer entre eux, le serveur et le client devront utiliser le même port.**
 - Ces ports sont numérotés et certains sont plus connus que d'autres. Le port 80 par exemple est utilisé pour les connexions http utilisées sur internet. Plusieurs de ces ports sont réservés. Retenir que les ports 5800 à 5900 sont libres. Pour une liste des ports, voir ici : http://fr.wikipedia.org/wiki/Liste_des_ports_logiciels



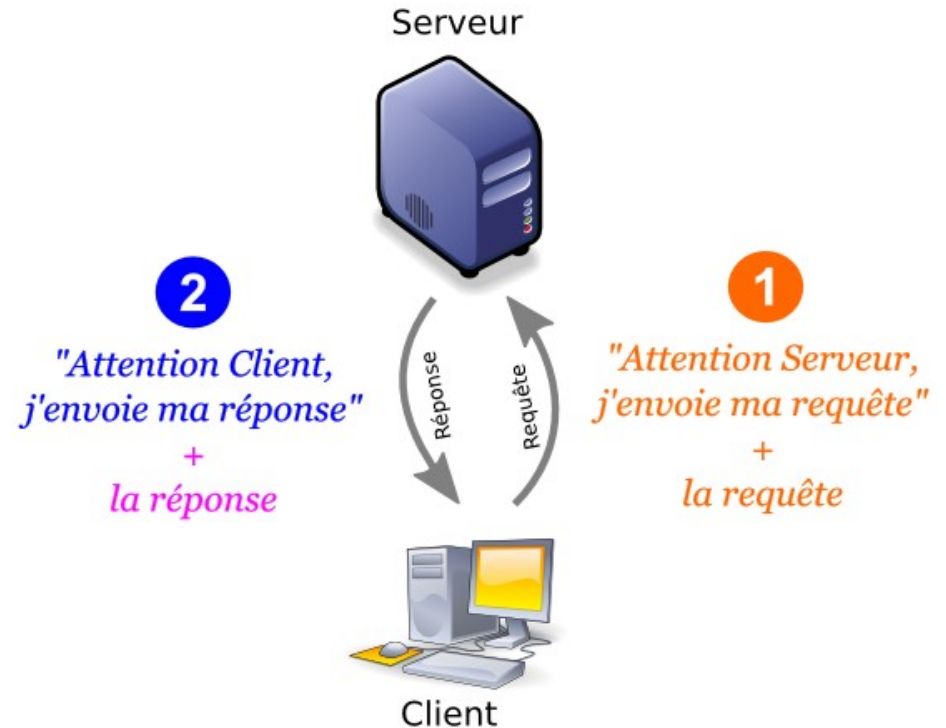
13. Technique : Réseau : Notion de protocole http: la communication entre le serveur et le client web. (1)

Présentation

- Lorsque un client et un serveur communiquent sur un réseau, dans le cas d'une connexion de type « web » ou internet, ils utilisent pour se comprendre un protocole d'échange simple. **Ce protocole d'échange s'appelle HTTP** (ça vous connaissez non ?)
- « L'HyperText Transfer Protocol, plus connu sous l'abréviation HTTP — littéralement « protocole de transfert hypertexte » — est un protocole de communication client-serveur développé pour le World Wide Web », autrement dit internet. (source : wikipédia)
- **Techniquement, ce protocole utilise une connexion de type TCP (celle qu'utilise le shield Ethernet par défaut) et le port 80 par défaut.**

Pour comprendre

- Comme on l'a dit précédemment, sur un réseau, on va trouver typiquement :
 - un serveur qui écoute les demandes et est prêt à donner des réponses
 - un ou plusieurs clients qui ont l'initiative de la connexion et vont se connecter au serveur.
- En langage courant, ça donne ça :
 - Le client commence par envoyer un message au serveur : « Serveur Attention : le message qui va suivre est une demande »
 - puis suit le message que le client envoie au serveur.
 - Le serveur sait que le message est une requête : il va donc préparer sa réponse.
 - Puis le serveur dit : « Attention client : le message qui va suivre est une réponse »
 - puis il envoie le message de réponse qui est reçu par le client.



Votre navigateur web utilise le **protocole http** pour communiquer avec les serveurs auxquels il se connecte quand vous naviguez sur internet.
Vous utilisez quotidiennement le protocole http... sans le savoir !

14. Technique : Réseau : Notion de protocole http: la communication entre le serveur et le client web. (2)

La même chose en http...

