

Créer un serveur de formulaire HTML avec Arduino et contrôler des dispositifs sur le réseau local depuis un poste client distant.



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

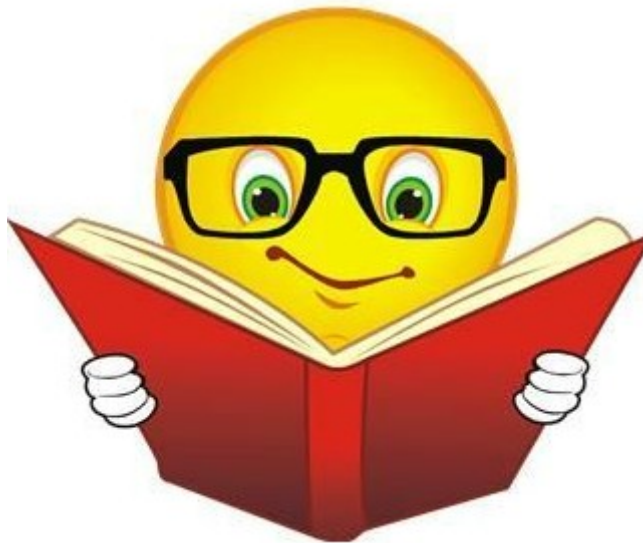
L'objectif ici est :

- d'apprendre à écrire un formulaire HTML
- d'apprendre à écrire un programme Arduino de serveur de formulaire HTML
- d'apprendre à extraire l'information utile à partir de la réponse du client à un formulaire HTML
- à contrôler un dispositif à partir de la réponse reçue d'un client

... afin d'être en mesure de contrôler des dispositifs avec Arduino sur le réseau local.

Remarque

Dans cet atelier les notions abordées vont à nouveau être nombreuses, notamment en ce qui concerne la mise en place d'un formulaire HTML. Le sujet est 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 ces notions sont nouvelles pour vous. La bonne nouvelle : à la fin de cet atelier, vous serez capable d'utiliser Arduino pour contrôler des dispositifs via un réseau Ethernet local ! Une fois que vous saurez le faire en local, vous pourrez facilement transposer en accès depuis le Web.

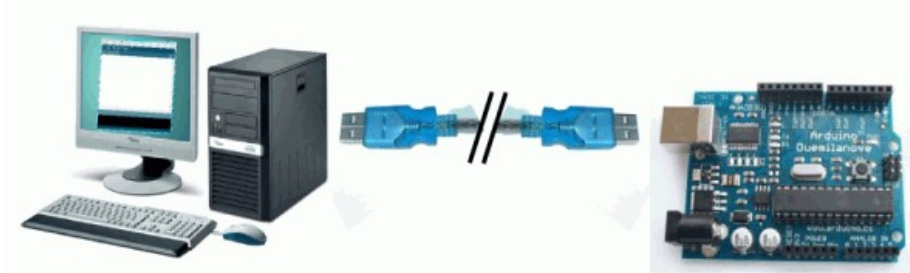


Prêt ? C'est parti !

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

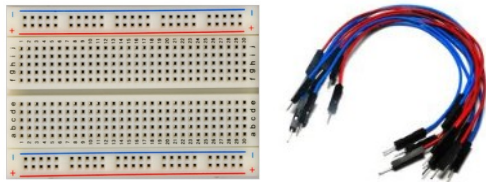


L'espace de développement Arduino associe :

- 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

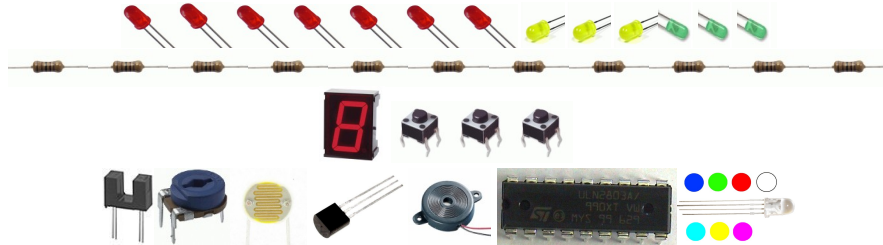


Pour réaliser des montages sans soudure, vous aurez besoin :

- 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

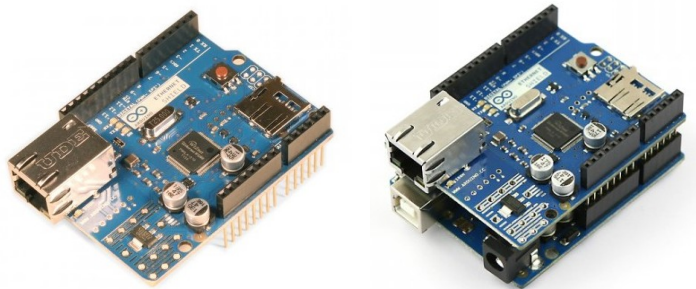
Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- 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.

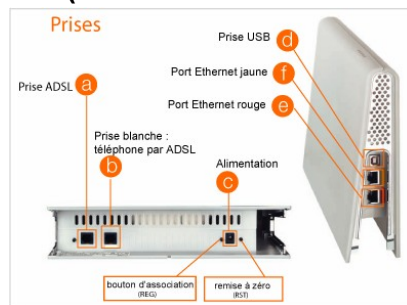
Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 +/- 4) pour communiquer avec Arduino.

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.

