Bonjour,

Le document qui suit est le support de la formation ''Arduino applications distantes''.

Ce document propose des exemples d'applications que vous pourrez utiliser, modifier et adapter à vos différentes problématiques. Il n'est pas figé et s'enrichit à chaque formation au contact des stagiaires et des rencontres.

Merci à tous et bonne lecture

# Arduino : Applications distantes

# Problématique et présentation du support

Le but est de disposer sur internet des principaux paramètres de notre éolienne.

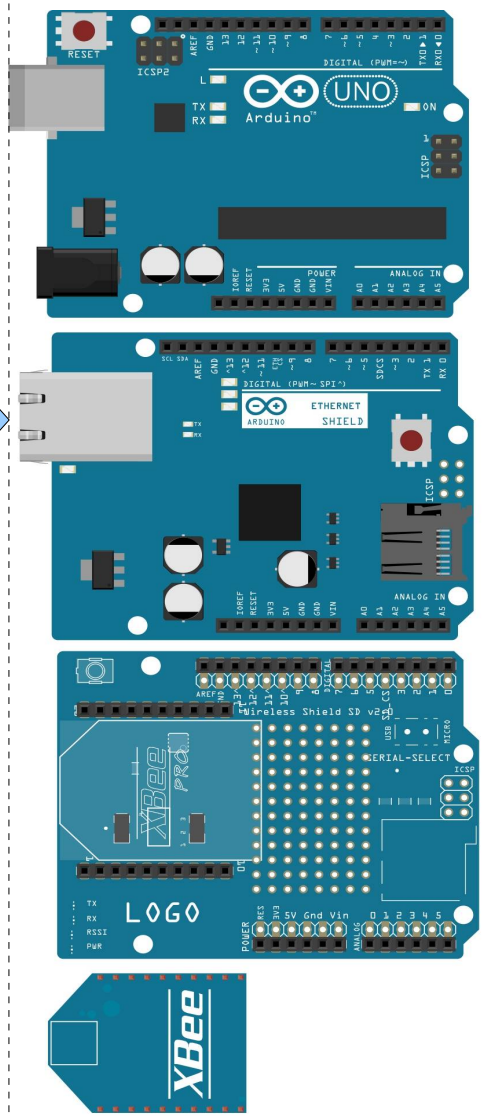
Gestion de la surveillance

Gestion de l'éolienne

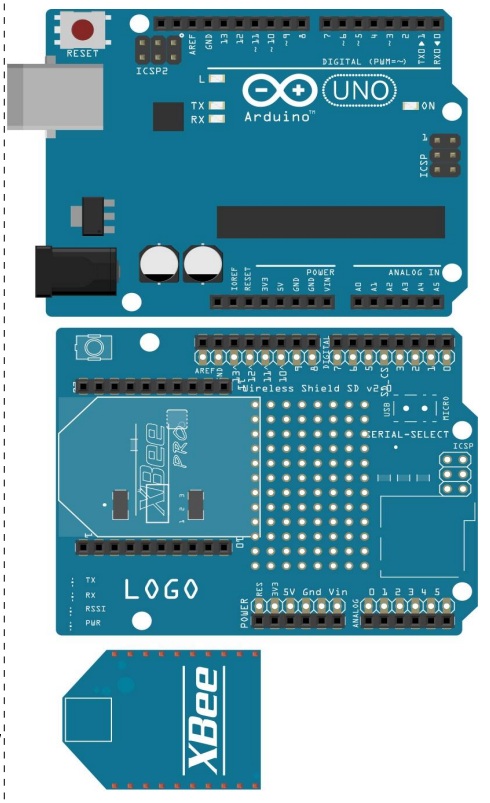
Intérieur

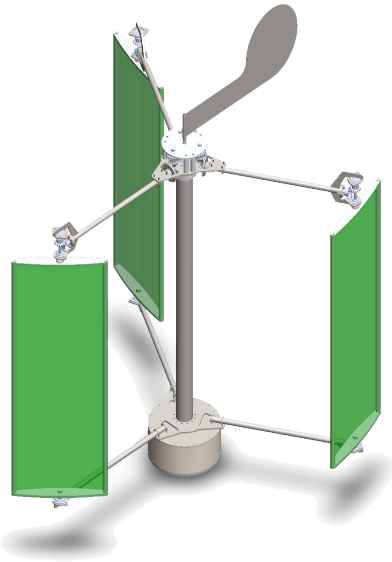
Extérieur

Internet




Transmission sans fil



Depuis peu de temps une société innovante propose une éolienne identique à notre exemple.
http://www.aeroseed.com/innovation/eolienne.php

Made with Fritzing.org

Voici une éolienne à axe vertical synchrone (réalisée par des élèves de troisième en démarche de projet)

Collège Vallis Aeria (2011-2014)

Vidéo sur :

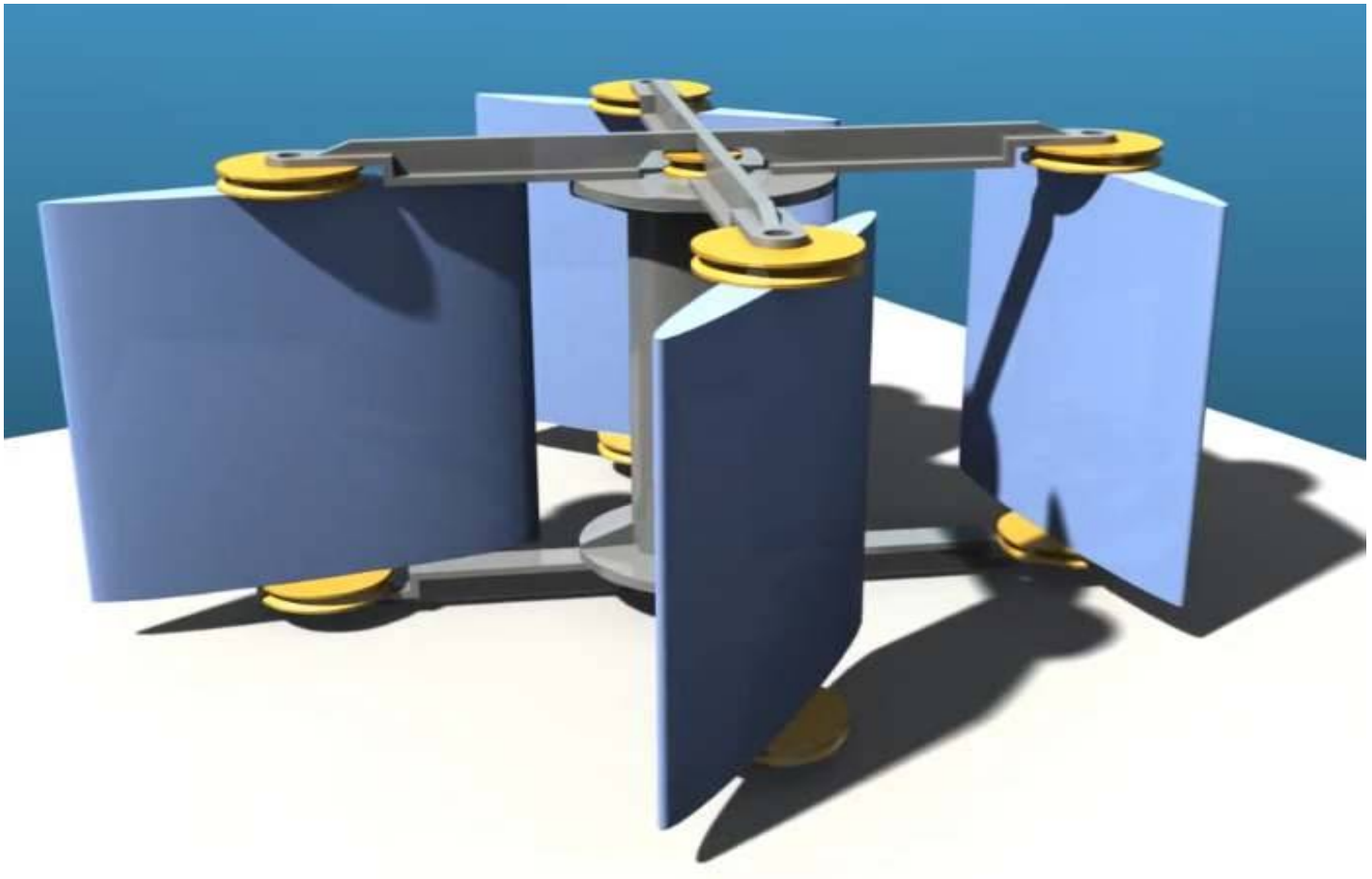
http://www.dailymotion.com/video/x27e1z7_eolienne-axe-vertical_tech

Le système doit gérer une éolienne d'un type un peu particulier : axe verticale à voilure tournante.

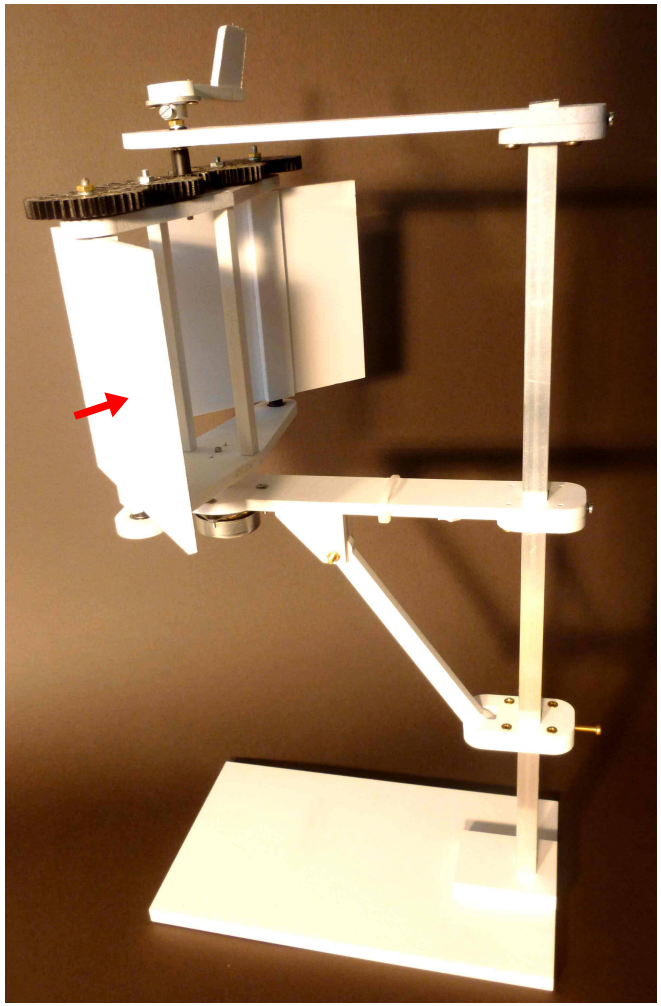
Cette éolienne pour fonctionner doit être synchronisée par rapport au sens du vent.

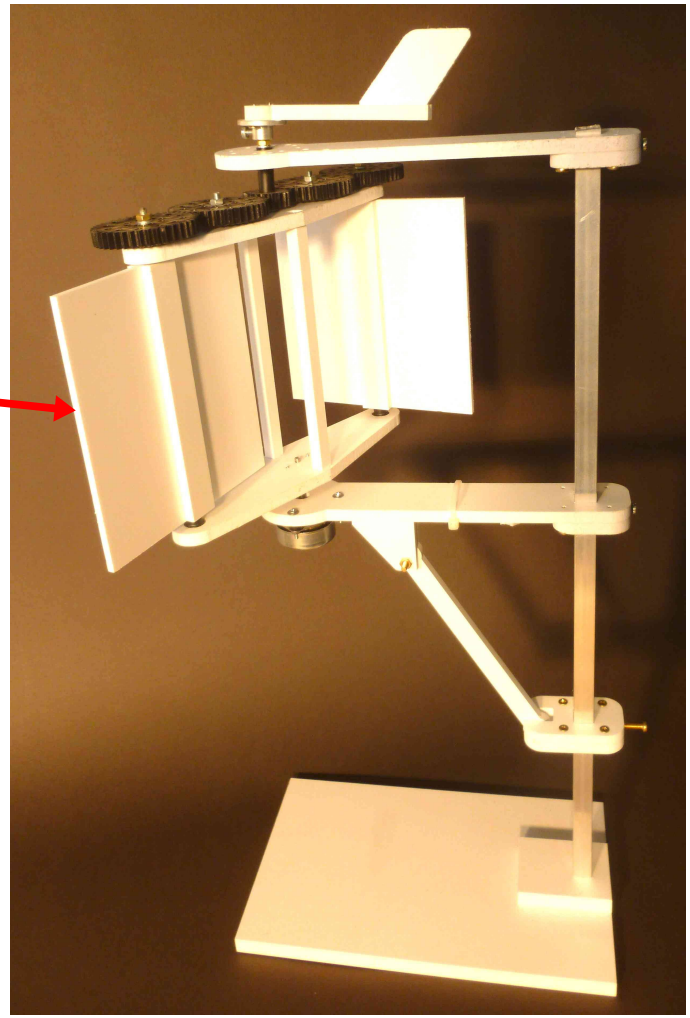
Si nous la désynchronisons nous pouvons faire varier sa vitesse  en relation avec l'effort qu'elle doit fournir et ainsi réguler sa vitesse.

(http://vimeo.com/52000418  http://www.eolprocess.com  http://www.aeroseed.com/innovation/eolienne.php)

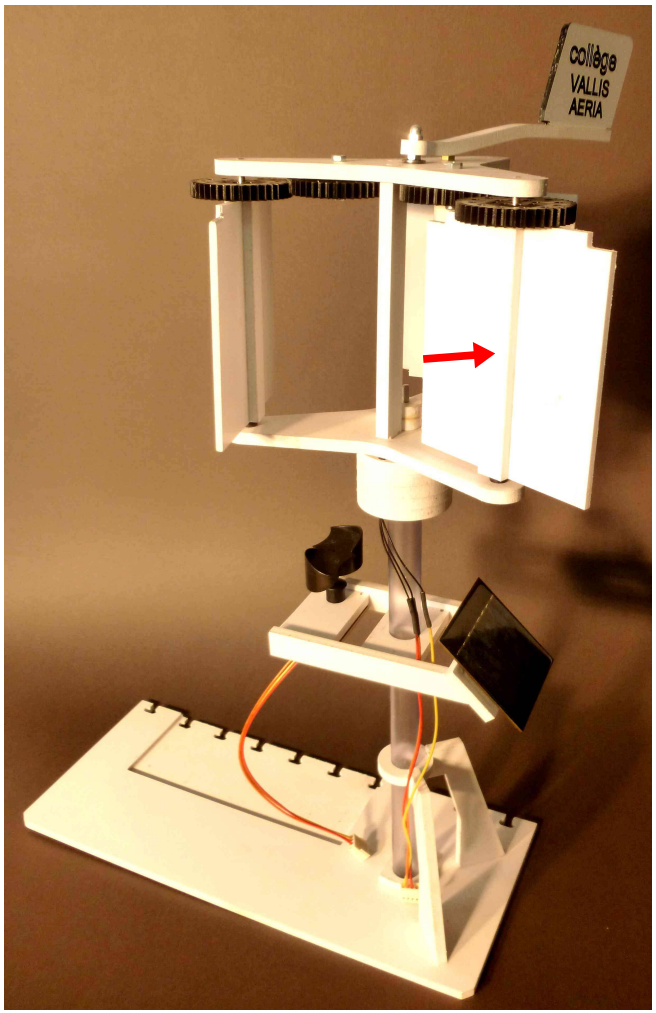


Exemples de réalisations fonctionnelles d'étude :
Ici la girouette est la référence et synchronise l'éolienne.
Cette éolienne a la particularité de récupérer l'énergie aussi sur la pale contre le vent.
Exactement comme un voilier qui remonte au près.