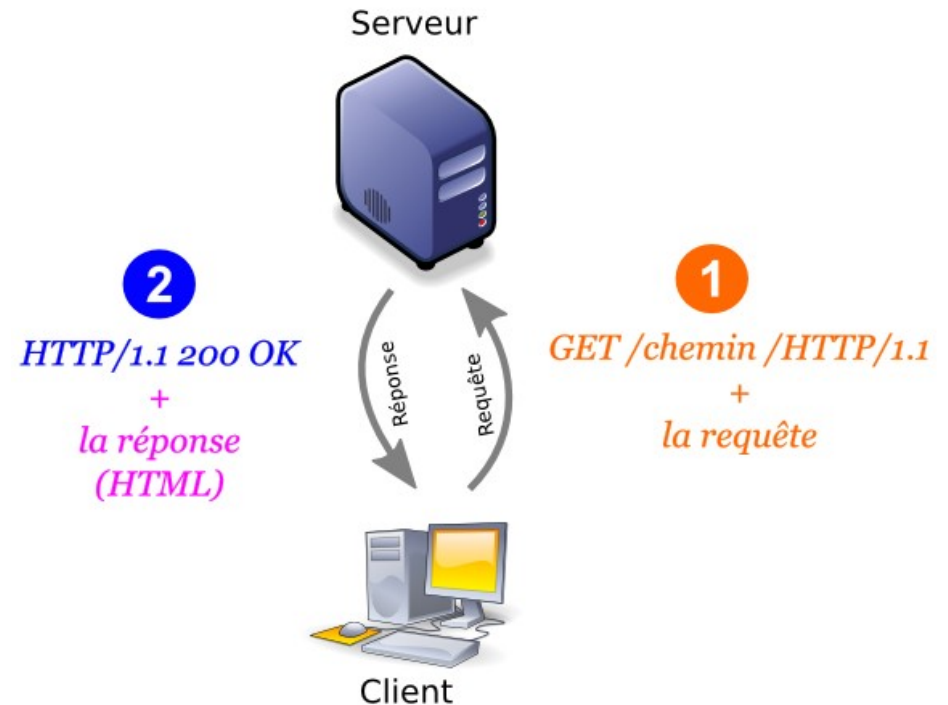
En protocole HTTP, ça donne ça :

Le client :

- commence par envoyer un message au serveur : **GET /chemin /HTTP/1.1** (=« Serveur Attention : le message qui va suivre est une demande ») Vous pouvez constater que dans le programme précédent, le navigateur (=le client distant) a bien envoyé un message commençant par GET .
- Ici, le client demande la page qui se trouve à la racine / et indique au serveur que la communication se fait en version 1.1 du protocole HTTP (il existe plusieurs versions successives...). Il existe d'autres type de requêtes, POST notamment, mais qu'il n'est pas utile de connaître à ce stade.
- puis suit le message que le client envoie au serveur : c'est qui s'est affiché dans le Terminal Série dans le programme précédent.
- Les informations envoyées sont à destination du serveur pour qu'il sache qui lui parle et quelles sont les options utilisées par le client. Ainsi, en HTTP 1.1, le champ **host** : est obligatoire. Il n'est pas nécessaire de connaître ces détails.

Le serveur :

- sait que le message est une requête : il va donc préparer sa réponse. Pour faire simple, le serveur peut aussi tout simplement attendre la fin de la requête : c'est ce que nous allons faire ici au début.
- Puis une fois qu'il est prêt le serveur dit : **HTTP/1.1 200 OK** (=« Attention client : le message qui va suivre est une réponse »).
 - Ici le serveur indique qu'il répond en protocole HTTP 1.1.
 - Puis suivent des chiffres + lettres correspondant au code de réponse. Le code **200 OK** indique que le serveur a pu répondre correctement à la requête. Vous connaissez sûrement aussi le classique **404 NOT FOUND** qui indique que le serveur n'a pas trouvé de réponse à la requête.
 - Suivent également quelques champs d'informations à destination du client, qu'il n'est pas nécessaire de connaître, sauf « Content-Type: text/html » qui indique que le serveur va envoyer une réponse texte/HTML.
- puis le serveur envoie le message de réponse qui est à son tour reçu par le client. Le message de réponse est une page dite HTML que le navigateur va être capable de décoder pour afficher les éléments dans la fenêtre du navigateur.
- **Important : bien comprendre que seul l'entête est du HTTP : la réponse sera codée différemment, en HTML.**



Pour aller plus loin...

- La présentation que je vous ai fait ici est extrêmement sommaire mais vous donne les bases indispensables pour comprendre comment les choses se passent.
- Pour aller plus loin, si vous le souhaitez, voici quelques liens utiles :
 - <http://stielec.ac-aix-marseille.fr/cours/caleca/http/index.html>
 - http://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol (les messages de requêtes client possibles)
 - www.commentcamarche.net/contents/internet/http.php3 (les codes de réponse serveur)

15. Le shield Ethernet en serveur : réaliser une communication HTTP simple : le programme

Ce qu'on va faire ici...

- Dans un précédent programme, nous avons affiché la requête du client, mais sans envoyer de réponse. Ici, nous allons compléter ce code et envoyer une réponse HTTP valide au client distant, ce qui permettra de finaliser la connexion.