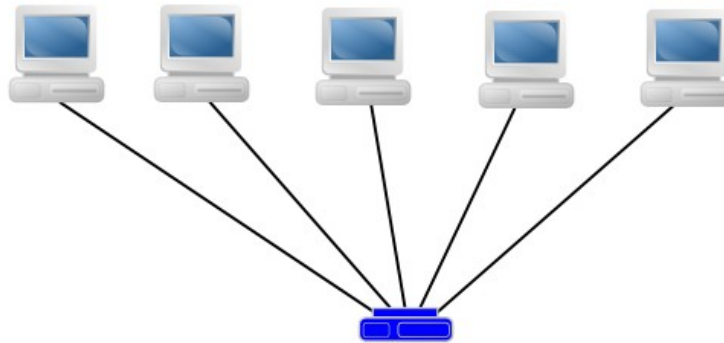
Votre poste peut-être sous Windows, Mac OS X ou Gnu/Linux, peu importe. Vous pouvez utiliser indifféremment un PC de bureau, un netbook ou un portable.

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 : Créer un serveur de formulaire HTML avec Arduino et contrôler des dispositifs sur le réseau local depuis un poste client distant.

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 :

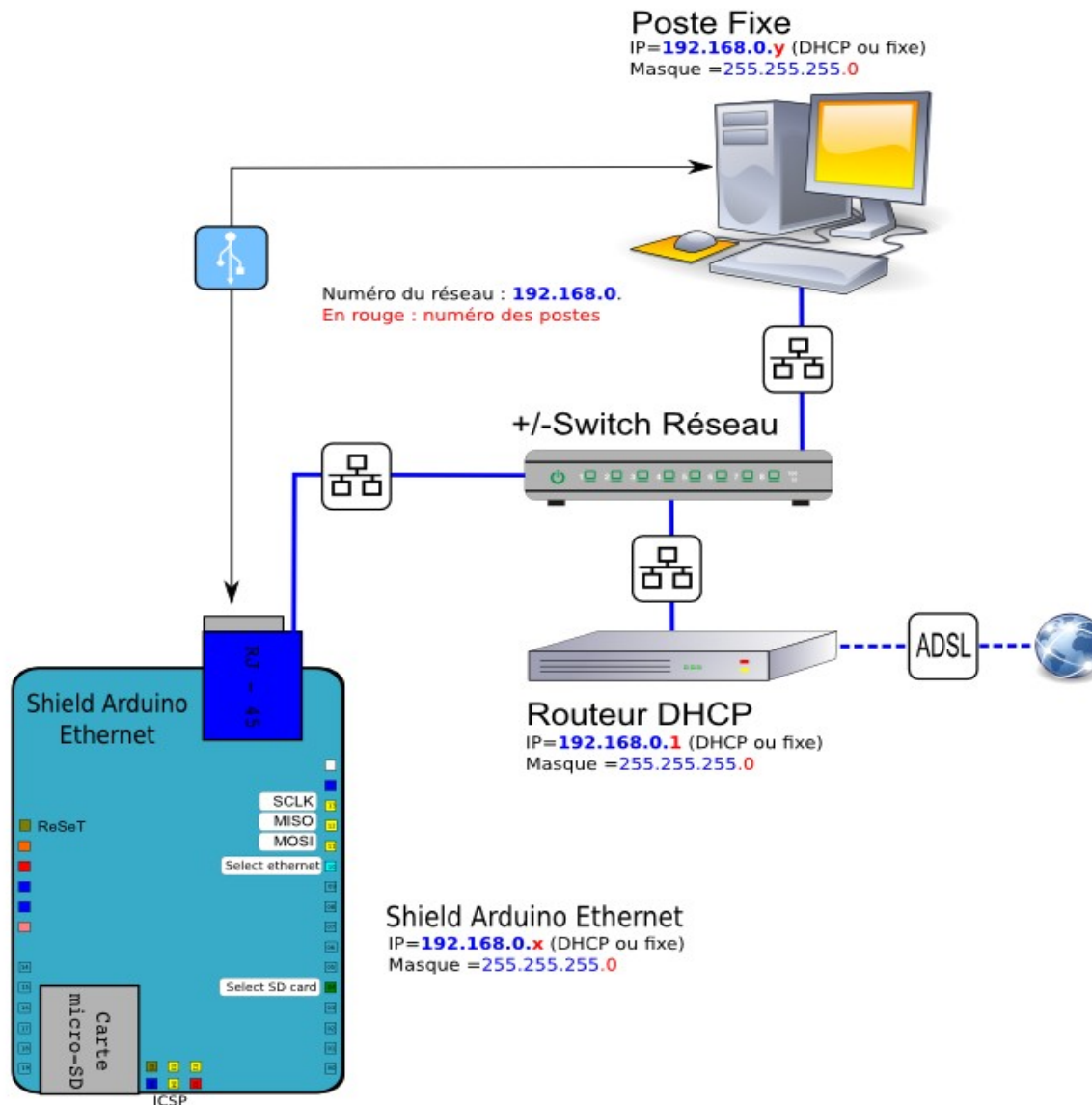
Se dit d'un poste du réseau qui envoie des requêtes (des demandes) à un serveur. Un navigateur sur un poste fixe constitue un client réseau.

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.

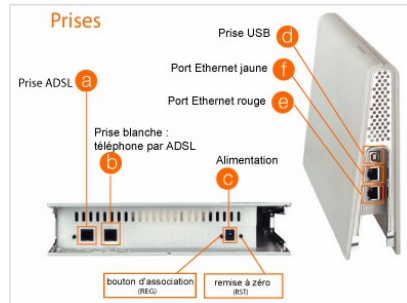
La signification des numéros (adresses IP) indiqués sur ce schéma seront expliqués par la suite.

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.