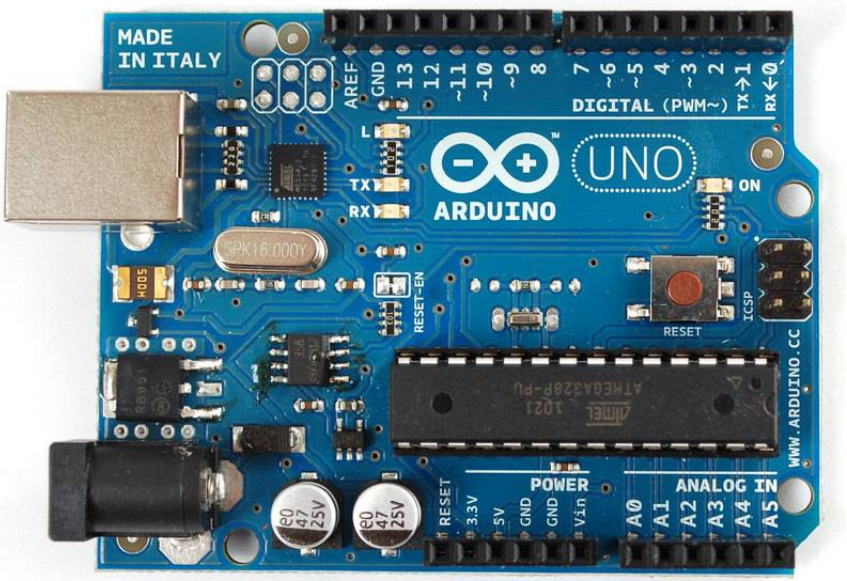
Modèle à deux pales dans différentes orientation du vent



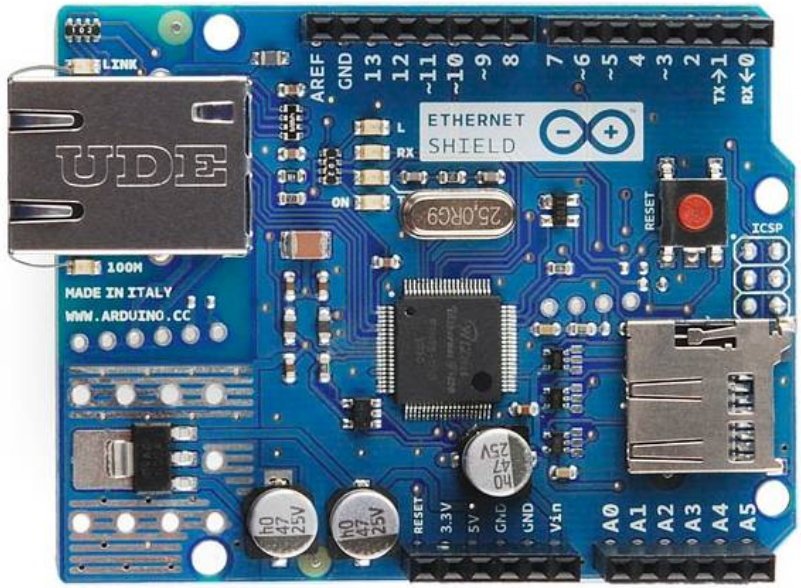

Modèle à trois pales

La synchronisation nécessite un rapport de deux entre la référence au vent et les pales.