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

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. Rermarker 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 programm
e ---

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

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

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

//Ethernet.begin(mac); // forme pour attribution automatique DHCP - util
ise 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("Shield Ethernet OK : L'adresse IP du shield Ethernet est :
");

Serial.println(Ethernet.localIP());

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

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

Fonction `loop()` (1)

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 : 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 ("-----");
Serial.println ("Client present !");
Serial.println ("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 ("Reception requete terminee");
```


Fonction `loop()` (2)

Affichage et analyse de la chaîne reçue

- Ensuite, tout simplement, on affiche la chaîne reçue
- Puis on teste si la chaîne commence bien l'entête GET attendue dans le cas de la réception d'une requête HTTP valide :
 - si oui, on envoie une réponse HTTP valide, affichée également en copie dans le terminal série :
 - **HTTP/1.1 200 OK** indique que le serveur a pu traiter la requête
 - Le champ **Content-Type: text/html** indique que la réponse sera du texte ou de l'html (on va voir ça après)
 - Le champ **Connection: close** indique que la connexion doit être fermée après réception de la réponse.
 - Un saut de ligne précède le message de réponse
 - Une simple ligne de texte est envoyée qui sera affichée dans le navigateur.
 - noter que la réponse HTTP est suivie d'un simple message texte qui s'affichera dans la navigateur.
 - si non, un message indique que la requête n'est pas valide.

Fermeture de la connexion avec le client

- Une fois que l'analyse est terminée, on ferme le client.

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

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

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

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

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

    //-- la reponse à afficher dans le navigateur
    client.println("Reception de la reponse du serveur http !"); // message texte
qui va s'afficher dans le navigateur client

    //-- envoi en copie de la réponse http sur le port série
    Serial.println("La reponse HTTP suivante est envoyee au client distant :");
    Serial.println("HTTP/1.1 200 OK");
    Serial.println("Content-Type: text/html");
    Serial.println("Connection: close");
    Serial.println();

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

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

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

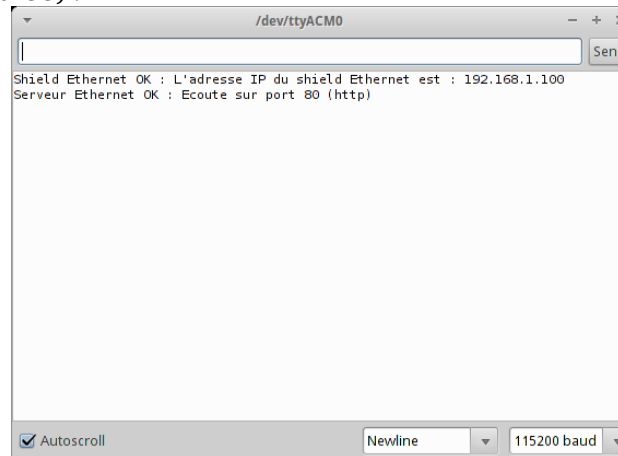
} // --- fin if client connected

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

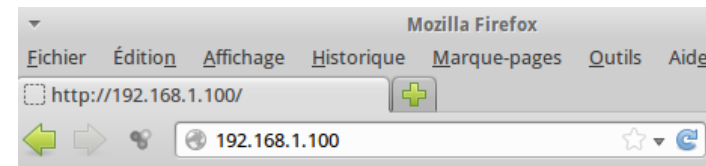
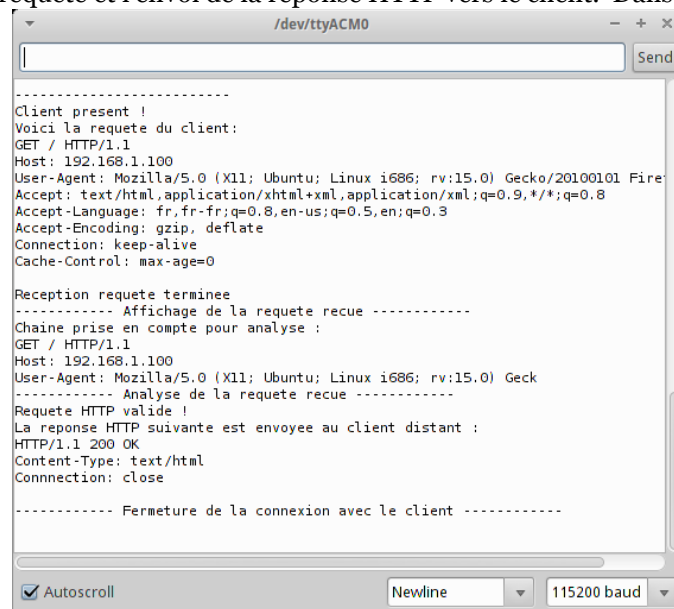
} // fin de la fonction loop()
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne un message indiquant l'adresse du serveur (ici 192.168.1.100) et le port d'écoute (ici 80) :



- A présent, ouvrir une fenêtre de navigateur Firefox sur le poste fixe connecté au réseau et saisir l'adresse du shield dans la barre d'adresse (ici 192.168.1.100). On doit alors voir apparaître dans le Terminal Série toute une série de lignes de texte correspondant à la requête envoyée par le navigateur. Puis doit suivre l'analyse de la requête et l'envoi de la réponse HTTP vers le client. Dans la fenêtre du navigateur client, doit apparaître le message de réception.



Reception de la reponse du serveur http !

A présent, notre serveur répond correctement et donc le navigateur ne reste pas en « attente » mais reçoit bien une réponse valide.

16. Technique : Réseau : Notion de base d'HTML

Introduction

- Comme on vient de le voir, pour communiquer entre eux, le serveur et le client utilisent un protocole dit HTTP. Le serveur, la plupart du temps, sur le web, envoie ensuite une réponse en HTML...
- **Un point important : bien comprendre que cette réponse « HTML » est indépendante du protocole HTTP et est en fait un fichier texte un peu particulier !**

Quezako HTML ?

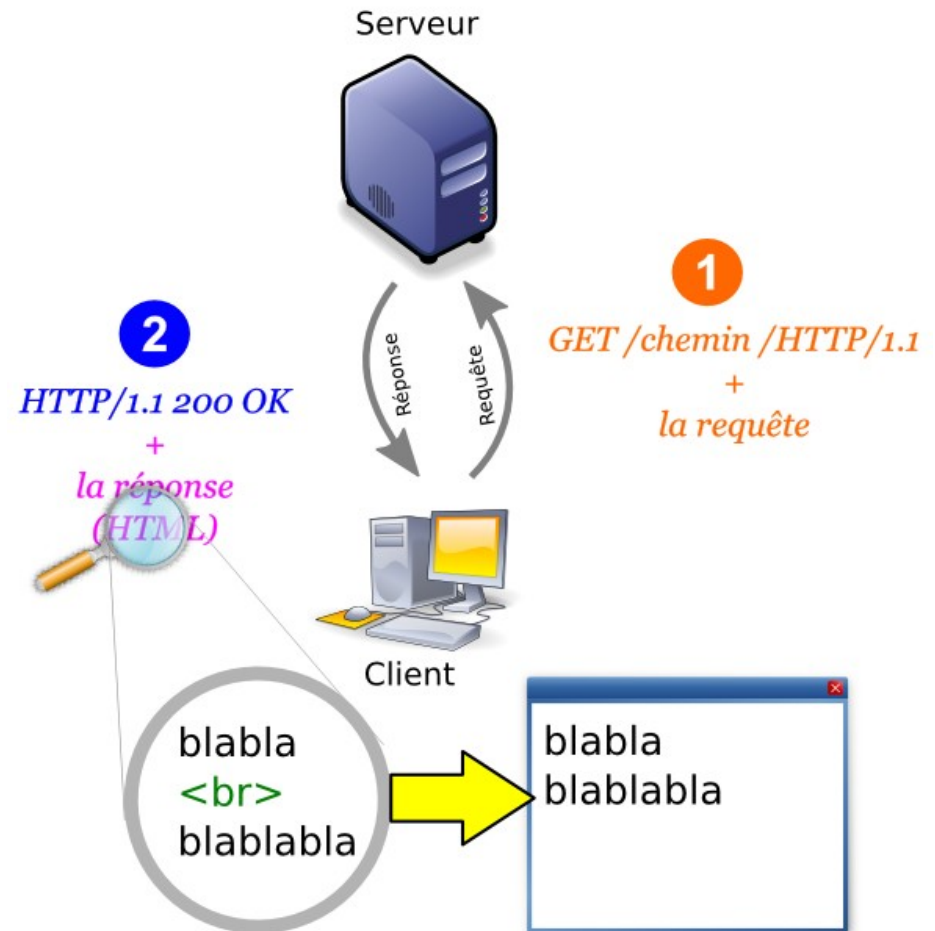
- HTML, ça doit vous dire quelque chose non ? En fait c'est un langage basé sur un système de balises qui a l'avantage de pouvoir être envoyé puis utilisé et décodé sur et vers n'importe quel type de plateforme sur un réseau : ce n'est que du texte mis en forme.