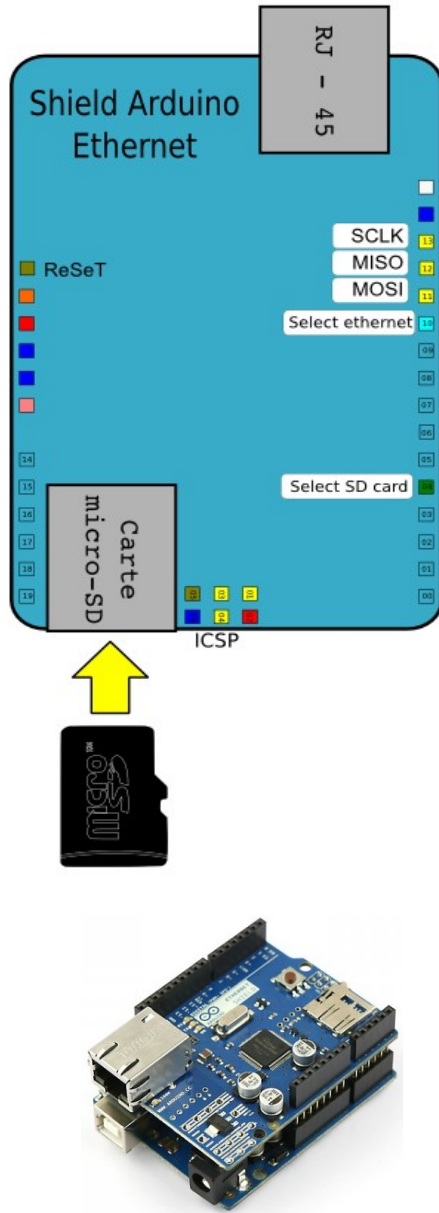


Le troisième élément de votre réseau : le couple « carte Arduino + shield Ethernet »

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.



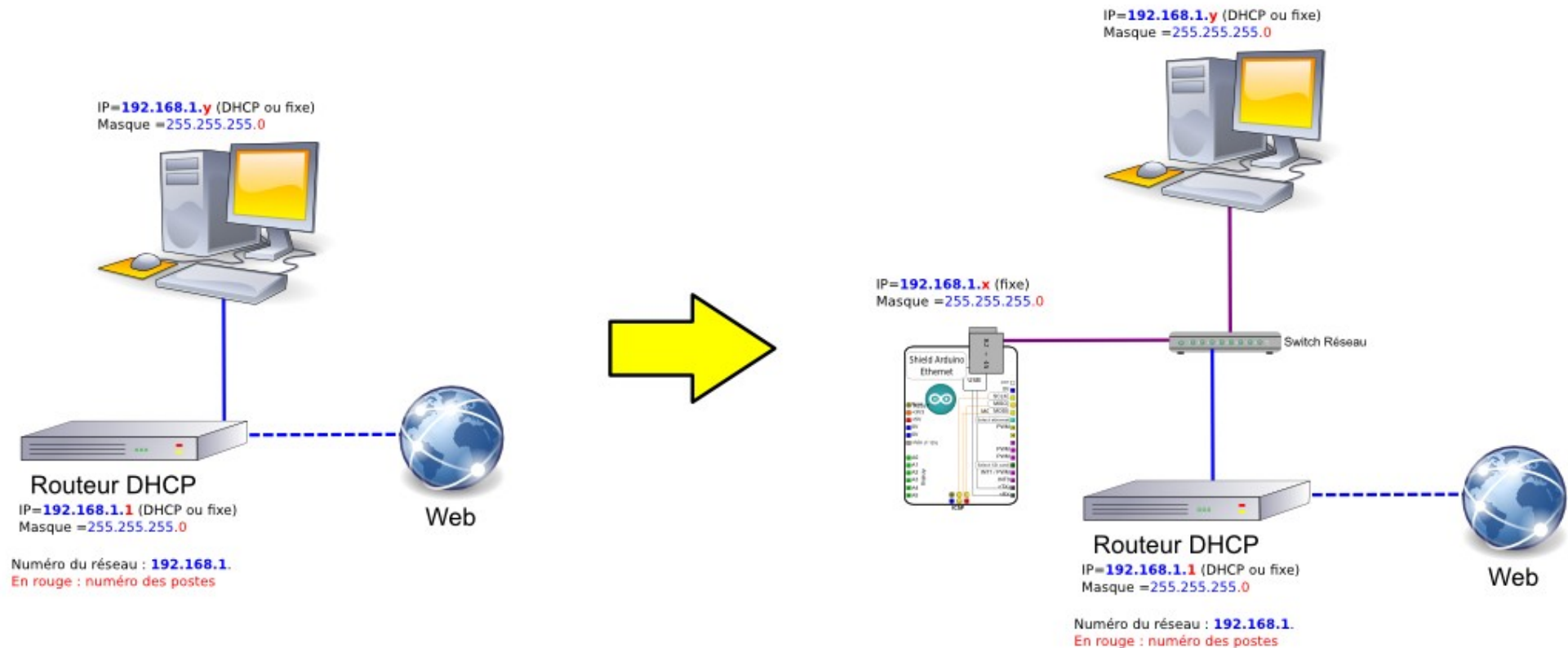
Dans notre cas, on connecte le couple Arduino + shield Ethernet à la fois en USB et en Ethernet au poste fixe, ceci uniquement à des fins didactiques et pour faciliter les développements.