# Présentation du matériel

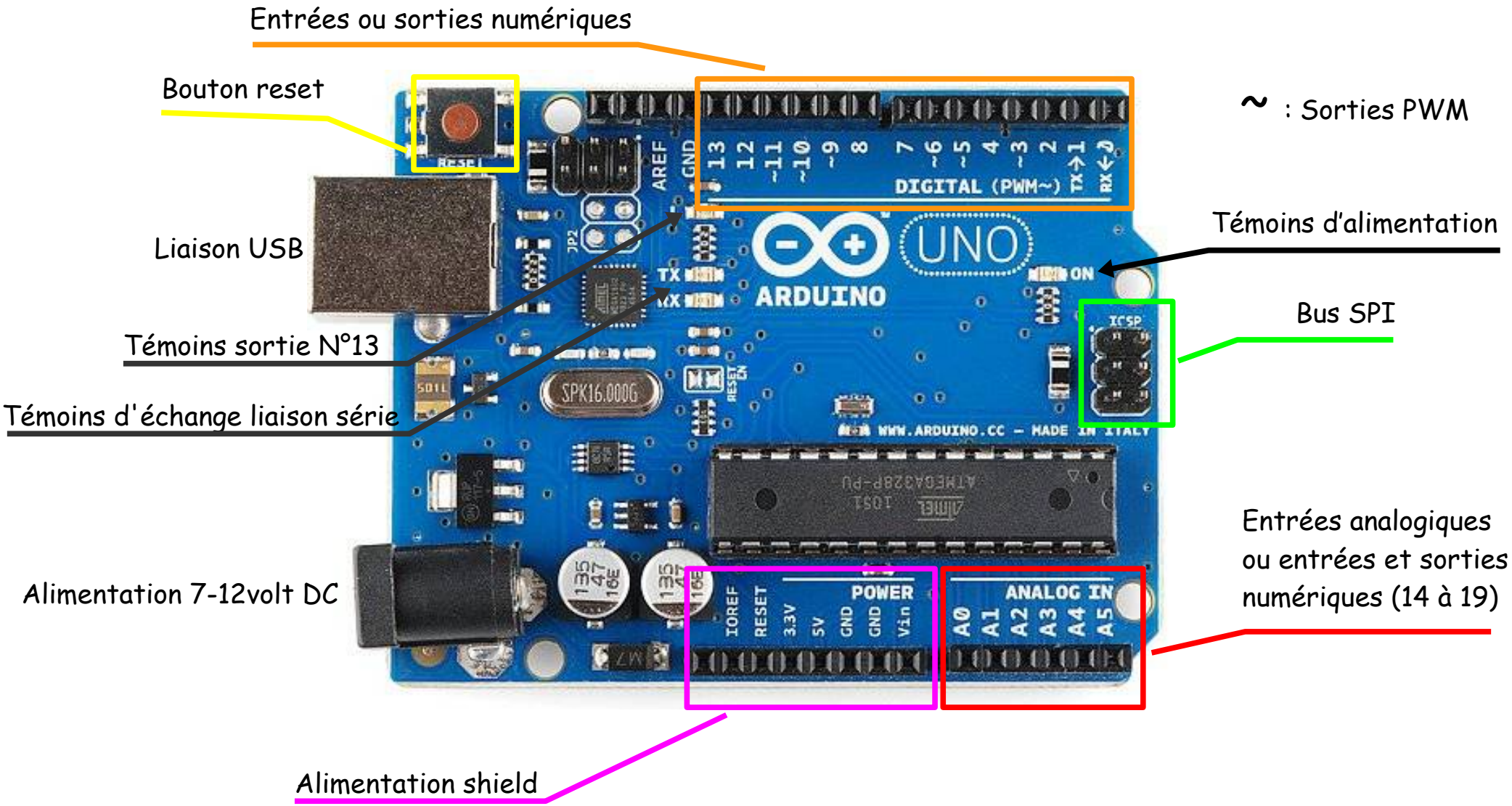



**Arduino Uno**

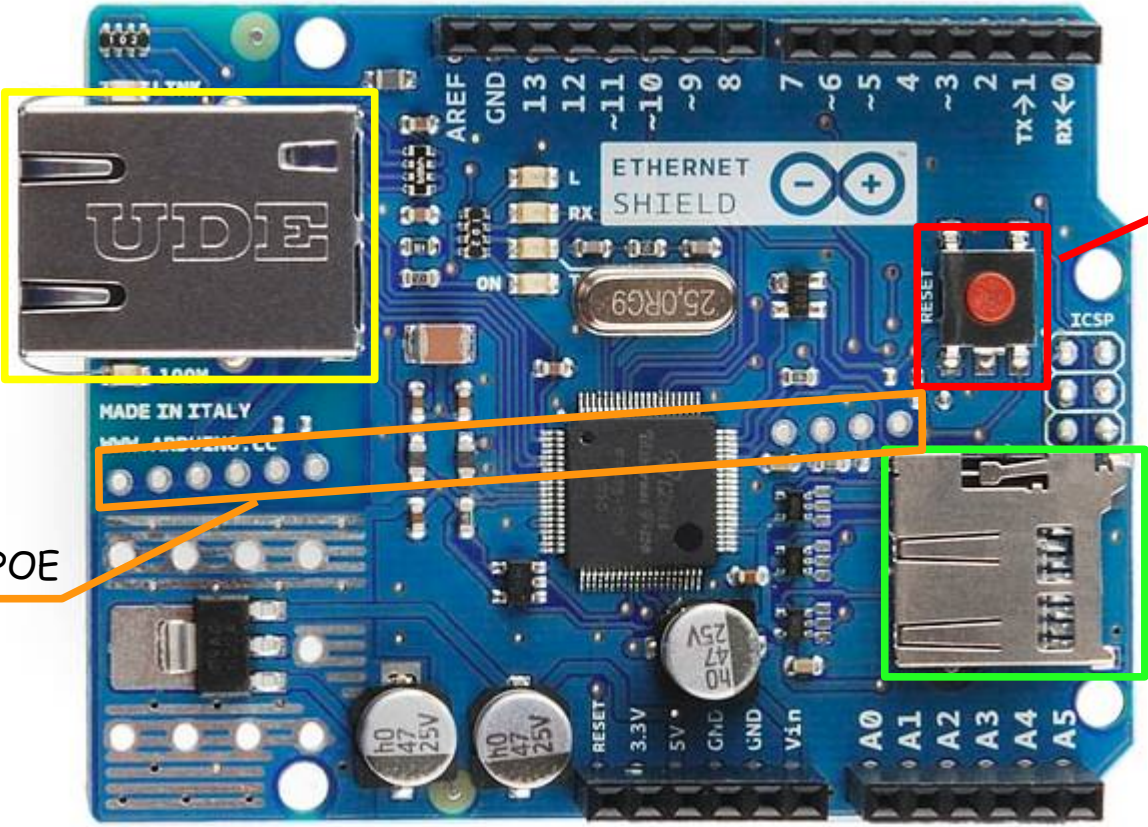


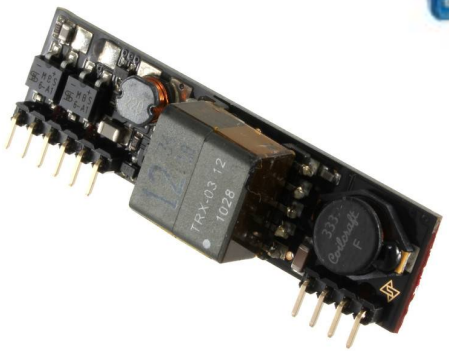

**Ethernet shield**

# Arduino UNO : l'automate

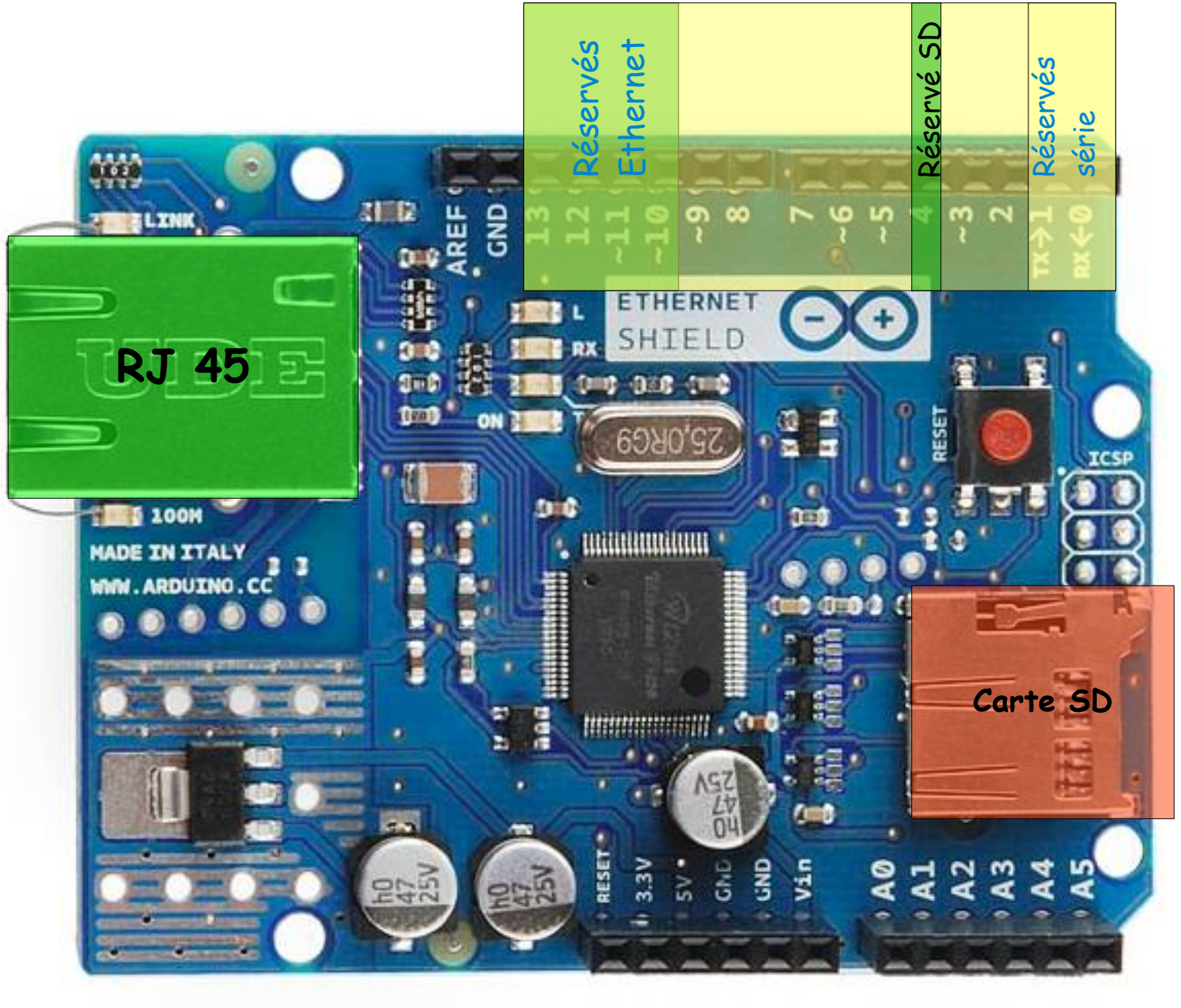


# Shield Ethernet : la carte réseau

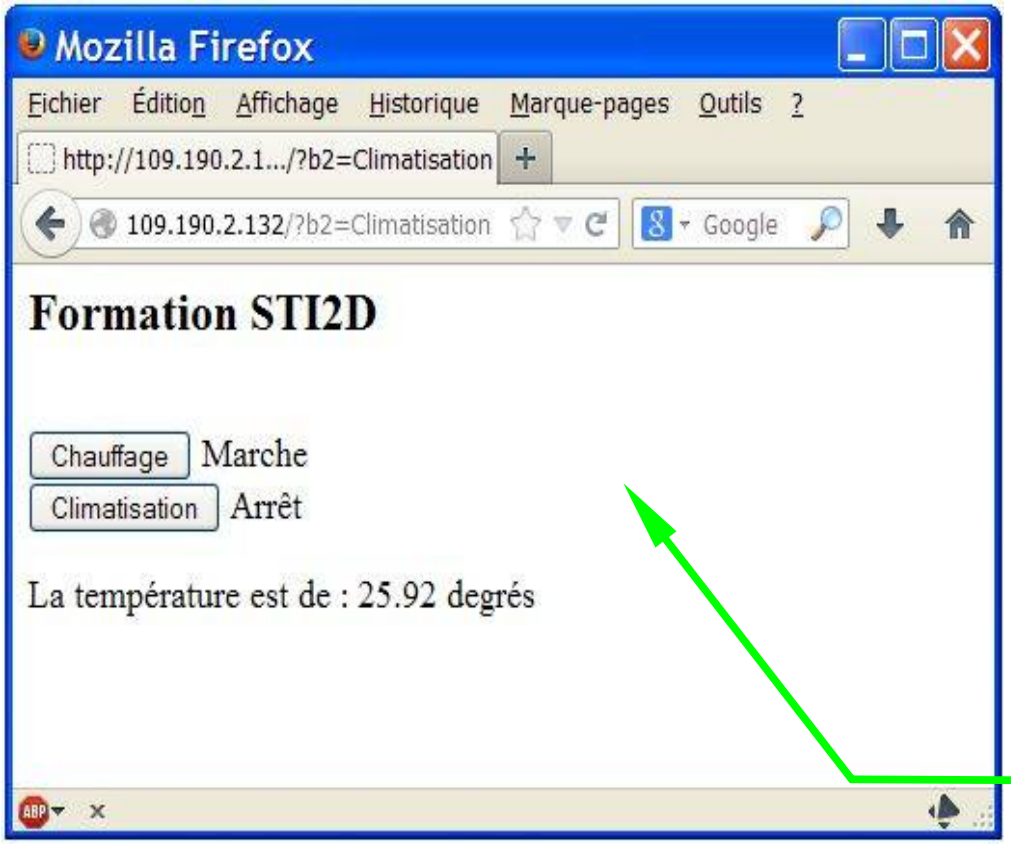


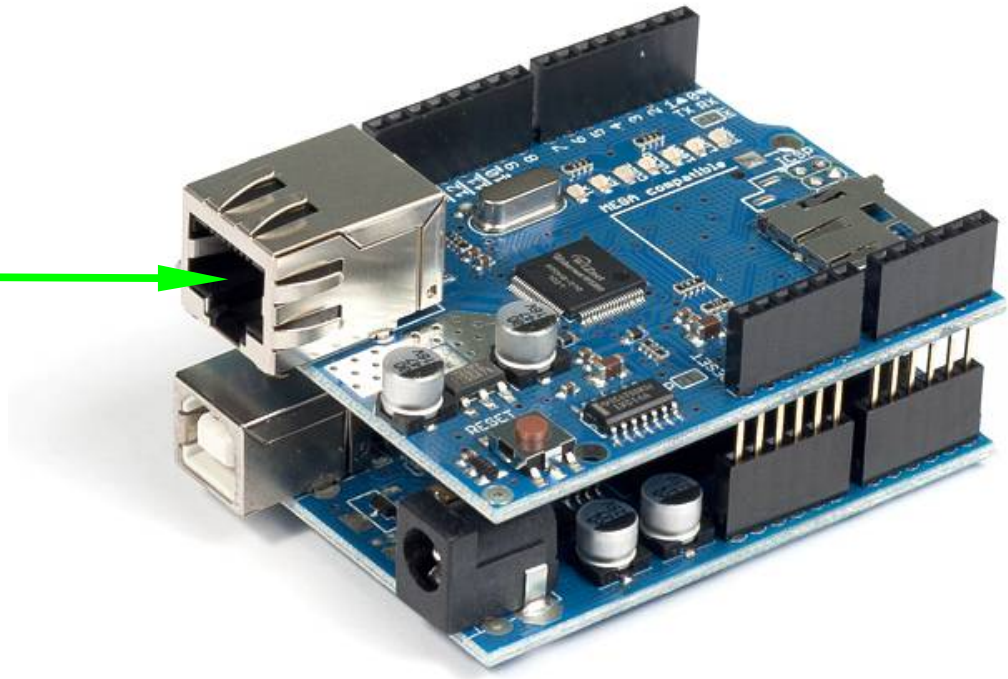
Module POE : alimentation de la carte réseau par le câble RJ


Réservés Ethernet
Réservé SD
Réservés série
RJ 45
Carte SD
ETHERNET SHIELD
LINK
100M
MADE IN ITALY
WWW.ARDUINO.CC
AREF
GND
ON
RESET
ICSP
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5

Le shield Ethernet permet de commander ou de recevoir des informations via un navigateur internet.




Le shield Ethernet transmet sur le réseau les états de l'Arduino. Celui-ci fonctionne en réseau local comme par exemple avec une ''box'' ou à travers le réseau internet.

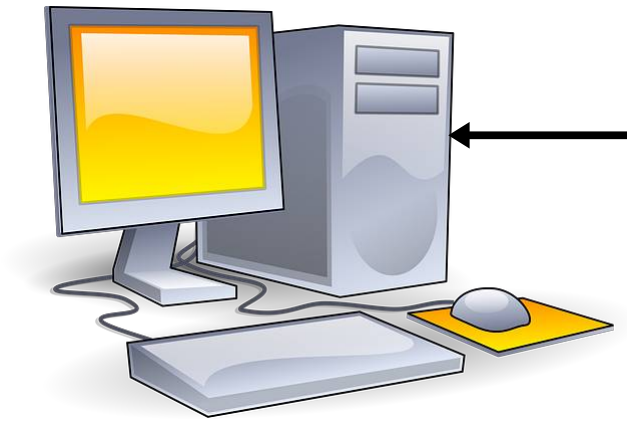


Le shield Ethernet dialogue avec l'Arduino via le bus SPI.

# Structure d'une installation et configuration

# Architecture d'un réseau local

La ''box'' sert de modem ADSL et de routeur. Le routeur se charge de diriger les ''paquets'' au bon contrôleur réseau des différentes stations. Chaque système est identifié par une adresse IP ( Internet Protocol).
-Elle peut être fixe, c'est alors la station elle- même qui l'impose au réseau local (problèmes de conflits).



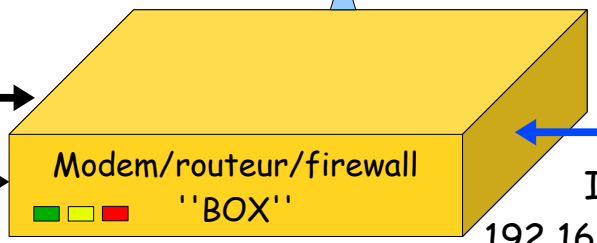
Poste 1 IP locale 192,168,1,10 (LAN)

RJ45

WIFI

ADSL
Réseau
téléphone

Internet

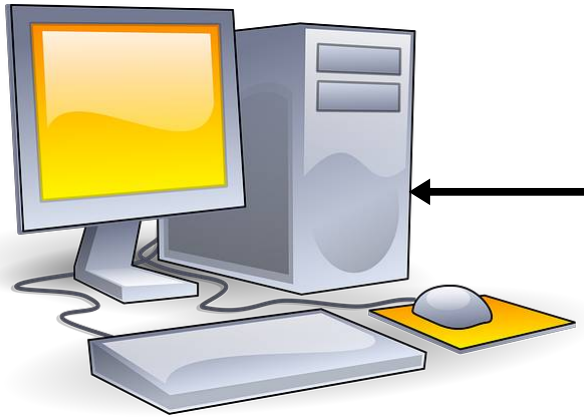



Modem/routeur/firewall
''BOX''

RJ45

IP locale
192,168,1,13 (LAN)
Plus IP publique
(WAN)

Poste 2 IP locale 192,168,1,11 (LAN)



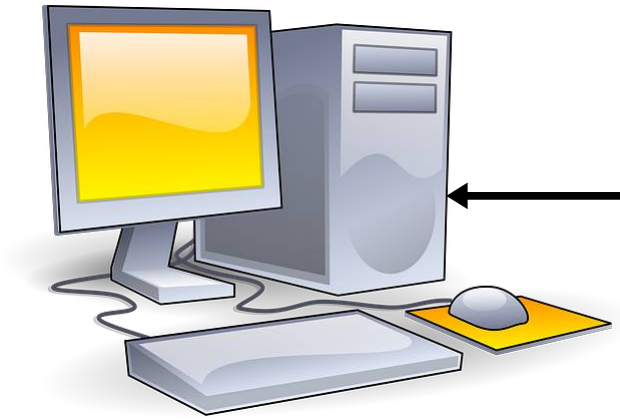
WIFI

Téléphone IP locale 192,168,1,12 (LAN)

WIFI

Tablette IP locale 192,168,1,13 (LAN)

# Branchement d'un Arduino+shield Ethernet sur un réseau local

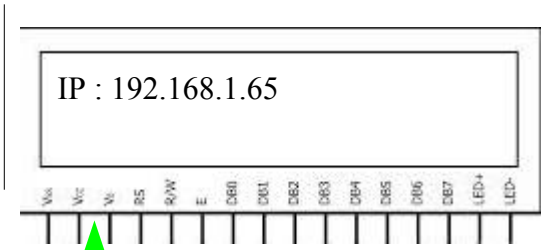


Poste 1 IP locale 192,168,1,10 (LAN)

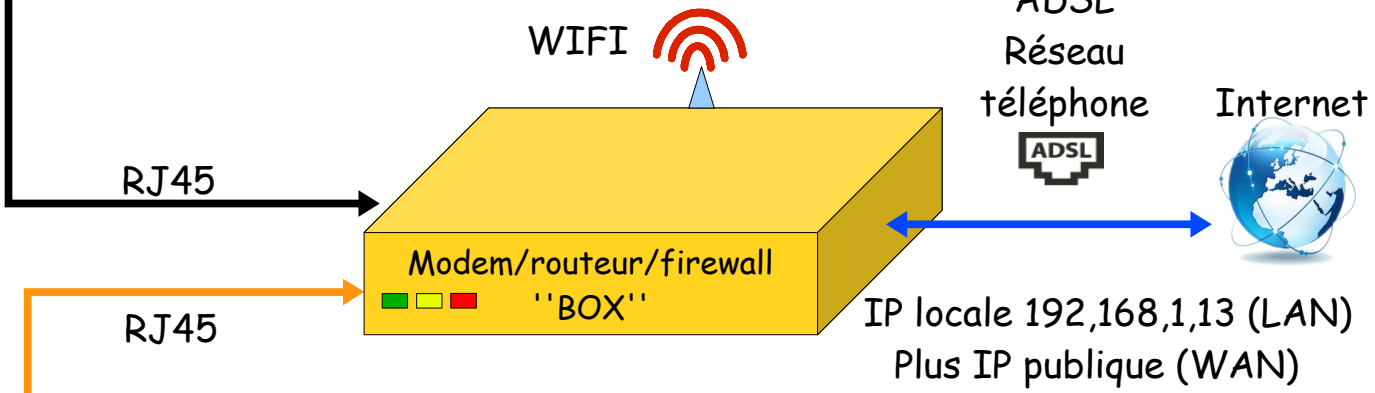
Il y a deux cas de figures :

- vous possédez un des premiers shield Ethernet et il ne fonctionne qu'en adresse IP fixe. Vous indiquerez l'adresse de votre choix dans le programme de l'Arduino. C'est aussi possible pour les Shields récents. Vous n'aurez qu'à taper l'adresse IP dans le navigateur d'un ordinateur de votre réseau pour avoir accès à votre shield.
- vous possédez un shield récent qui peut fonctionner en DHCP. C'est alors le routeur qui va attribuer une adresse IP à votre shield. Le seul problème consiste à récupérer cette adresse IP. Il est possible de demander au programme écrit dans l'Arduino de l'afficher sur le moniteur série du logiciel ''processing'' lorsque le shield est encore branché en USB.

Sinon il est possible de le faire afficher sur un écran extérieur ou bien de le lire dans la page de configuration de votre ''box'' ou du serveur.




Arduino IP locale 192,168,1,65 (LAN)

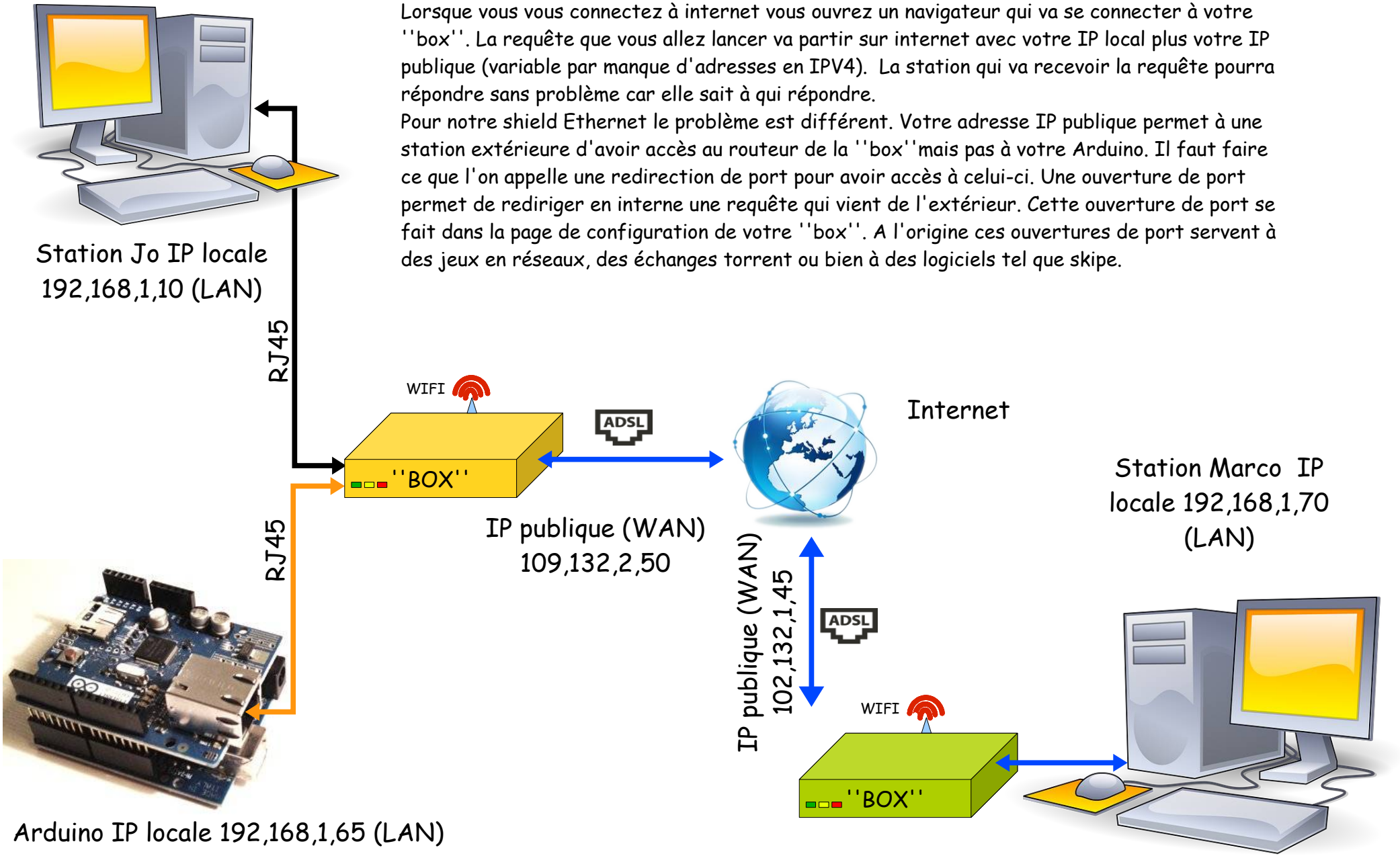



En réseau local il n'y a pas d'autre configuration à réaliser. Par contre si vous utilisez une adresse IP fixe, elle doit se trouver dans l'interval DHCP de votre ''box '' sinon votre Shield ne sera pas reconnu. Ces paramètres sont disponibles dans une page accessible en tapant l'adresse de votre ''box'' dans un navigateur.