- L'HTML, c'est une sorte de langage qui est utilisé pour écrire la réponse qu'envoie le serveur et va être décodé par le client (le navigateur) qui va ainsi afficher les éléments voulus.
- Le navigateur est en fait un « décodeur » d'HTML (entre autre, car un navigateur sait faire plein d'autres choses...)

Comment ça marche ?

- Sur un réseau, l'un des problèmes que l'on rencontre, c'est qu'il peut s'y trouver toutes sortes de machines différentes : des PC sous Windows, sous Linux, des Mac, des systèmes Solaris, etc... Comment envoyer de l'information depuis un PC Windows vers un Mac et que celui-ci comprenne ? Ou inversement ?
- C'est alors qu'intervient l'HTML : l'idée consiste à utiliser du simple texte (tous les systèmes savent utiliser du texte) et à encadrer le texte qui a une signification fonctionnelle entre <> : on appelle ça une balise. C'est que va faire le serveur dans la réponse qu'il envoie.
- Par exemple, la balise
 correspond à un saut de ligne en HTML.
- Côté client, le poste qui reçoit les caractères, il affichera le texte qui n'est pas une balise mais « exécutera » le texte entre <> : ainsi lorsque le navigateur (= le client) reçoit
, il insère un saut de ligne sur la page.
- Il existe ainsi tout plein de balises : nous en présenterons quelques-unes.

Comment écrire de l'HTML ?

- Pour écrire de l'HTML, il suffit d'un simple éditeur de texte ! Mais pour que les choses soient plus simples, on pourra utiliser un éditeur à coloration syntaxique. Il y en a des tas. Sous Linux, Bluefish est sympa.



17. Technique : Réseau : Structure type d'une page HTML simple et écrire une première page HTML

Quelques balises essentielles

- De ce qu'on a dit précédemment, vous connaissez la balise **
** qui correspond au saut de ligne.
- La plupart des autres balises fonctionnent 2 par 2 avec une balise de début **<balise>** et une balise de fin **</balise>**
- la première balise à connaître est la balise **<html> </html>** : ces balises signalent le début et la fin du contenu de la page html. Tout ce qui sera compris entre ces 2 balises sera interprété comme de l'html.
- Une balise utile est celle qui délimite les commentaires : **<!-- -->**
- Toute page html comporte une obligatoirement entête contenant diverses informations et paramétrages de la page. L'**entête** est délimité par les balises **<head> </head>**
- Une balise utile au sein de l'entête est celle du titre de la page délimité par les balises **<title> </title>**
- Toute page html va également comporter obligatoirement le **corps de la page**, ce qui sera affiché dans le navigateur. Le corps est délimité par les balises **<body> </body>**.
- A l'intérieur du corps, de très nombreuses balises sont disponibles pour mettre en forme la page :
 - les balises de niveaux de titre : **<h1> </h1>** pour le titre de niveau 1, **<h2> </h2>** pour le titre de niveau 2, etc...
 - la balise d'hyperlien : **<a> **
 - la balise d'image : ****
 - etc...

Pour aller plus loin...

- Je vous présente ici seulement quelques notions de base d'HTML qui vont permettre d'écrire un serveur HTML avec Arduino. Mais il existe de très nombreuses balises et le langage HTML est un véritable « langage » de mise en forme de page avec ses subtilités, etc...
- On trouvera sur internet toutes sortes de sites permettant d'apprendre l'HTML si on le souhaite, notamment :
 - <http://www.siteduzero.com/tutoriel-3-13666-apprenez-a-creez-votre-site-web-avec-html5-et-css3.html>
 - <http://www.w3schools.com/html/default.asp> (en anglais)
 - <http://www.w3schools.com/tags/default.asp> (toutes les balises)

Vous pouvez vous contenter de ce que je vous présente ici pour passer à la suite. Nous n'utiliserons ici que des rudiments d'HTML. Mais en même temps ça vous initie en douceur à la création de pages web. Sympa non ?

Structure de la page HTML minimale

- D'après ce que l'on vient de dire, la structure de base d'une page HTML comprend :
 - les balises de limitation du début et de fin
 - les balises de l'entête
 - les balises du corps
- A cette structure minimale, s'ajoute volontiers :
 - le titre de la page
 - la définition de l'encodage de la page

Ecrire sa première page HTML

- Ouvrir un simple éditeur de texte ou mieux un éditeur HTML (je conseille bluefish sous Ubuntu, mais il y en a pour tous les goûts)
- Copier/coller le code suivant

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8" />
    <title>Titre</title>
  </head>

  <body>
    Ma première page HTML !
  </body>

</html>
```