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. Rappel : 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>
```

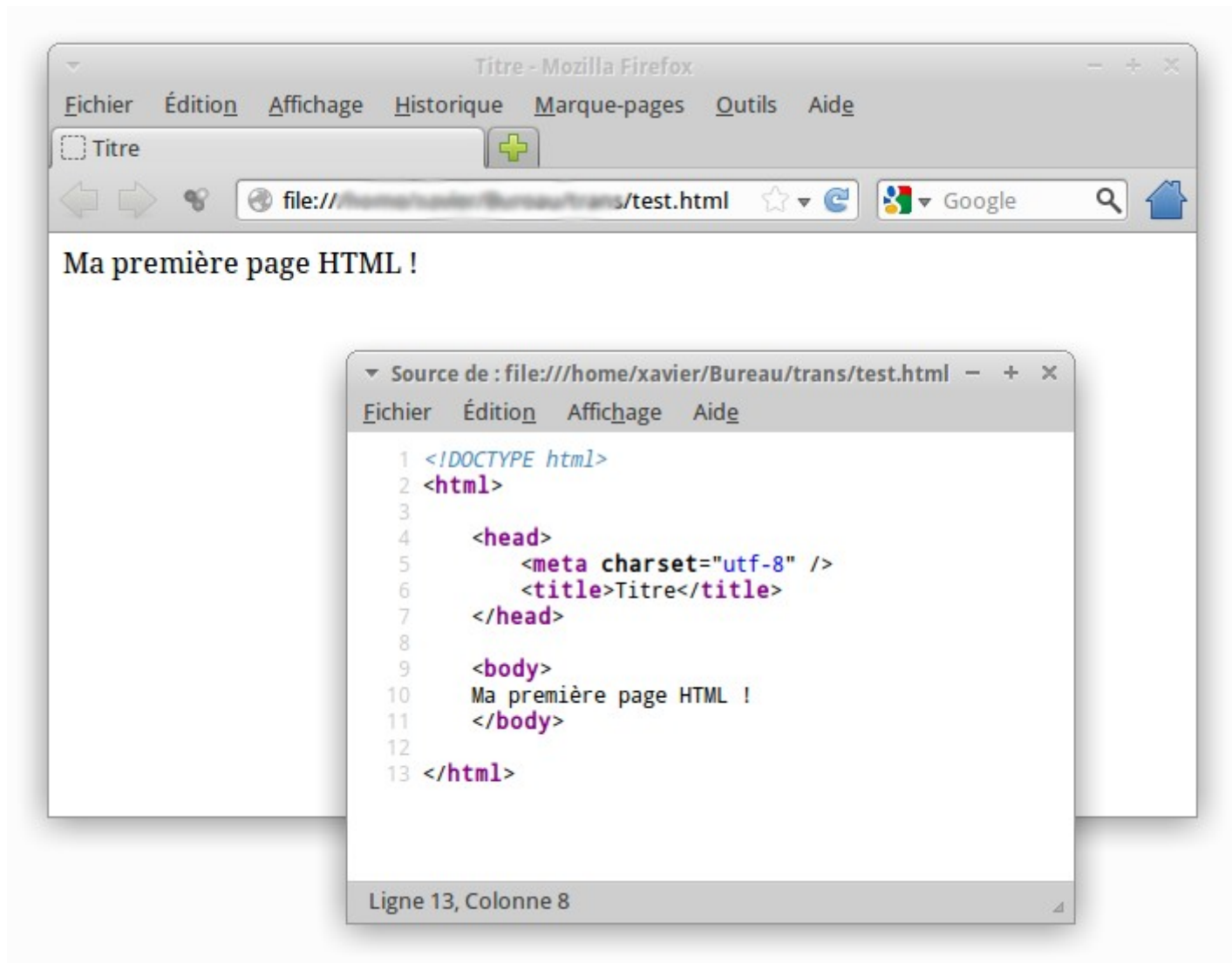
- Enregistrer le fichier au format *.html : voilà, c'est fait.

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.

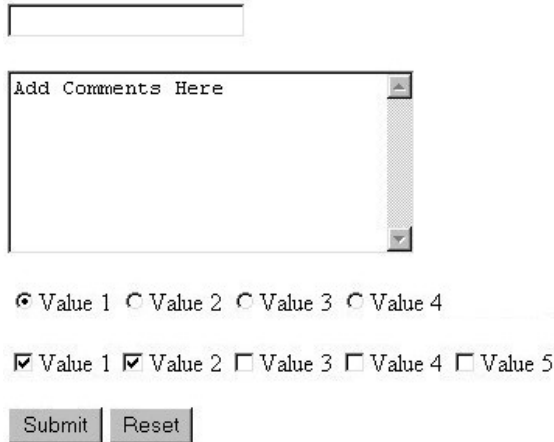
13. HTML : écrire un formulaire...

Ce qu'on va faire ?

- Je vais vous présenter ici les rudiments de la création d'un formulaire HTML afin de pouvoir ensuite créer des pages web simple permettant de contrôler Arduino.

C'est quoi un formulaire HTML ?

- Vous connaissez je pense... Un formulaire HTML, c'est un ensemble d'éléments sur une page HTML qui vont permettre d'envoyer des données.
- Typiquement, ce sont des champs texte, des boutons, des cases à cocher qui vont permettre de fixer des paramètres, des valeurs.



Exemple de contrôles de formulaire HTML :

- Champ de texte
- Champ de commentaire (Add Comments Here)
- Boutons radio : Value 1, Value 2, Value 3, Value 4
- Cases à cocher : Value 1, Value 2, Value 3, Value 4, Value 5
- Boutons : Submit, Reset

Exemple contrôles de formulaire HTML

Comment ça marche ?

- Typiquement, la page HTML indique au navigateur qu'il doit créer un formulaire et quels éléments il doit mettre sur la page : champ de saisie, case à cocher, bouton radio, etc...
- La page dispose également d'un bouton d'envoi : l'appui sur ce bouton va entraîner la lecture de l'état des éléments du formulaire et envoyer les informations à l'adresse spécifiée.
- Les informations sont envoyées sous la forme texte suivante : champ1=valeur1&champ2=valeur2&champ3=valeur3

Comment créer un formulaire HTML ?

- Un formulaire est défini à l'aide de la balise `<form paramètres >` `</form>`
- Chaque élément de formulaire est défini à l'aide de la balise `<input paramètres >` où les paramètres définissent le type de l'élément, etc...

La balise form

- Le premier paramètre de la balise form est `method` qui définit l'entête http qui sera envoyée lors de l'appui sur le bouton d'envoi. Utiliser « get ». L'autre possibilité est « post ».
- Le second paramètre de la balise form est `action` qui définit l'adresse de destination pour l'envoi du formulaire : utiliser l'adresse du serveur.

La balise input

- Le premier paramètre de la balise input est `type` qui définit le type d'élément à utiliser parmi notamment :
 - `checkbox` : case à cocher
 - `radio` : bouton radio
 - `text` : champ texte
 - `submit` : bouton d'envoi
 - `reset` : bouton d'initialisation sans envoi
- Le second paramètre est `value` correspondant à la valeur par défaut de l'élément. C'est aussi la valeur qui sera renvoyée.
- Le troisième paramètre est `name` correspondant au nom de l'élément.
- Il y a d'autres éléments et paramètres. Voir notamment : <http://www.commentcamarche.net/contents/html/htmlform.php3>

Ecrire simple formulaire 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>
    Mon premier formulaire HTML
    <form method=get action="http://192.168.1.100">
      <INPUT type="checkbox" value="ON" name="CaseACocher"> Case
      <br>
      <INPUT type="submit" value="Envoi" name="Envoi">
    </form>
  </body>
</html>
```