De nombreuses ''box'' possèdent la même IP de connexion : 192,168,1,1, ou 192,162,2,254

# Lecture et écriture sur le shield via le réseau internet

Lorsque vous vous connectez à internet vous ouvrez un navigateur qui va se connecter à votre ''box''. La requête que vous allez lancer va partir sur internet avec votre IP local plus votre IP publique (variable par manque d'adresses en IPV4). La station qui va recevoir la requête pourra répondre sans problème car elle sait à qui répondre.
Pour notre shield Ethernet le problème est différent. Votre adresse IP publique permet à une station extérieure d'avoir accès au routeur de la ''box''mais pas à votre Arduino. Il faut faire ce que l'on appelle une redirection de port pour avoir accès à celui-ci. Une ouverture de port permet de rediriger en interne une requête qui vient de l'extérieur. Cette ouverture de port se fait dans la page de configuration de votre ''box''. A l'origine ces ouvertures de port servent à des jeux en réseaux, des échanges torrent ou bien à des logiciels tel que skipe.



## Fonctionnement IP locale et IP privée

Le principal intérêt de l'utilisation d'adresses IP privées est de disposer d'un grand nombre d'adresses pour bâtir ses réseaux privés (entreprises, domicile) et ainsi de palier au cruel manque d'adresses IP publiques du réseau IP V 4.

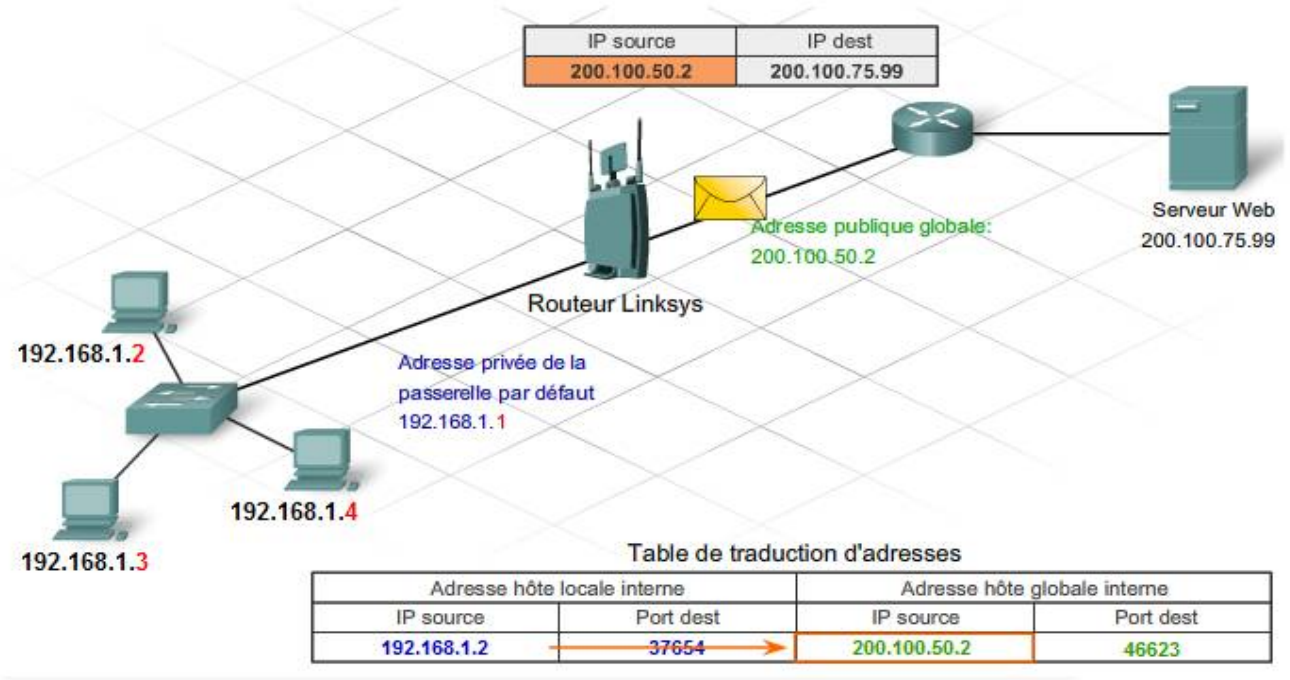
La version 6 permet de résoudre en partie ce problème en proposant pas moins de "667 millions de milliards d'adresses IP disponibles par mm2 de la surface de la Terre" (source Wikipédia).

Le déploiement d'IP v6 étant en cours (pas totalement), il est indispensable d'utiliser les technologies de NAT et de PAT pour permettre aux machines disposant d'adresses privées de pouvoir communiquer sur Internet.

Une des différences entre l'IPv4 et l'IPv6 est l'apparence des adresses. L'IPv4 utilise 4 nombres décimaux d'un octet séparé par un point (exemple : **192.168.1.1**) tandis que l'IPv6 utilise des nombres hexadécimaux séparés par colonnes (exemple : **fe80:d4a8:6435:d2d8:d9f3b11**).

| | **IPv4** | **IPv6** |
|---|---|---|
| Nombre de bits dans une adresse IP | 32 | 128 |
| Format | décimal | hexadécimal |
| Capacité d'adressage | 4.3 milliards | Nombre infini |
| Comment faire un test Ping ? | ping XXX.XXX.XXX | ping6 |

# NAT ("Network Address Translation" / Translation d'adresses)




Dispositif pour sortir du réseau privé

Source : http://isrdoc.wordpress.com

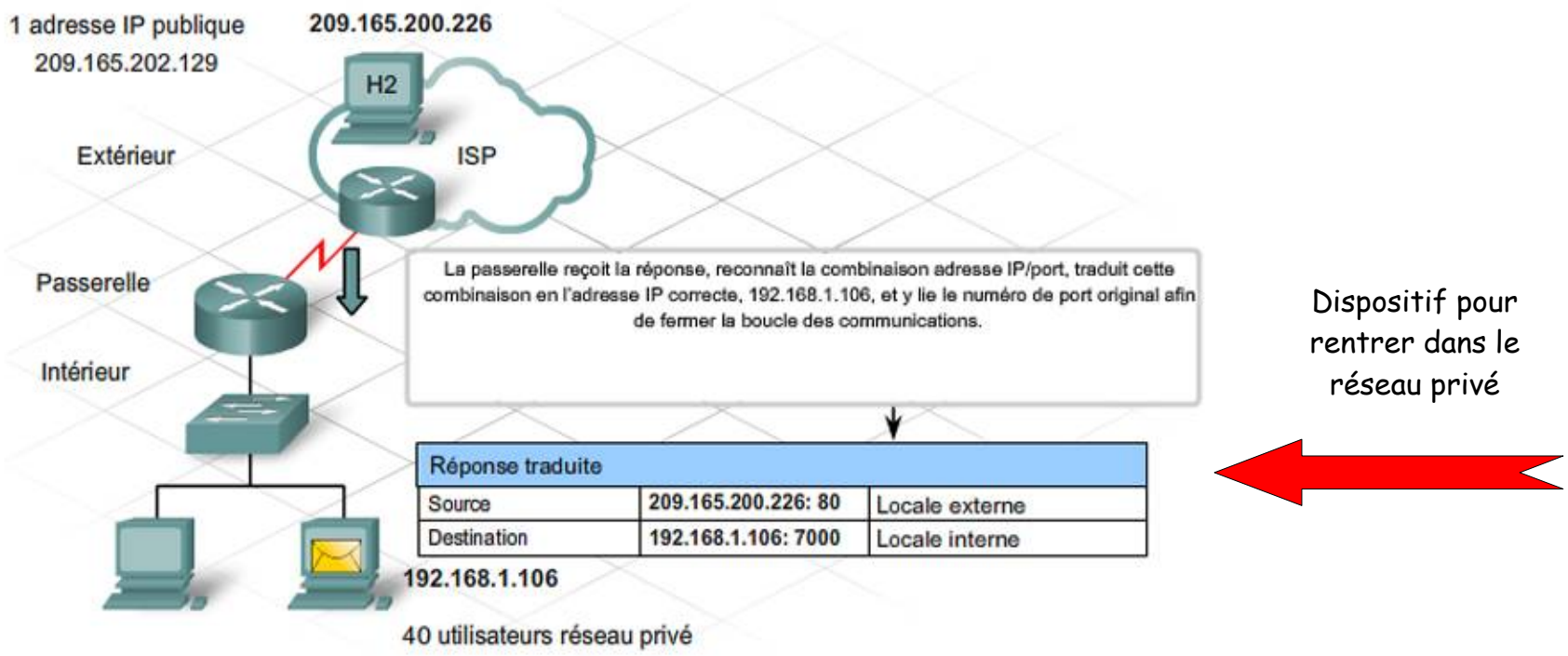
On active le mécanisme de NAT sur les routeurs faisant le lien entre les réseaux privés et publics. Le principe général est de remplacer l'adresse IP source privée de la machine par l'adresse IP publique du routeur.

L'exemple le plus répandu est celui d'un PC client domestique voulant surfer sur Internet (vers un serveur Web par exemple) à travers une ''Box'' (routeur Freebox, Livebox, box...) disposant d'une fonction de NAT dynamique.

Le PC client va émettre un paquet sur son réseau avec comme adresse source son adresse privée. La ''Box'' (qui active par défaut le mécanisme NAT dynamique), va remplacer dans le paquet l'adresse privée du PC par son adresse publique. Elle va en parallèle , garder en mémoire l'association (Adresse IP privée du PC > Adresse IP publique du serveur / Port client-serveur). Le serveur va donc recevoir par Internet ce paquet modifié auquel il va répondre avec un paquet de retour ayant pour adresse de destination l'adresse IP publique de la Box. Celle-ci va recevoir le paquet et finalement remplacer l'adresse IP publique de la Box par l'adresse privée du PC.

Il est donc possible avec une seule adresse IP publique de faire communiquer simultanément sur Internet plusieurs machines d'adresses IP privées.

# PAT ("Port Address Translation" / Translation de port)



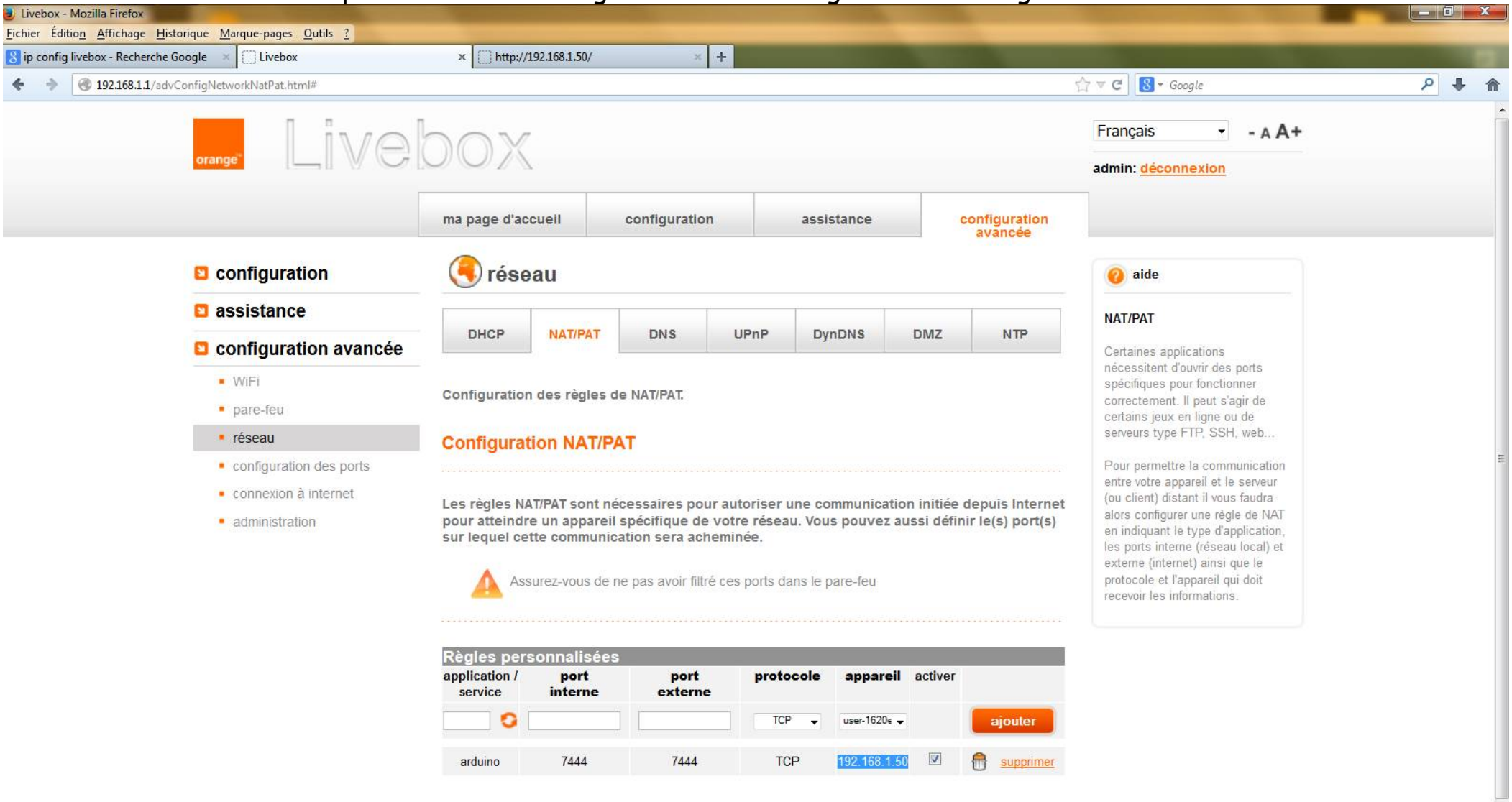

Source : http://isrdoc.wordpress.com

On ne peut pas utiliser une adresse privée pour se déplacer sur Internet. Ainsi, dans un réseau disposant d'une plage d'adresse IP privée, il va falloir ruser. En effet, dans ce cas précis, le NAT n'est d'aucune utilité car il  ne fonctionne que pour les sessions à l'initiative des machines se trouvant sur le réseau privé. Dans notre cas, nous avons besoin d'un mécanisme permettant de rendre visible une machine depuis Internet. C'est le PAT qui va nous offrir cette fonctionnalité.