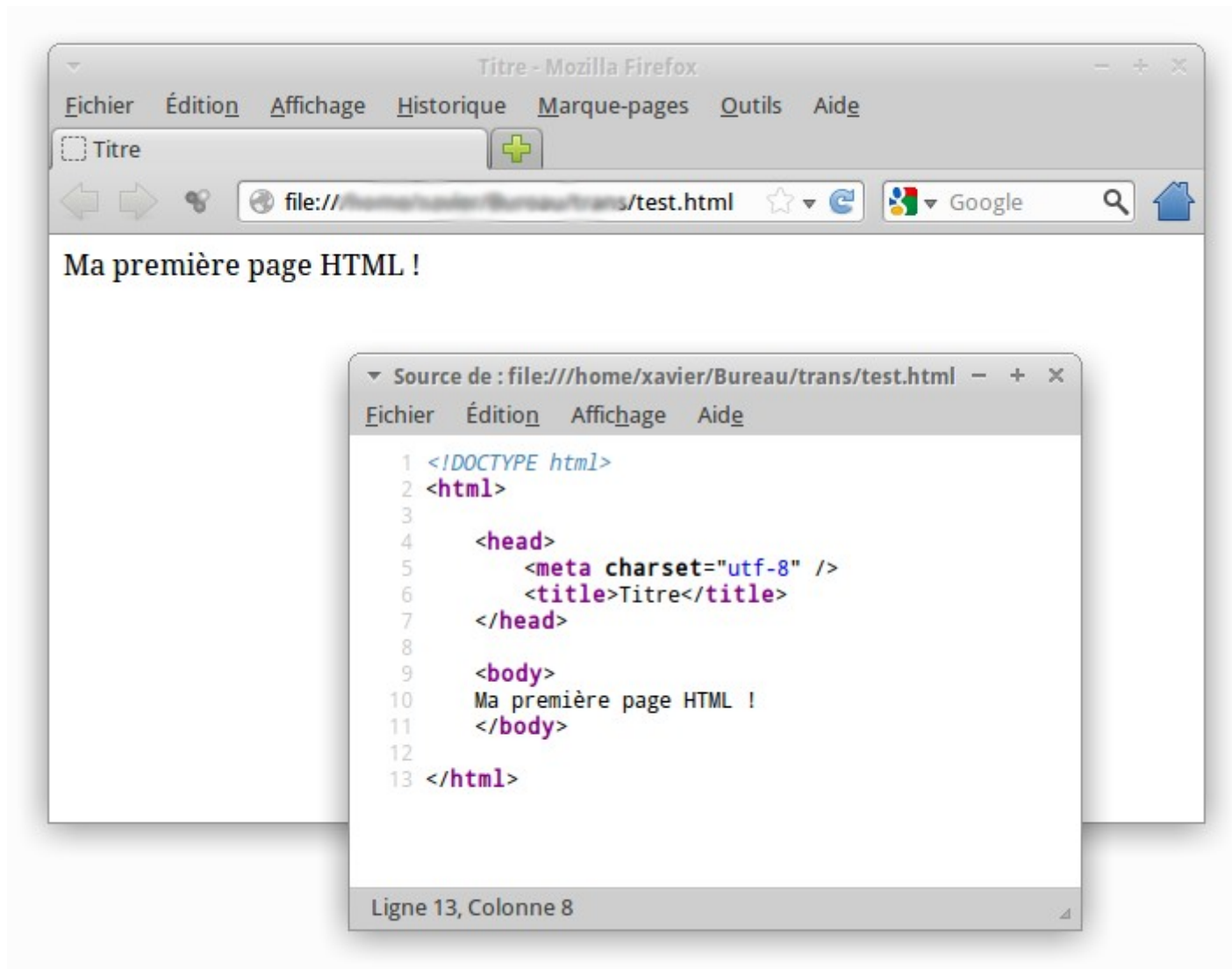
- Enregistrer le fichier au format *.html : voilà, c'est fait.