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

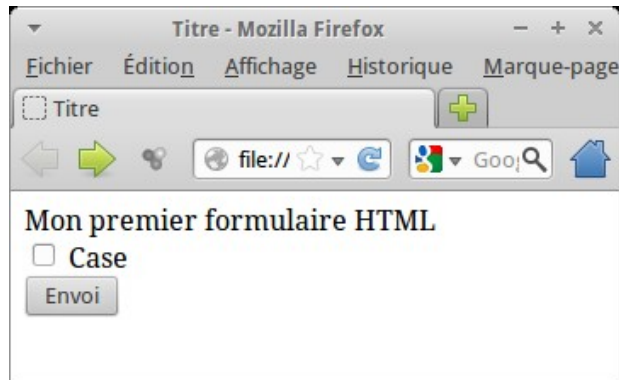
Lire la page HTML obtenue

- 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 !

Dans notre cas, ça donne :



```
Source de : file:///home/ourion/Bureau/franck/arduino/serveur/testform.html - Mozilla Firefox
Fichier  Édition  Affichage  Aide

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8" />
5      <title>Titre</title>
6    </head>
7
8    <body>
9      Mon premier formulaire HTML
10     <form method=get action="http://192.168.1.100">
11       <INPUT type="checkbox" value="ON" name="CaseACocher"> Case
12       <br>
13       <INPUT type="submit" value="Envoi" name="Envoi">
14     </form>
15   </body>
16
17 </html>
```

14. Shield Ethernet : Afficher la requête reçue depuis un formulaire dans le Terminal série : le programme

Ce qu'on va faire ici...

- Maintenant que nous sommes capables d'envoyer un formulaire vers un serveur, nous allons visualiser tout simplement la requête envoyée par le formulaire au serveur. Une fois n'est pas coutume, nous reprenons ici, en le modifiant à peine, un programme utilisé précédemment.

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.
 - Ici, on extrait également la sous chaîne utile de la requête qui est ensuite affichée dans le 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

    //-- 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();

    //-- 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
    client.println("<br>"); // saut de ligne HTML
    String requeteUtile=chaineRecue.substring(0, chaineRecue.indexOf('\n')); //
garde la sous-chaîne jusqu'au premier saut de ligne
    client.println("Requete recue par serveur = " + requeteUtile); // message
texte qui va s'afficher dans le navigateur client