Prenons l'exemple d'une personne voulant héberger son serveur Web (en écoute sur le port TCP/80) chez lui, derrière sa ''Box'' :

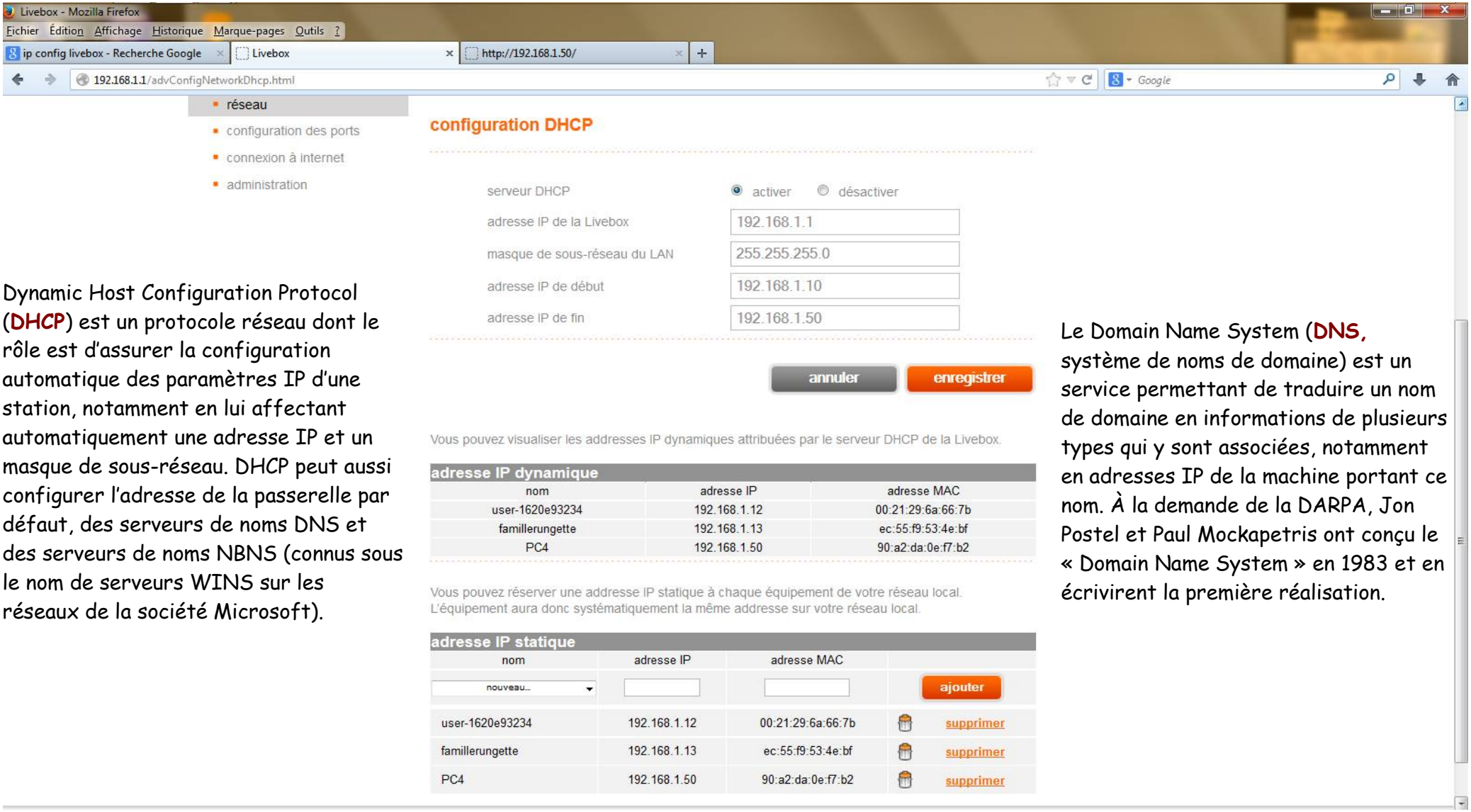
Le client va envoyer une requête HTTP vers l'adresse IP publique de la Box (via la résolution DNS). La Box, préalablement configurée avec une redirection du port 80 vers le serveur (PAT), va remplacer l'adresse destination du paquet (l'adresse publique de la Box) par celle du serveur (l'adresse privée du serveur). Le serveur va ensuite répondre en utilisant son adresse IP privée comme adresse source. La Box va ensuite remplacer celle-ci par son adresse IP publique.

# Ouverture de ports et configuration de la''box''
## Exemple ''Livebox'' Orange Télécom® configuration des règles de NAT/PAT




**NAT** ("Network Address Translation" / Translation d'adresses) et **PAT** ("Port Address Translation" / Translation de port).

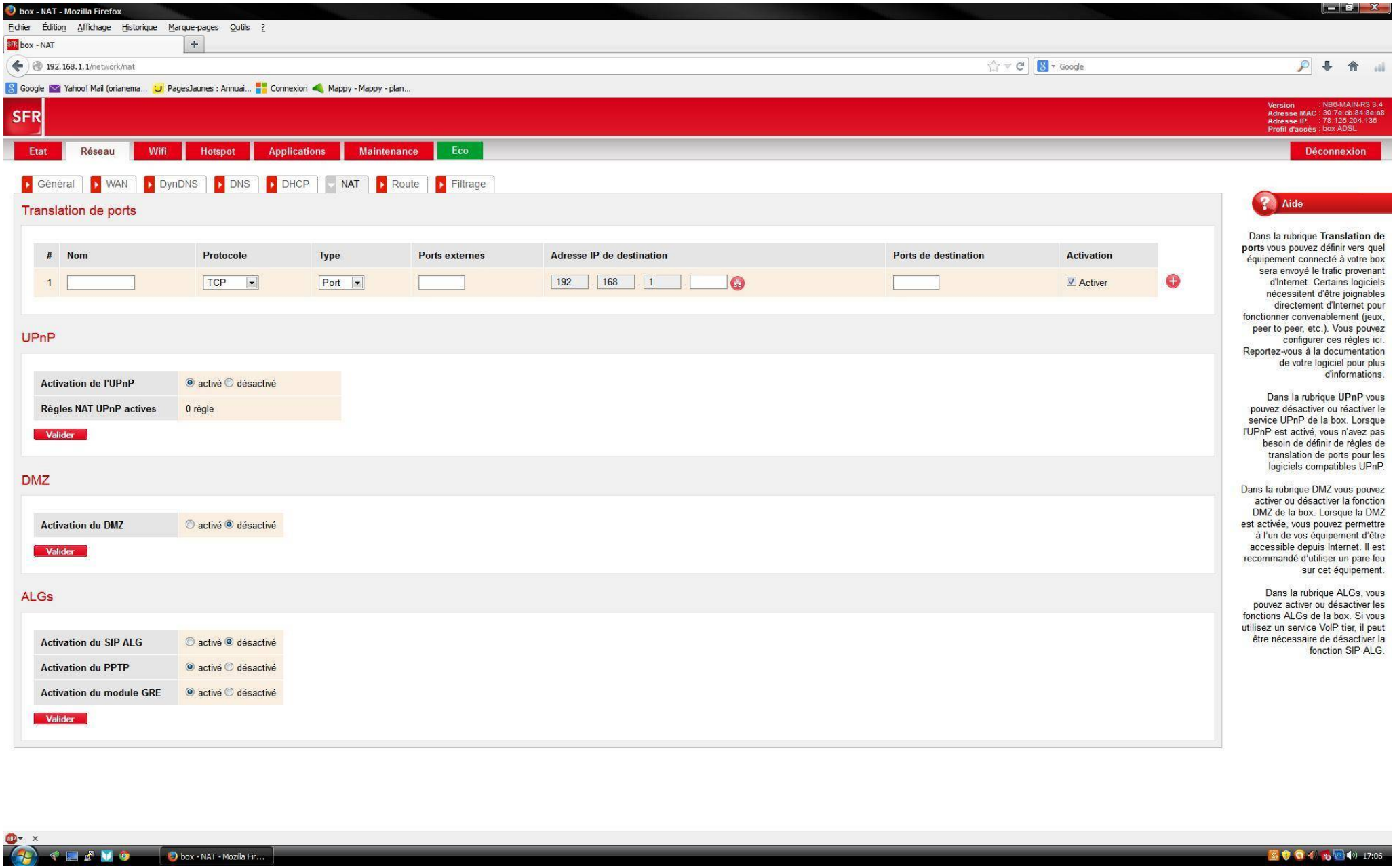
# Exemple ''Livebox'' Orange Télécom® configuration DHCP
## L'adresse IP fixe de l'arduino doit se trouver dans l'interval DHCP de la ''Box''



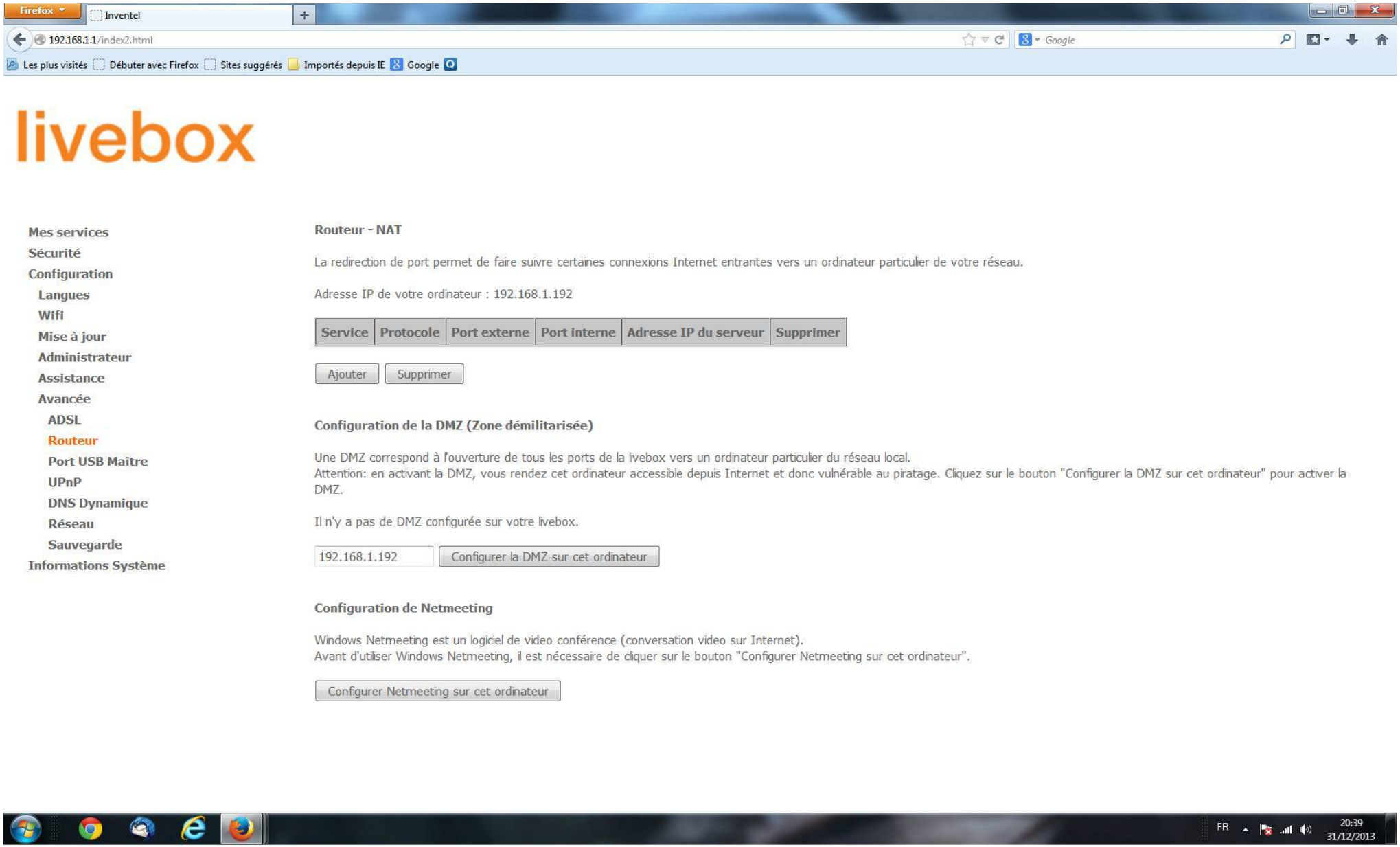

Dynamic Host Configuration Protocol (**DHCP**) est un protocole réseau dont le rôle est d'assurer la configuration automatique des paramètres IP d'une station, notamment en lui affectant automatiquement une adresse IP et un masque de sous-réseau. DHCP peut aussi configurer l'adresse de la passerelle par défaut, des serveurs de noms DNS et des serveurs de noms NBNS (connus sous le nom de serveurs WINS sur les réseaux de la société Microsoft).

Le Domain Name System (**DNS,** système de noms de domaine) est un service permettant de traduire un nom de domaine en informations de plusieurs types qui y sont associées, notamment en adresses IP de la machine portant ce nom. À la demande de la DARPA, Jon Postel et Paul Mockapetris ont conçu le « Domain Name System » en 1983 et en écrivirent la première réalisation.

# Il est aussi possible d'utiliser le Shield Ethernet en DHCP Exemple ''box'' SFR® configuration NAT

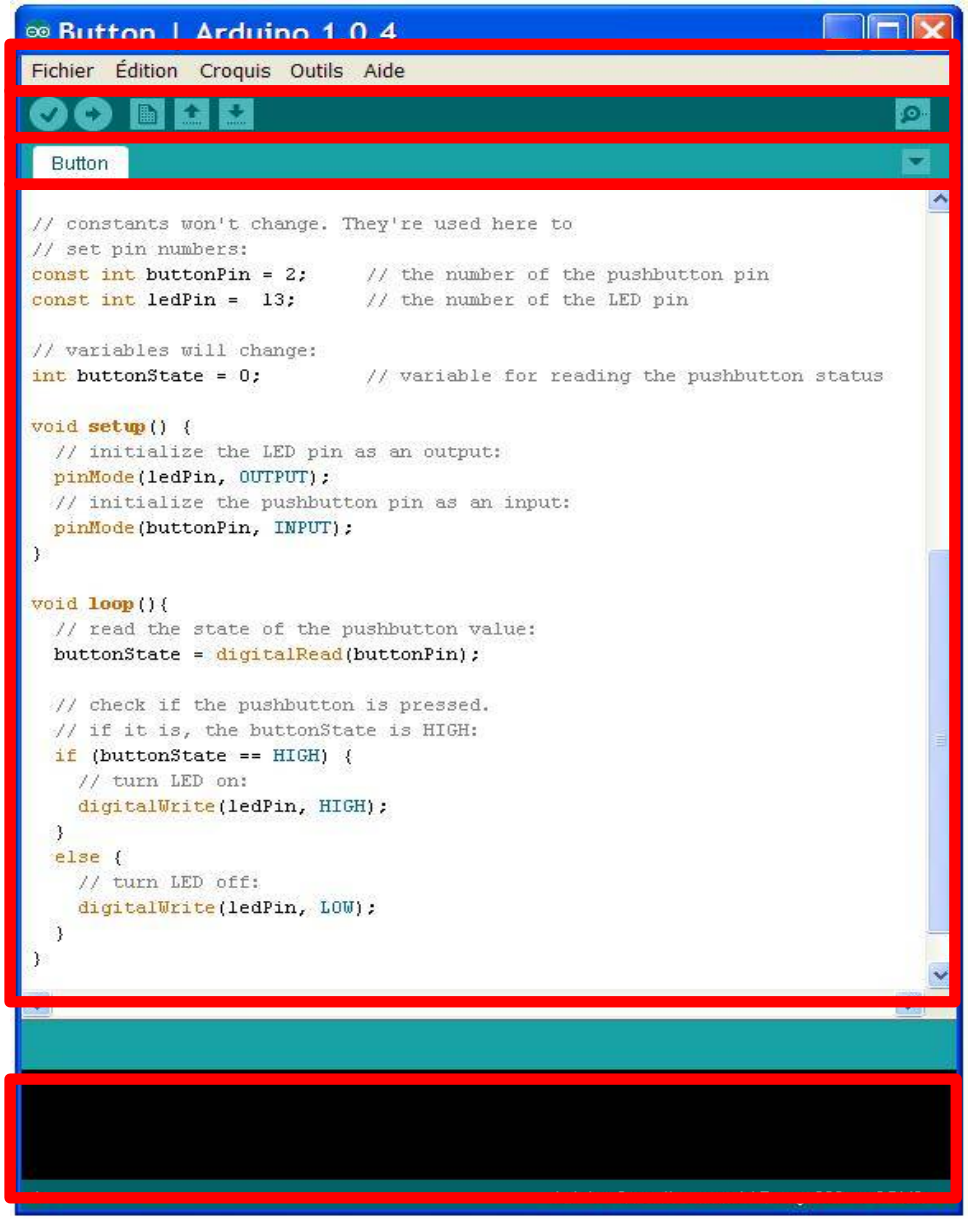


# Exemple ''Livebox'' Orange Télécom® (autre version)



# Rappel sur les bases de programmation et la structure d'un programme

Maintenant que nous avons une petite idée de la structure d'une installation nous allons rentrer dans le vif du sujet.