Lire une page HTML

- Pour lire une page HTML, logiquement, on utilise un navigateur, Firefox par exemple (vraiment au hasard...!)
- Puis menu Fichier > Ouvrir un fichier et sélectionner votre fichier HTML : la page doit s'afficher !

Truc : connaître le source d'une page HTML

- Quand on visualise une page HTML dans le navigateur, on voit la page, pas le code HTML, appelé aussi le source.
- Pour visualiser le source, dans Firefox, faire un clic droit dans la fenêtre de visualisation et sélectionner « Code source de la page » : vous devez voir s'afficher les secrets de la page web où vous êtes !



Votre première page HTML dans Firefox !

Faire un clic droit dans la fenêtre puis clic sur « Code source de la page » pour visualiser le code source de la page.

18. Shield Ethernet : Transformer la carte Arduino en un simple serveur HTML : le programme

Ce qu'on va faire ici...

- A ce stade, nous sommes fins prêts pour attaquer la création de notre serveur web réaliser avec Arduino. Certes, cela fonctionnera simplement sur le réseau local pour le moment, mais la chose est transposable à internet au besoin.
- Ici, le couple Arduino + Shield Ethernet va renvoyer une page HTML simple lorsqu'il recevra une requête GET en provenance d'un client distant (le navigateur du poste fixe). Il sera donc possible d'afficher des informations en provenance d'Arduino sur la page HTML ! Allez... go !