} // 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 et la requête.

```

/dev/ttyACM0
Shield Ethernet OK : L'adresse IP du shield Ethernet est : 192.168.1.100
Serveur Ethernet OK : Ecoute sur port 80 (http)

/dev/ttyACM0
----- Affichage de la requete recue -----
Chaine prise en compte pour analyse :
GET /?CaseACocher=ON&Envoi=Envoi HTTP/1.1
Host: 192.168.1.100
User-Agent: Mozilla/5.0 (X11; Ubuntu)
----- Analyse de la requete recue -----
Requete HTTP valide !
La reponse HTTP suivante est envoyee au client distant :
HTTP/1.1 200 OK
Content-Type: text/html
Connexion: close
----- Fermeture de la connexion avec le client -----

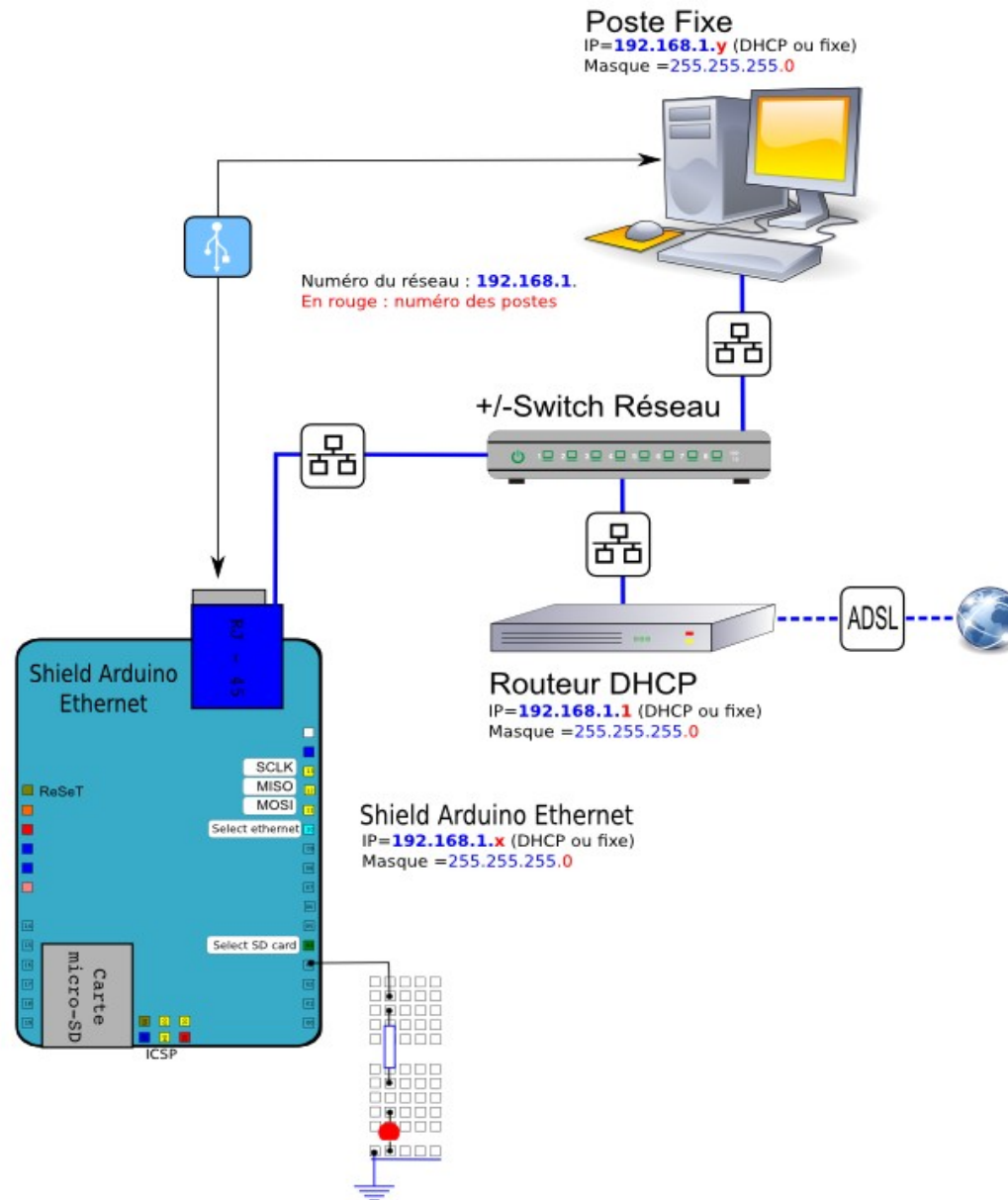
```



Remarquer le format de la requête reçue par le serveur (et donc envoyée par le client) :
si la case à cocher est cochée, on a la chaîne CaseACocher=ON suivie de Envoi=Envoi
ce qui correspond respectivement à l'état de la case à cocher et du bouton de soumission conformément à leur définition avec la balise <input>
Noter que l'enchaînement d'état des éléments du formulaire se fait avec le signe &
Si nous avons plusieurs case à cocher, ça donnerait :
GET /?CaseACocher1=ON&CaseACocher2=ON&CaseACocher3=ON&Envoi=Envoi HTTP/1.1
Remarquer enfin que si la case n'est pas cochée, aucune chaîne CaseACocher n'est alors envoyée.

15. Shield Ethernet : Contrôler une LED depuis un navigateur avec un formulaire : le montage

- nous allons reprendre le même réseau que précédemment, mais cette fois, une LED sera connectée sur une broche de la carte Arduino.



16. *Shield Ethernet :Contrôler une LED depuis un navigateur avec un formulaire : le programme*

Ce qu'on va faire ici...

- De ce que l'on vient de voir, on comprend qu'il suffit de tester la présence de la chaîne NomCaseACocher=ON dans la requête reçue pour savoir si la case en question est cochée. Sur ce principe, il suffit de nommer une case à cocher LED et ensuite de tester la présence de la chaîne LED=ON pour allumer une LED en conséquence. C'est ce que nous allons faire ici.
- Par ailleurs, il faudra également renvoyer vers le navigateur client le code de la page Web du formulaire en prenant en compte l'état courant de la LED.

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.

Déclaration broche utilisée et variable d'état

- On déclare une constante de broche pour la LED
- On déclare également une variable pour mémoriser l'état de la LED

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 constantes ---  
const int LED=3; // broche utilisée pour la LED  
int etatLED=LOW; // reflet état de la LED - étente au départ  
  
// --- 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()**

Configuration broche utilisée

- On configure la broche utilisée pour la LED en sortie

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 -----

pinMode(LED,OUTPUT); // broche en sortie
digitalWrite(LED,etatLED); // met la LED dans l'état voulu - éteinte au début

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("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 extrait sous chaîne utile de la requête
 - si la chaîne « LED=ON » est présente, on allume la LED
 - si la chaîne n'est pas présente, on éteint la LED
 - noter l'utilisation de la variable témoin de l'état de la LED pour mémoriser l'état courant de la LED

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

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

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

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

    //----- extraction et analyse de la sous-chaîne utile -----
    String requeteUtile=chaineRecue.substring(0, chaineRecue.indexOf('\n')); // garde la sous-chaîne jusqu'au premier saut de ligne
    Serial.println("Requete recue par serveur = " + requeteUtile); // message texte qui va s'afficher dans le navigateur client

    if (requeteUtile.indexOf("LED=ON")!=-1) { // si la chaîne voulue est dans la réponse utile - indexOf() renvoie -1 si sous-chaîne pas trouvée