1. un menu
2. une barre d'actions
3. un ou plusieurs onglets correspondant aux ''sketchs''
4. une fenêtre de programmation
5. une console affiche les informations, erreurs de compilation et le téléversement du programme

### Coloration syntaxique

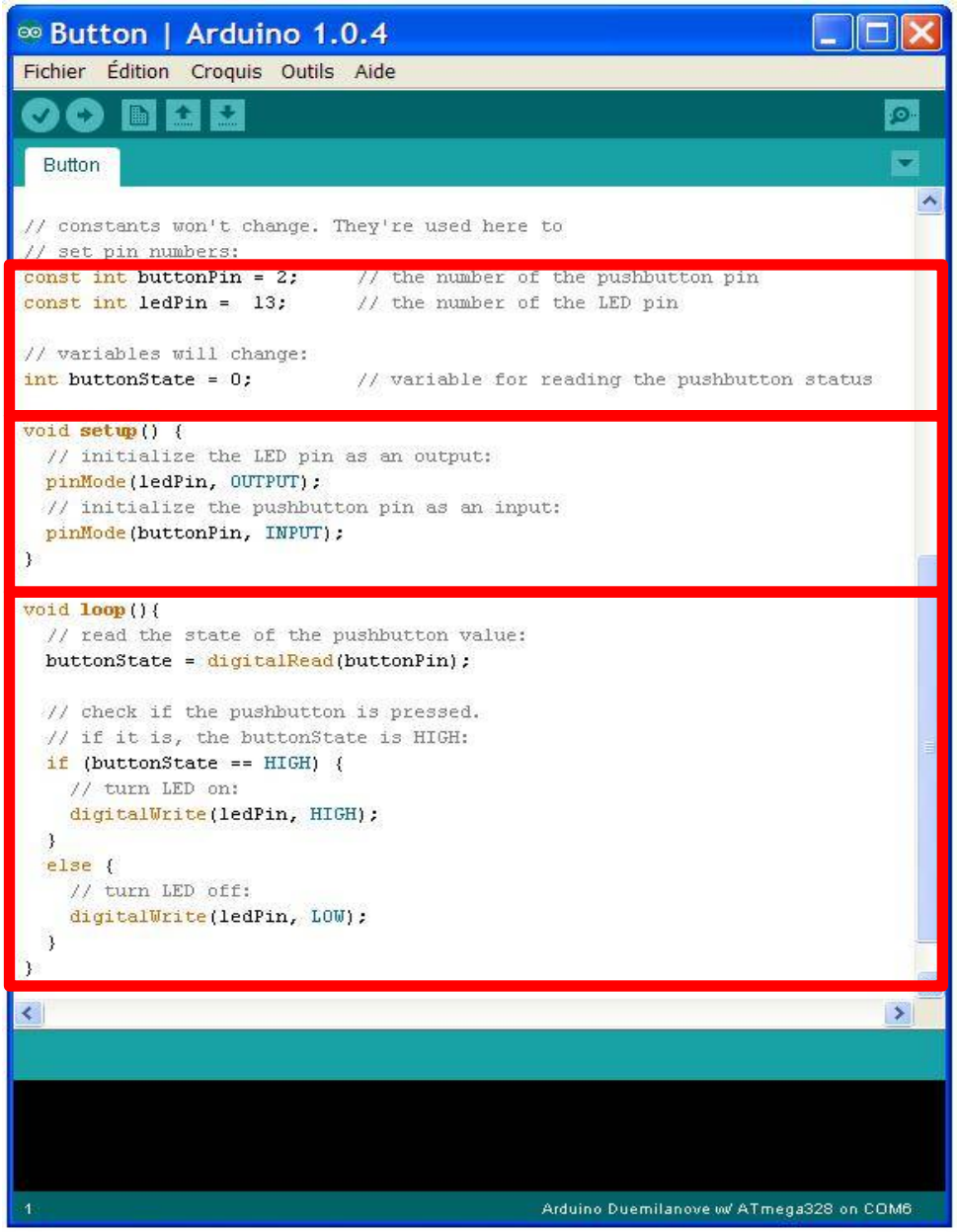
Lorsque du code est écrit dans l'interface de programmation, certains mots apparaissent en différentes couleurs qui clarifient le statut des différents éléments :

En **orange**, apparaissent les mots-clés reconnus par le langage Arduino comme des fonctions existantes. Lorsqu'on sélectionne un mot coloré en orange et qu'on effectue un clic avec le bouton droit de la souris, on a la possibilité de choisir « Find in reference » : cette commande ouvre directement la documentation de la fonction sélectionnée.

En **bleu**, apparaissent les mots-clés reconnus par le langage Arduino comme des constantes.

En **gris**, apparaissent les commentaires qui ne seront pas exécutés dans le programme. Il est utile de bien commenter son code pour s'y retrouver facilement ou pour le transmettre à d'autres personnes. On peut déclarer un commentaire de deux manières différentes :

## Structure d'un programme Arduino




//Commentaires (1 ligne)
/* zone de commentaire*/ pour éviter de se perdre dans un programme.

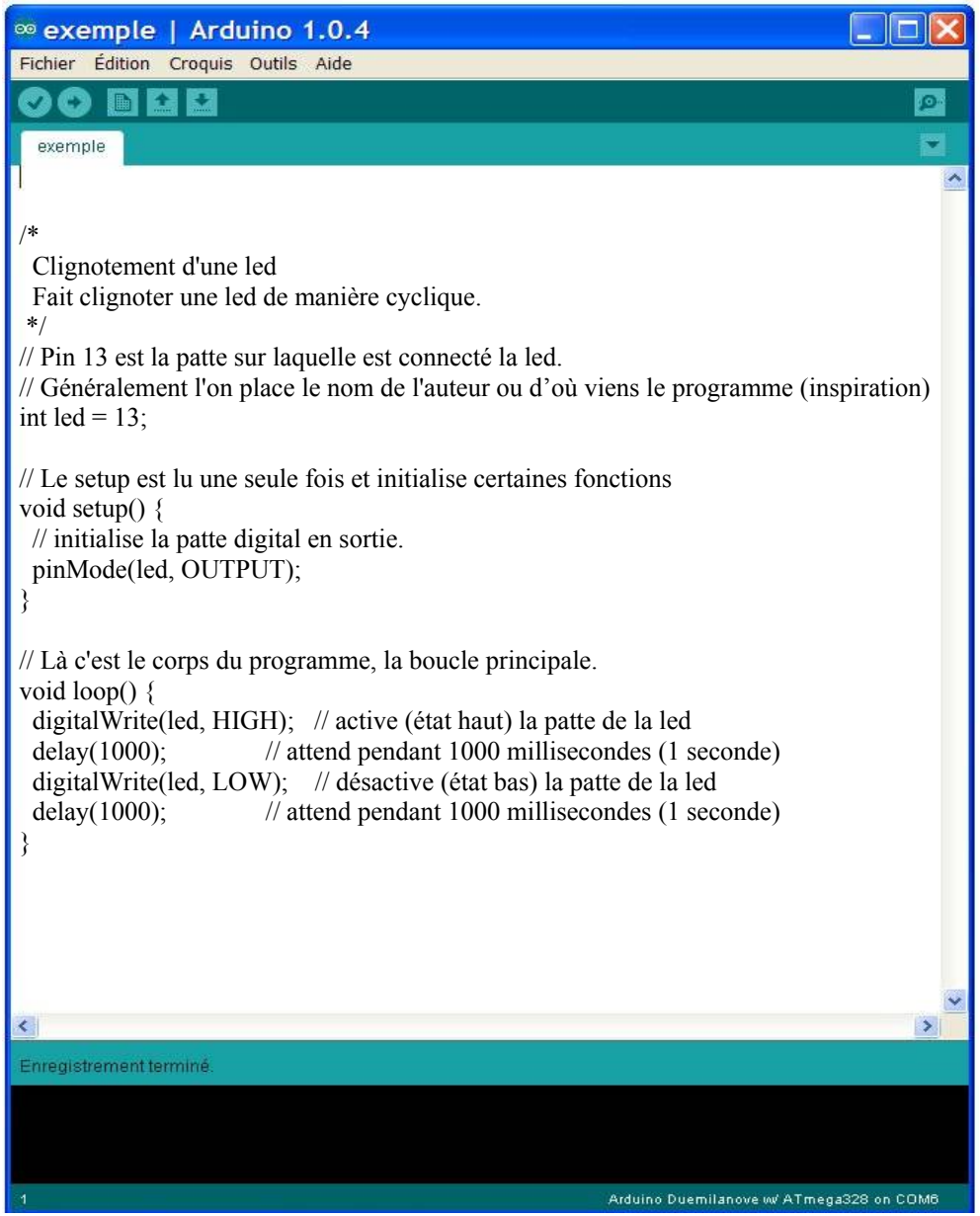
❶ la partie déclaration des variables (optionnelle)

❷ la partie initialisation et configuration des entrées/sorties : la fonction setup {}

❸ la partie principale qui s'exécute en boucle : la fonction loop {}

Dans chaque partie d'un programme sont utilisées différentes instructions issues de la syntaxe du langage Arduino.

Source :fr.flossmanuals.net. Flossmanuals Propose des livres libres sur les logiciels libres (très bien fait, documents dans la boite à outils)




## Commençons par la ponctuation

Le code est structuré par une ponctuation stricte :

- **toute ligne** de code se termine par un point-virgule **;**
- le contenu d'une **fonction** est délimité par des accolades **{** et **}**
- les **paramètres** d'une fonction sont contenus par des parenthèses **(** et **)**.

## Les variables

Une variable est un espace réservé dans la mémoire de l'Arduino. C'est comme un compartiment dont la taille n'est adéquate que pour un seul type d'information. Elle est caractérisée par un nom qui permet d'y accéder facilement.

Il existe différents types de variables identifiées par un mot-clé dont les principaux sont :

- nombres entiers (int)
- nombres à virgule flottante (float)
- texte (String et string, attention à la casse)
- valeurs vrai/faux (boolean).

Un nombre décimale, par exemple *3.14159*, peut se stocker dans une variable de type float. Notez que l'on utilise un point et non une virgule pour les nombres décimaux. Dans Arduino, il est nécessaire de déclarer les variables pour leur réserver un espace mémoire adéquat. On déclare une variable en spécifiant son type, son nom puis en lui assignant une valeur initiale

Exemple : int buttonPin = 2;

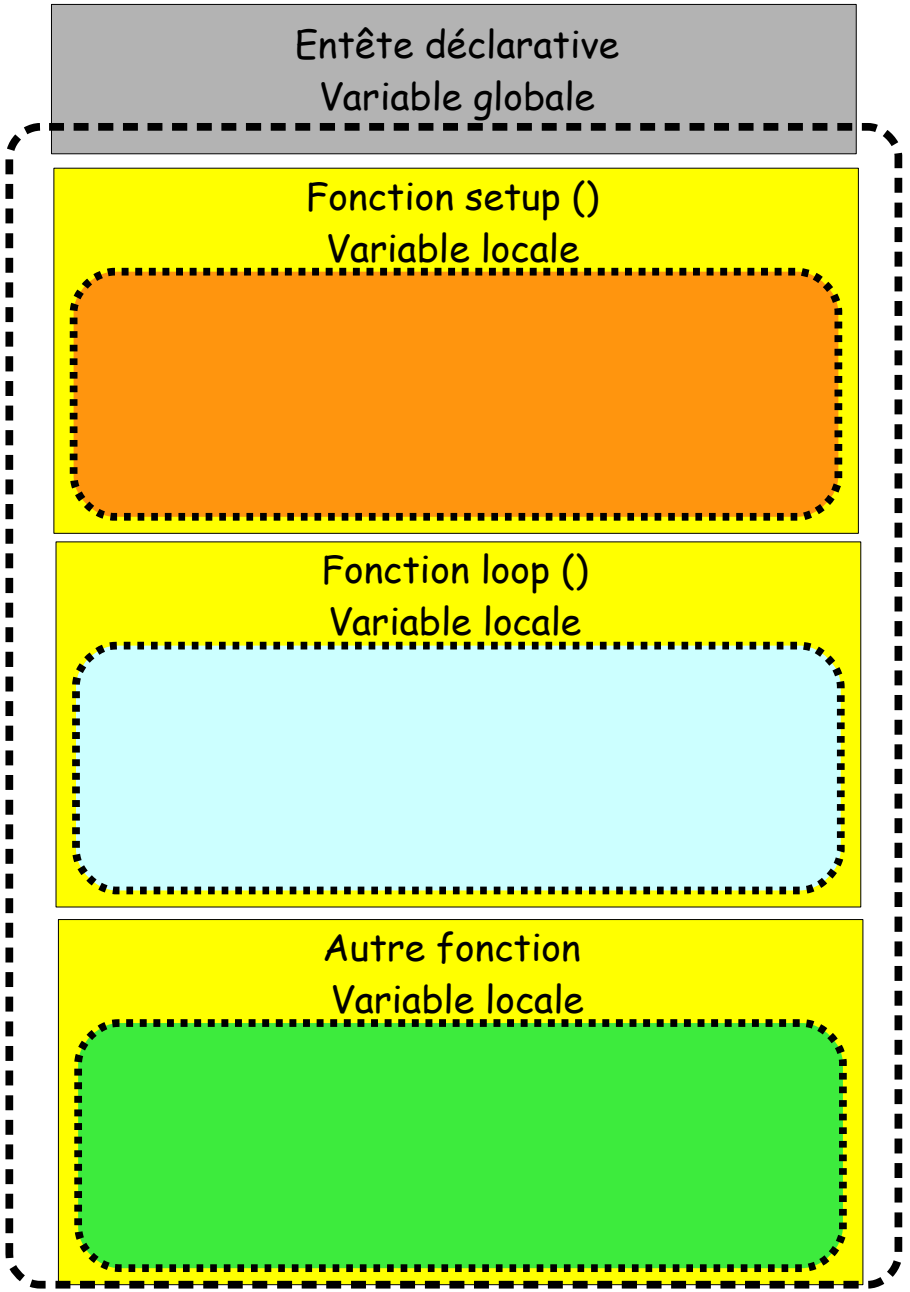
La variable s'appelle buttonPin, elle fonctionne avec des nombres entiers et sa valeur par défaut est 2.