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

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. Rermarker 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 programm
e ---

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

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

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

//Ethernet.begin(mac); // forme pour attribution automatique DHCP - util
ise 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("Shield Ethernet OK : L'adresse IP du shield Ethernet est :
");

Serial.println(Ethernet.localIP());

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

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

Fonction `loop()` (1)

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 ("-----");
Serial.println ("Client present !");
Serial.println ("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 ("Reception requete terminee");
```

Fonction `loop()` (2)

Affichage et analyse de la chaîne reçue

- Ensuite, tout simplement, on affiche la chaîne reçue
- Puis on teste si la chaîne commence bien l'entête GET attendue dans le cas de la réception d'une requête HTTP valide :
 - si oui, on envoie une réponse HTTP valide, affichée également en copie dans le terminal série :
 - **HTTP/1.1 200 OK** indique que le serveur a pu traiter la requête
 - Le champ **Content-Type: text/html** indique que la réponse sera du texte ou de l'html (on va voir ça après)
 - Le champ **Connection: close** indique que la connexion doit être fermée après réception de la réponse.
 - Un saut de ligne précède le message de réponse
 - Une simple ligne de texte est envoyée qui sera affichée dans le navigateur.
 - noter que la réponse HTTP est suivie d'un simple message texte qui s'affichera dans la navigateur.
 - si non, un message indique que la requête n'est pas valide.

Fermeture de la connexion avec le client

- Une fois que l'analyse est terminée, on ferme le client.

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

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

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

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

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

    //-- la reponse à afficher dans le navigateur
    client.println("Reception de la reponse du serveur http !"); // message texte
qui va s'afficher dans le navigateur client

    --- envoi en copie de la réponse http sur le port série
    Serial.println("La reponse HTTP suivante est envoyee au client distant :");
    Serial.println("HTTP/1.1 200 OK");
    Serial.println("Content-Type: text/html");
    Serial.println("Connection: close");
    Serial.println();
```

Fonction `loop()` (3)

Envoi de la page HTML

- Ensuite, on analyse la réponse reçue : si c'est un GET valide qui a été envoyé par le client distant, on envoie la page HTML
- 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
 - on affiche du texte ainsi qu'une image

Noter le paramétrage de la couleur de fond et de la police également : pour plus de détails, voir une documentation HTML.

Fermeture de la connexion avec le client

- Une fois que l'analyse est terminée, on ferme le client.

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

// code HTML pour police jaune sur fond bleu
// <body style="color: rgb(255, 255, 0); background-color: rgb(0, 0,
255);" >
client.print("<body style="),client.print("\"); // <body style="
client.print("color: rgb(255, 255, 0); background-color: rgb(0, 0,
255);"");
//color: rgb(255, 255, 0); background-color: rgb(0, 0, 255);
client.print("\");client.print(">"); // " >
client.println("<br>");

// affiche chaines caractères simples
client.print("Coucou !");
client.println("<br>"); // saut de ligne HTML
client.print("Tu vas bien ? ");
client.println("<br>"); // saut de ligne HTML
client.print("Arduino fait le serveur pour te servir !! ");
client.println("<br>"); // saut de ligne HTML

// code HTML pour inclure une image à partir lien web
//<br> <br>
client.println("<br>");// <br>
client.print(""); // " >
client.println("<br>");// <br>