        Serial.println("La chaîne LED=ON est valide");
        etatLED=HIGH; // mémorise état LED

    } // fin if LED=ON
    else { // sinon = si pas LED=ON

        Serial.println("Pas de chaîne LED=ON ");
        etatLED=LOW; // mémorise état LED

    } // fin else

    digitalWrite(LED,etatLED); // met la LED dans l'état voulu
}
```

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

- on envoie ensuite une réponse HTTP valide, affichée également en copie dans le terminal série :
 - **HTTP/1.1 200 OK** indique que le serveur a pu traiter la requête
 - Le champ **Content-Type: text/html** indique que la réponse sera du texte ou de l'html (on va voir ça après)
 - Le champ **Connection: close** indique que la connexion doit être fermée après réception de la réponse.
 - Un saut de ligne précède le message de réponse

```
////////// Réponse HTTP suivie de la Page HTML de réponse //////////  
  
//-- envoi de la réponse HTTP ---  
client.println("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  
  
//--- 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();  
  
//--- la réponse HTML à afficher dans le navigateur  
//----- ici on renvoie le code HTML du formulaire  
// envoie les chaines de caractères voulues = du code HTML
```


Fonction **loop()** (4) : Envoi du début de la page HTML de réponse

- Par un jeu de **println()**, on envoie la page HTML avec :
 - les balises **<html>** et **</html>** de début et fin de page
 - les balises **<head>** et **</head>** d'entête
 - **<body>** et **</body>** du corps de la page
- Noter également la balise **** pour l'insertion d'une image de LED à partir de son adresse web : l'image sera affichée si le routeur du réseau est connecté à internet : **vous pouvez de cette façon intégrer simplement des images sur la page web renvoyée par votre serveur Arduino !!**
- Remarquer également l'utilisation de la balise **<center>** qui permet de centrer le contenu de la page HTML

```
//--- la réponse HTML à afficher dans le navigateur
//----- ici on renvoie le code HTML du formulaire
// envoie les chaines de caractères voulues = du code HTML

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

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

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

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

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

// affiche chaines caractères simples
client.println("<CENTER>"); //pour centrer la suite de la page
client.println("<br>");
client.println("formulaire HTML de controle d'une LED");
client.println("<br>");

// intègre une image - suppose connexion internet disponible
client.println("<CENTER> <img src=\"http://www.arduino.cc/mes_images/communs/led_rouge_5mm.gif\"> </CENTER>");
client.println("<br>");
```

Fonction **loop()** (5) : Envoi du formulaire HTML et de la fin de la page HTML de réponse

- De la même façon, par un jeu de `println()`, on envoie le code HTML correspondant au formulaire avec :
 - la balise `<form>` :
 - Le premier paramètre de la balise form est `method` qui définit l'entête http qui sera envoyée lors de l'appui sur le bouton d'envoi. Utiliser « get » dans notre cas.
 - Le second paramètre de la balise form est `action` qui définit l'adresse de destination pour l'envoi du formulaire : utiliser l'adresse du serveur.
 - Une balise `<input>` de case à cocher :
 - Le premier paramètre de la balise input est `type` qui définit le type d'élément à utiliser ici : **checkbox** : case à cocher
 - Le second paramètre est `value` correspondant à la valeur par défaut de l'élément. C'est aussi la valeur qui sera renvoyée.
 - Le troisième paramètre est `name` correspondant au nom de l'élément.
 - A noter également le paramètre « checked » qui permet d'obtenir une case cochée par défaut. Est envoyé si la LED est HIGH**
 - Une balise `<input>` de bouton d'envoi :
 - Le premier paramètre de la balise input est `type` qui définit le type d'élément à utiliser ici : **submit** : bouton d'envoi
 - Le second paramètre est `value` correspondant à la valeur par défaut de l'élément. C'est aussi la valeur qui sera renvoyée.
 - Le troisième paramètre est `name` correspondant au nom de l'élément.
- Enfin, un simple message de texte est envoyé pour indiquer l'état de la LED. Suivent les balises de clôture de la page web

```
//----- debut du formulaire -----
client.print("<form method=get action=\"http://\"");
client.print(Ethernet.localIP()); // adresse IP du shield
client.println(">");

// case à cocher tenant compte état LED "" ou "checked"
client.print("<INPUT type=\"checkbox\" value=\"ON\" name=\"LED\"");
if (etatLED==HIGH) client.print(" checked"); // si la LED est HIGH la case à cocher est "checked"
client.println("> Allumer la LED");

client.println("<br>"); // saut de ligne HTML
client.println("<INPUT type=\"submit\" value=\"Envoi\" name=\"Envoi\">"); // bouton d'envoi

client.println("</form>");
//----- fin du formulaire -----

//---- infos sur l'état de la LED ----
client.println("<br>"); // saut de ligne HTML
if (etatLED==HIGH) client.println("La LED est ON");
else client.println("La LED est OFF");