Une variable est une ''boîte'' qui contient une donnée. Ces données peuvent être de plusieurs types :

| nom | caractéristiques | sans signe unsigned | exemples |
|---|---|---|---|
| **Données numériques** | | | |
| **byte** | 0 à 255 | Déclare une variable de type octet (8 bits) qui stocke un nombre entier non-signé | byte a=145 ; // la donnée a est égal à 145<br>byte a= b10010 ; // la donnée est en binaire et elle est égale à 18 en DEC |
| **int** | -32536 à +32535 | 0 à +65535 | int b=-2458 ; //la donnée b est égale à -2458 |
| **word** | 0 à +65535 | représente int unsigned | word w = 10000 ; |
| **long** | -2147483648 à +2147483647 | 0 à +4294967295 | long vitessemoteur = 186000L; //déclare une variable de type long |
| **float** | Nombre à virgule | $-3,4028235*10^{38}$ à $+3,4028235*10^{38}$ | float capteurvitesse = 1.117; // déclare une variable à virgule appelée capteurvitesse |
| **Données logiques** | | | |
| **boolean** | 0 ou 1, vrai ou faux true ou false occupe 1 octet | | boolean marche = false ; // la variable marche est fausse |
| **Void** | | | void setup () // la fonction setup ne fournit aucun résultat |
| **Données caractères ou chaîne de caractères** | | | |
| **String** | Chaîne de caractères évoluée | Les chaînes de caractères sont représentées sous forme de tableau de variables de type char et se termine par un zéro<br>[ ] | char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};<br>char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};<br>char Str4[ ] = "arduino";<br>char Str5[8] = "arduino";<br>char Str6[15] = "arduino"; |
| **string** | Chaîne de caractères | | |
| **char** | Tout caractères (code ASCII) -128 à +127 | 0 à +255 code ASCII étendu | Char lettreA = 'A' // la donnée lettreA est égale au caractère A, soit 65 en décimal qui est le code ASCII de A |

Déclaration d'une variable avec initialisation : type mavariable=valeur ;

## Comment et ou déclarer une variable

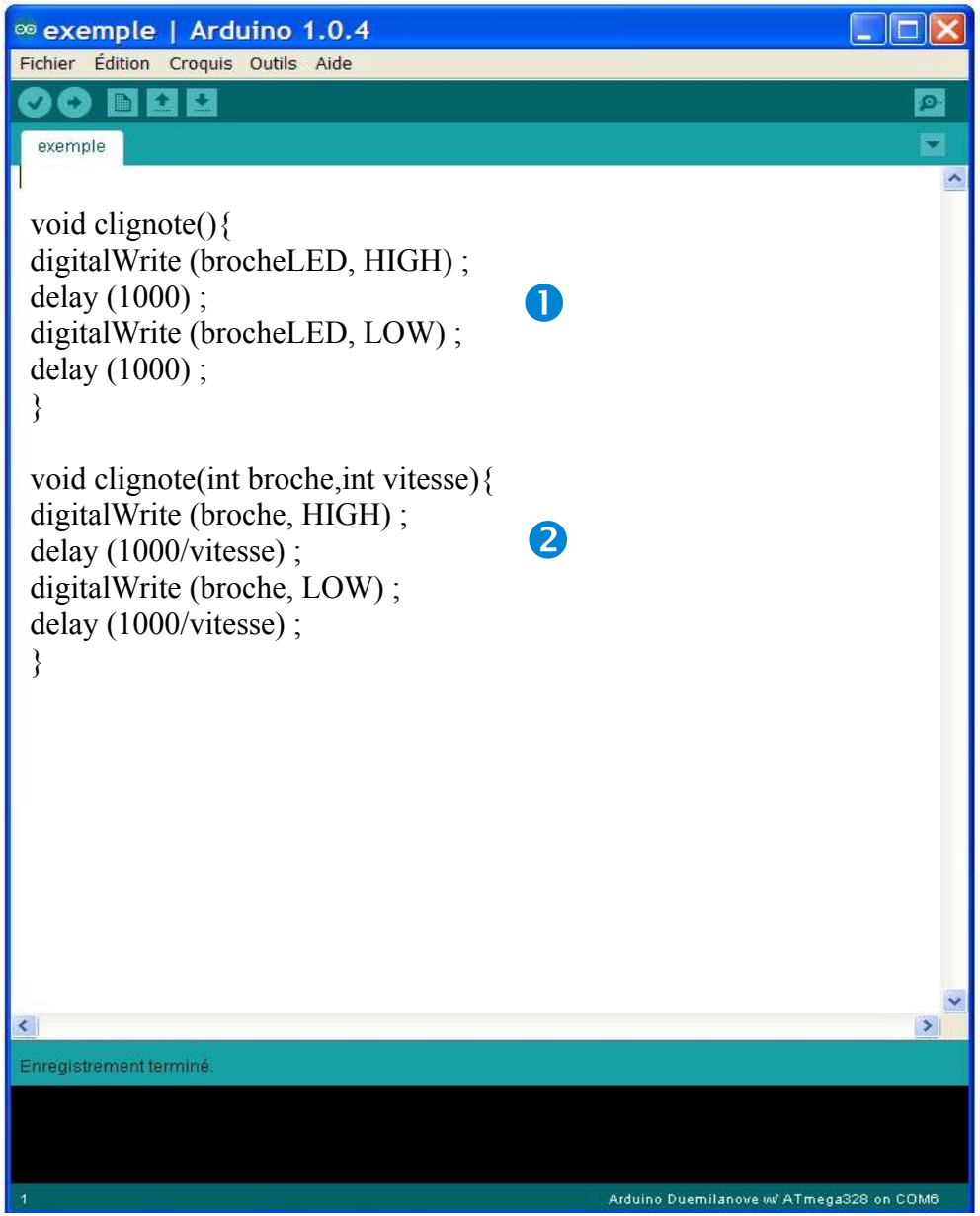



Globale : la variable est visible et utilisable depuis tout les parties du programme

Locale : la variable est visible et utilisable que dans une certaine partie du programme

Il est est également pratique de déclarer et d'initialiser une variable à l'intérieur d'une boucle for. Ceci crée une variable qui ne sera accessible uniquement à l'intérieur des accolades { } de la boucle for.

Il est préférable de déclarer les variables avant les parties setup et loop. Elles seront vues depuis n'importe quelle partie du programme.




## Les fonctions

Une fonction (également désignée sous le nom de procédure ou de sous-routine) est un bloc d'instructions que l'on peut appeler à tout endroit du programme.
Le langage Arduino est constitué d'un certain nombre de fonctions, par exemple analogRead(), digitalWrite() ou delay().

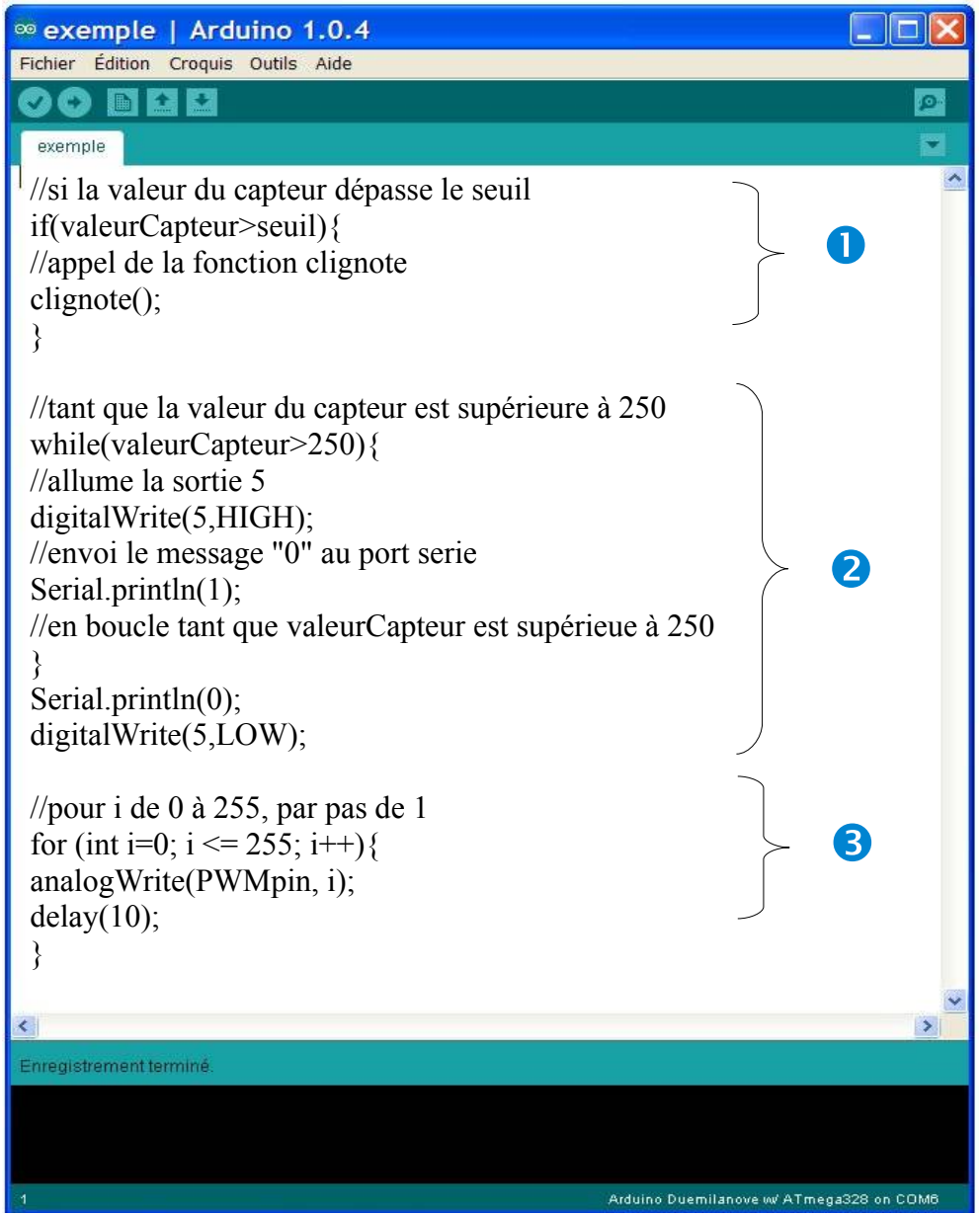
Il est possible de déclarer ses propres fonctions par exemple : 1
Pour exécuter cette fonction, il suffit de taper la commande :
clignote();

On peut faire intervenir un ou des paramètres dans une fonction : 2

Dans ce cas, l'on peut moduler leurs valeurs depuis la commande qui l'appelle :
clignote(5,1000); //la sortie 5 clignotera vite
clignote(3,250); //la sortie 3 clignotera lentement

## Les sous-routines

Quand votre code commence à tenir une place importante et que vous utilisez à plusieurs reprises les mêmes blocs d'instructions, vous pouvez utiliser une sous-routine qui vous permet de mieux organiser et d'alléger votre programme. Les sous-routines doivent être écrites après la boucle principale.




## Les structures de contrôle

Les structures de contrôle sont des blocs d'instructions qui s'exécutent en fonction du respect d'un certain nombre de conditions.
Il existe quatre types de structure :

**If** et **else** : exécutent un code **si** certaines conditions sont remplies et éventuellement exécuteront un autre code avec **sinon**.

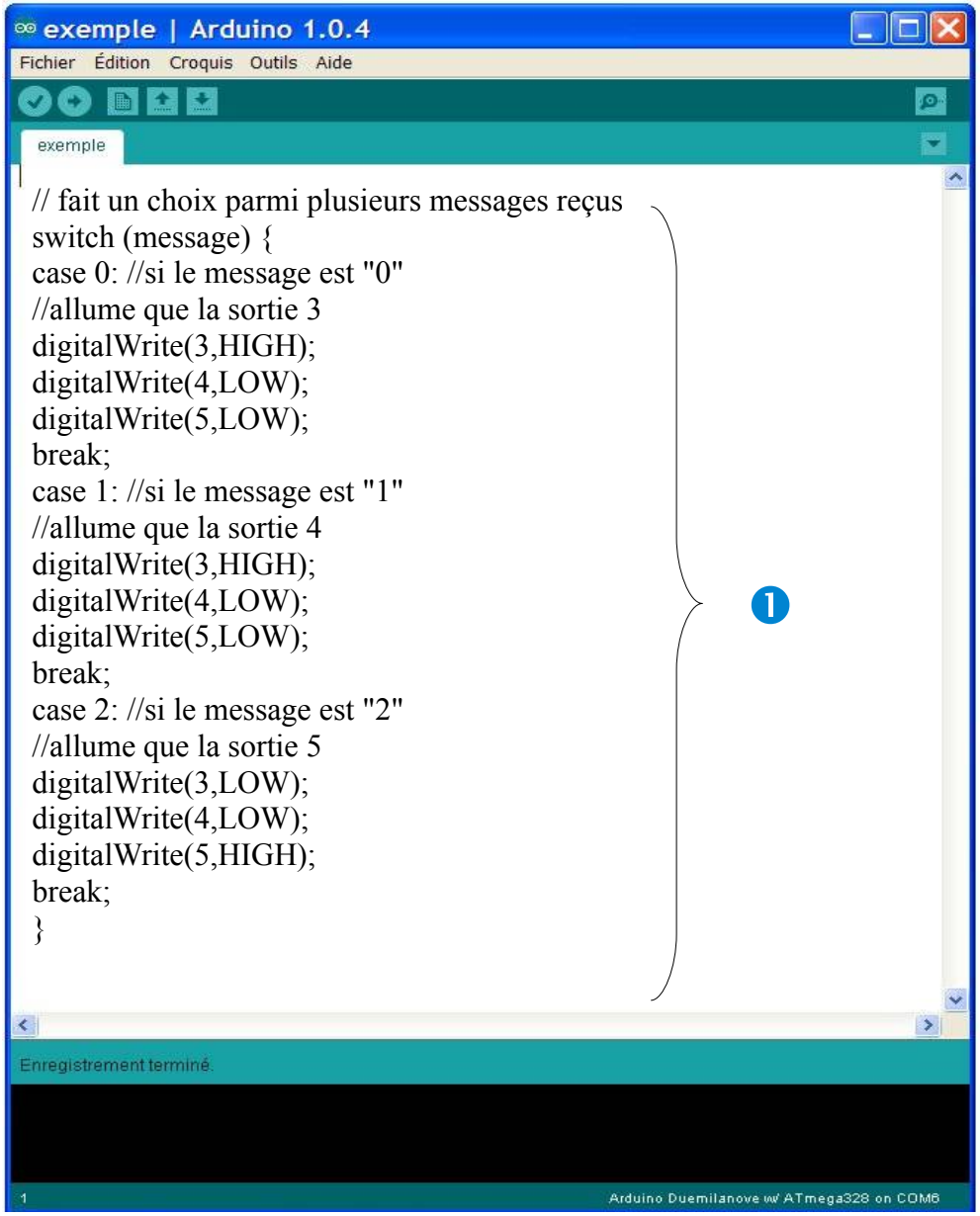
Exemple : 1

**while** : exécute un code **tant** que certaines conditions sont remplies.

Exemple : 2

**for** : exécute un code **pour** un certain nombre de fois.

Exemple : 3




**switch/case** : fait un choix entre plusieurs codes **parmi une liste de possibilités**

Exemple : 1

**Indenter son code**

Il est conseillé d'effectuer un retrait par rapport à la ligne précédente à chaque nouveau bloc d'instructions. Les blocs d'instructions inscrits dans les fonctions et boucles sont délimités par des accolades, l'une est ouvrante et l'autre fermante.

```
if (etatCaptation == 1) {
if (valeurCapteur >= seuil) {
analogWrite (pinLed, HIGH);}
}
```

Le code ci-dessus n'est pas indenté. Après indentation, nous obtenons ceci :

```
if (etatCaptation == 1) {
        if (valeurCapteur >= seuil) {
        analogWrite (pinLed, HIGH);
        }
}
```

Cela n'est bien sur qu'une légère approche, vous trouverez beaucoup de ressources sur internet.