//----- fin de la page HTML -----

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

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

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

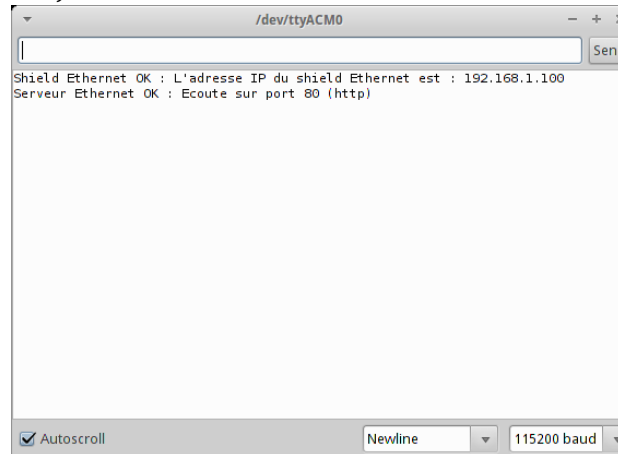
} // --- fin if client connected

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

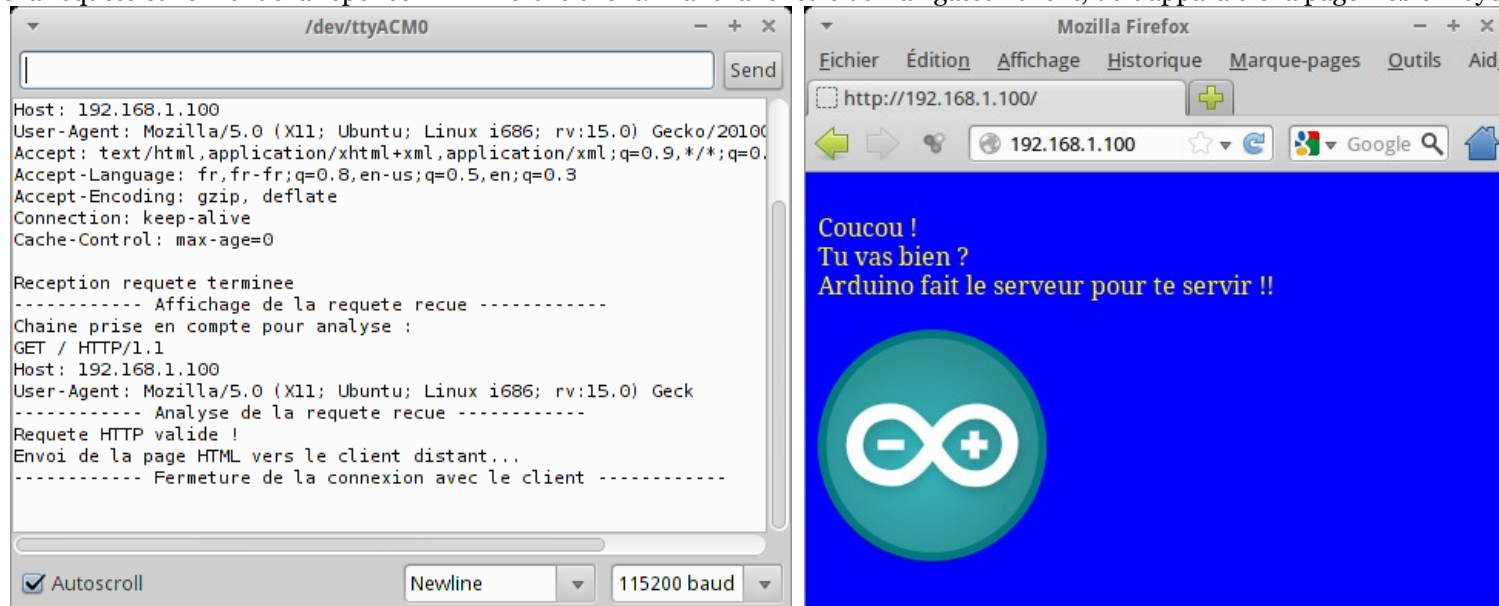
} // fin de la fonction loop()
```


Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne un message indiquant l'adresse du serveur (ici 192.168.1.100) et le port d'écoute (ici 80) :



- A présent, ouvrir une fenêtre de navigateur Firefox sur le poste fixe connecté au réseau et saisir l'adresse du shield dans la barre d'adresse (ici 192.168.1.100). On doit alors voir apparaître dans le Terminal Série toute une série de lignes de texte correspondant à la requête envoyée par le navigateur. Puis doit suivre l'analyse de la requête et l'envoi de la réponse HTTP vers le client. Dans la fenêtre du navigateur client, doit apparaître la page web envoyée par le serveur.



Bravo, vous venez de créer votre premier serveur HTML qui affiche une vraie page web (simple certes... mais quand même...) dans le navigateur !!

Grâce à un simple clic droit dans la fenêtre du navigateur (Firefox...), on accède au code source de la page qui correspond à la page qu'Arduino a envoyé :

A screenshot of a Mozilla Firefox browser window showing the source code of a web page. The window title is "Source de : http://192.168.1.100/ - Mozilla Firefox". The menu bar includes "Fichier", "Édition", "Affichage", and "Aide". The code is as follows:

```
1 <!DOCTYPE html>
2 <body style="color: rgb(255, 255, 0); background-color: rgb(0, 0, 255);"><br>
3 Coucou !<br>
4 Tu vas bien ? <br>
5 Arduino fait le serveur pour te servir !! <br>
6 <br>
7 <br>
8
```

La coloration syntaxique intégrée du navigateur montre que le code HTML reçu est valide !
A ce stade, la seule limite est votre imagination et votre connaissance du 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 :*

- de monter un réseau local utilisant Arduino
- d'écrire une page HTML basique
- d'écrire un programme Arduino pour réaliser un serveur HTTP simple
- d'écrire un programme Arduino pour réaliser un serveur Web simple
- accéder à au serveur Arduino depuis un poste fixe sur le réseau

Table des matières

Utiliser la carte d'extension (shield) Ethernet avec Arduino et créer un serveur web Arduino sur réseau local.

Intro |

Matériel nécessaire pour les ateliers Arduino |

Matériel spécifique nécessaire pour cet atelier |

Matériel spécifique nécessaire pour cet atelier (suite) |

Un peu de vocabulaire pour avoir les idées claires |

La structure du réseau que nous allons réaliser |

Les éléments du réseau local que nous allons utiliser |

Le shield Ethernet : description et principe d'utilisation |

Monter le réseau utilisant le shield Ethernet Arduino |

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

Technique : Réseau : Notion de serveur / client, notion de port |

Technique : Réseau : Notion de protocole http: la communication entre le serveur et le client web. (1) |

Technique : Réseau : Notion de protocole http: la communication entre le serveur et le client web. (2) |

Le shield Ethernet en serveur : réaliser une communication HTTP simple : le programme |

Technique : Réseau : Notion de base d'HTML |

Technique : Réseau : Structure type d'une page HTML simple et écrire une première page HTML |

Shield Ethernet : Transformer la carte Arduino en un simple serveur HTML : le programme |

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