client.println("</CENTER>"); //fin centrage de la page

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

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

} // fin if GET
```

Fonction **loop()** (6) : Fermeture de la connexion avec le client

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

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

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

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

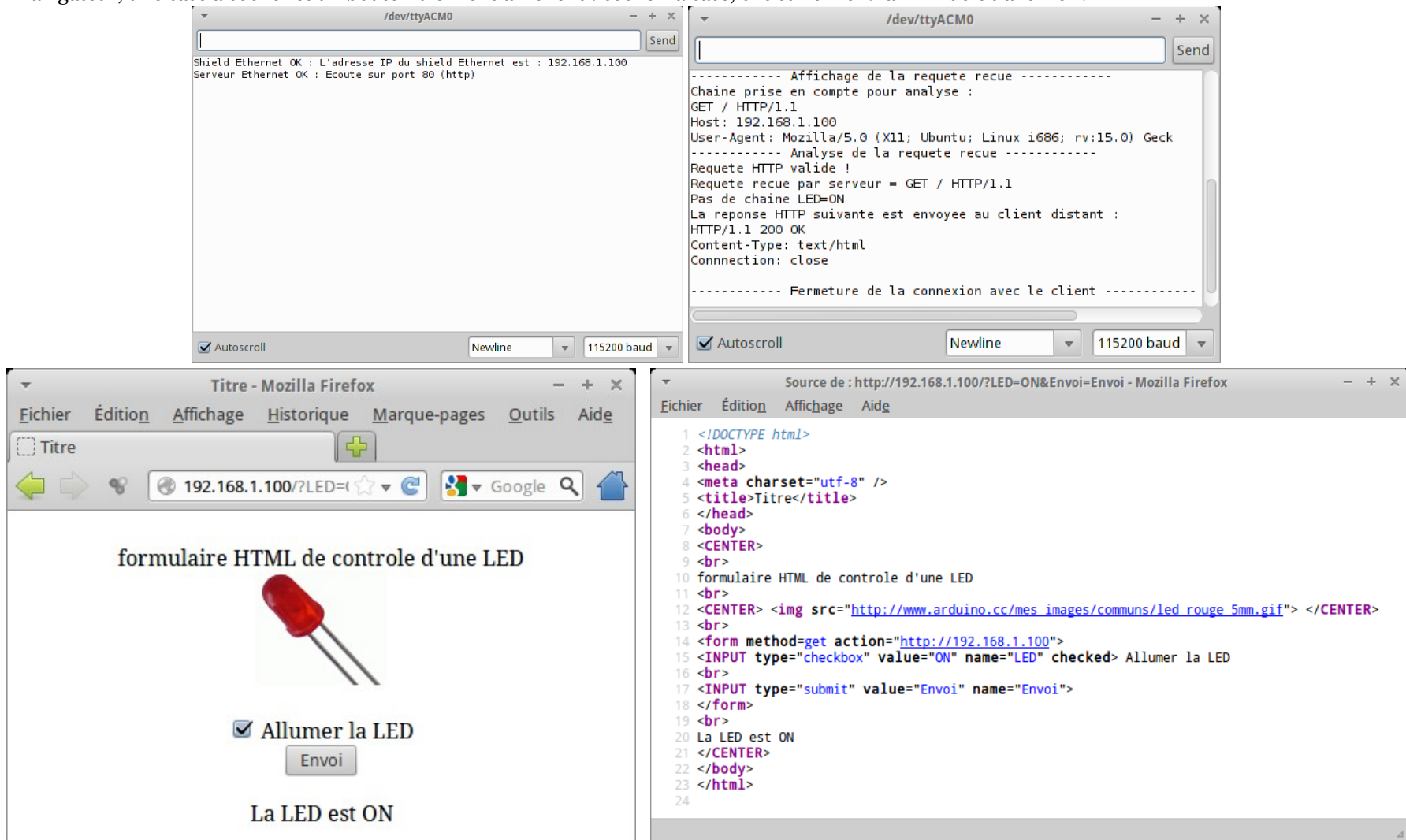
} // --- fin if client connected

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

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

Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne un message indiquant l'adresse du serveur (ici 192.168.1.100) et le port d'écoute (ici 80)
- A présent, **ouvrir une fenêtre de navigateur Firefox sur le poste fixe connecté au réseau et saisir l'adresse du shield dans la barre d'adresse** (ici 192.168.1.100). On doit alors voir apparaître dans le Terminal Série toute une série de lignes de texte correspondant à la requête envoyée par le navigateur. Dans le navigateur, une case à cocher et un bouton d'envoi s'affichent : cocher la case, clic sur envoi : la LED doit s'allumer !



Cocher la case, clic sur envoi : la LED s'allume !

L'affichage du source de la page HTML reçue par le navigateur montre le code HTML reçu depuis Arduino !

**Dans ce programme, tout se passe côté Arduino : la génération de la page HTML et du formulaire, le contrôle de la LED.
De cette façon, il suffit de se connecter à l'aide d'un navigateur à l'adresse IP du shield Ethernet sans aucune configuration particulière du
côté du client : simple et efficace !**

**Je vous conseille de garder sous le coude la structure de ce programme car elle pourra servir de base pour toute création d'un serveur
permettant de contrôler Arduino sur un réseau local ou même internet à partir d'un navigateur client.**

17. 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

18. A présent, vous devriez être capable :

- d'écrire un simple formulaire HTML
- de créer un serveur Arduino générant des formulaires HTML
- de récupérer l'information utile issue de la réponse au formulaire par le client
- de contrôler des dispositifs avec Arduino via le réseau à partir d'un formulaire HTML visualisé sur le poste client

Table des matières

Créer un serveur de formulaire HTML avec Arduino et contrôler des dispositifs sur le réseau local.

Intro |

Matériel nécessaire pour les ateliers Arduino |

Matériel spécifique nécessaire pour cet atelier |

Matériel spécifique nécessaire pour cet atelier (suite) |

Rappel : structure d'un réseau local et matériel nécessaire |

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 |

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

HTML : écrire un formulaire... |

Shield Ethernet : Afficher la requête reçue depuis un formulaire dans le Terminal série : le programme |

Shield Ethernet : Contrôler une LED depuis un navigateur avec un formulaire : le montage |

Shield Ethernet : Contrôler une LED depuis un navigateur avec un formulaire : 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