La liste exhaustive des éléments de la syntaxe du langage Arduino est consultable sur le site :

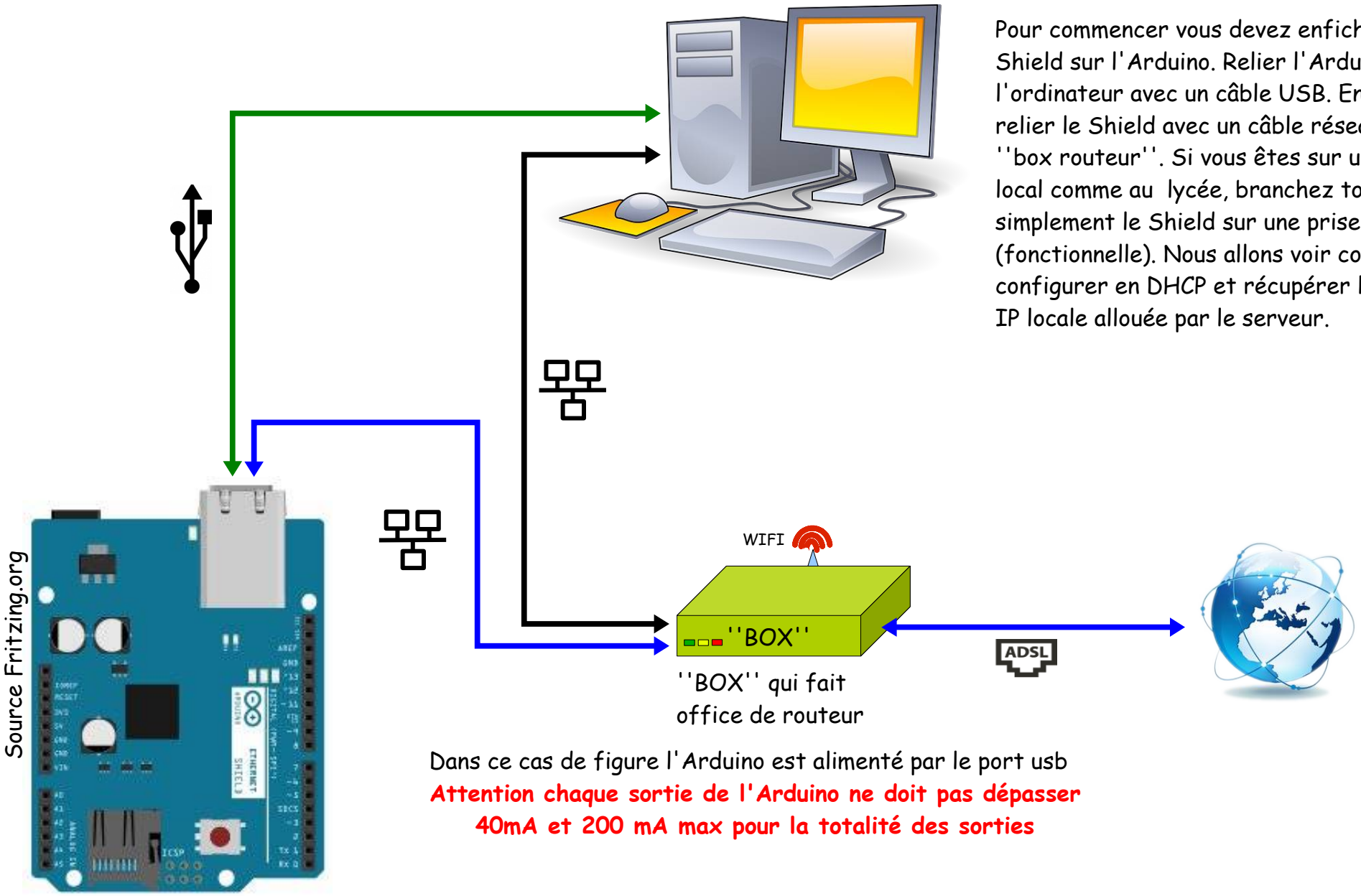
http://arduino.cc/en/Reference/HomePage

Ces pages sont inspirées du site :

http://fr.flossmanuals.net

# Exemple associé à l'éolienne

# Exemple avec le Shield Ethernet : structure du système




Source Fritzing.org

''BOX'' qui fait office de routeur

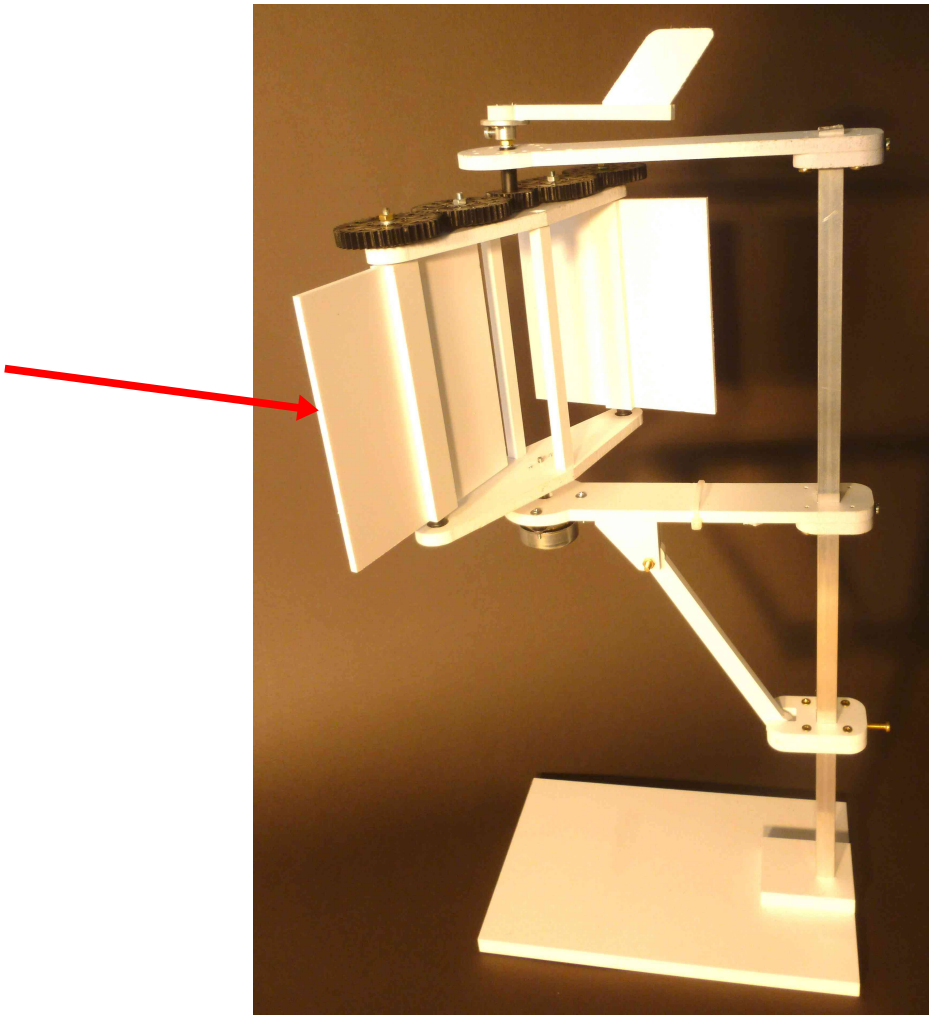
Pour commencer vous devez enficher le Shield sur l'Arduino. Relier l'Arduino à l'ordinateur avec un câble USB. Ensuite relier le Shield avec un câble réseau à votre ''box routeur''. Si vous êtes sur un réseau local comme au lycée, branchez tout simplement le Shield sur une prise réseau (fonctionnelle). Nous allons voir comment le configurer en DHCP et récupérer l'adresse IP locale allouée par le serveur.

Dans ce cas de figure l'Arduino est alimenté par le port usb

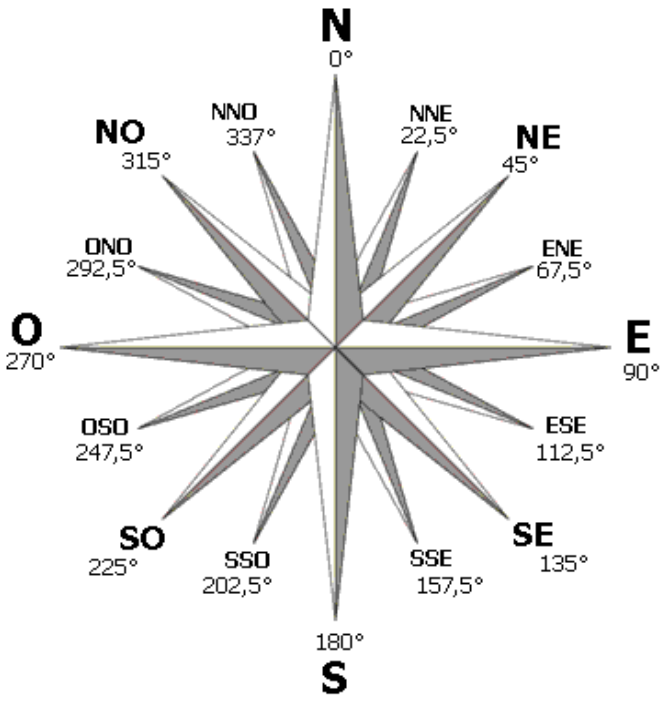
**Attention chaque sortie de l'Arduino ne doit pas dépasser 40mA et 200 mA max pour la totalité des sorties**

# Capter la position du vent

Afin de gérer l'éolienne nous allons remplacer la girouette par un actionneur (moteur pas à pas, servo-moteur ou moteur à courant continu suivant le cas)

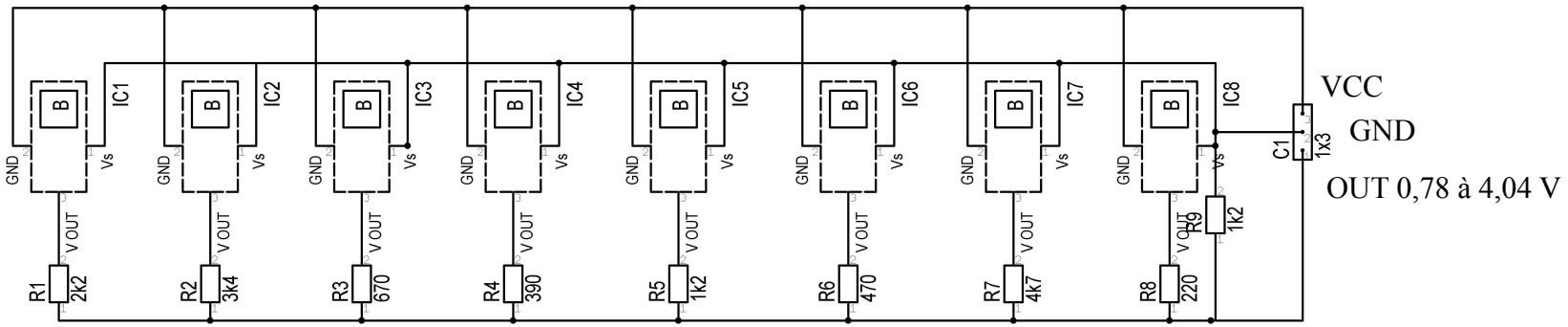


Dans tous les cas la synchronisation nécessite un rapport de deux entre la référence au vent et les pales.




Pour ce faire nous avons besoin d'un capteur qui nous indique la position du vent. Pour avoir une précision correct nous avons besoin de 16 positions (22,5°). Nous ne pouvons bien-sûr pas utiliser les entrées numériques, il n'y en a que 13 (+6) . Nous allons utiliser du ''faux multiplexage''. Il y a sur l'Arduino 6 entrées analogiques A0 à A5 codées en 10bits soit 5 volts/1024 qui nous donne 4,88 mVolts par pas. Il nous faudrait un changement tous les 0,312 volts dans l'idéal. En réalité c'est moins simple. Certains systèmes utilisent des contacts ''reed''souvent appelés ''ILS''. C'est un système à déconseiller à cause de l'usure des contacts et aux rebonds de linguets qui compliquent la programmation de l'Arduino. Fort heureusement, il existe un petit capteur à effet Hall : le TLE4905. Il a l'avantage d'être précis (la rapidité n'est pas nécessaire dans ce cas). Son coût est à peine plus important qu'un ''ILS''(1,8 euros). Il fonctionne indifféremment de 3Volts à 32 volts.

Les 8 capteurs sont montés sur un pont diviseur de tension.




L'astuce est dans le fait que l'aimant peut commander deux capteurs, donc nous avons une information intermédiaire, soit 22,5° (360° /16).

Tableau de construction :

Les valeurs ne sont pas linéaires, elles correspondent aux valeurs normalisées des résistances utilisées.

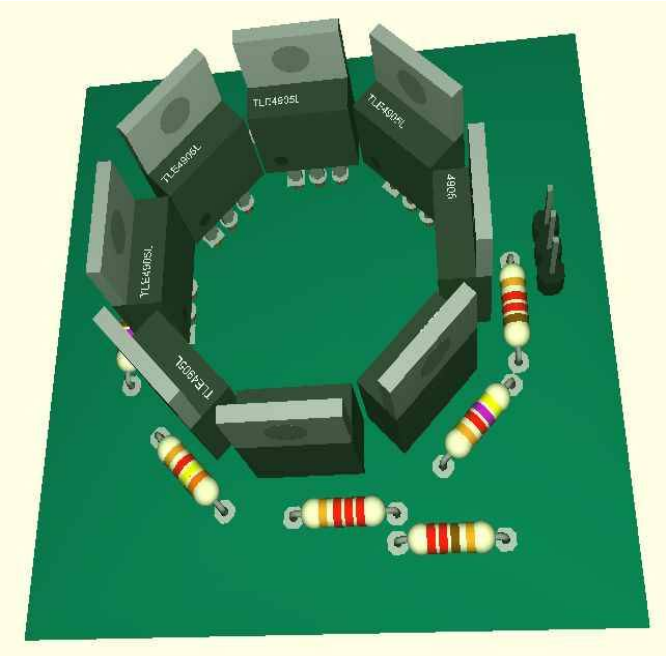
L'aimant peut faire basculer deux capteurs ce qui nous donne les 16 valeurs.

La mise en parallèle des deux résistances adjacentes du pont diviseur conditionne le montage.

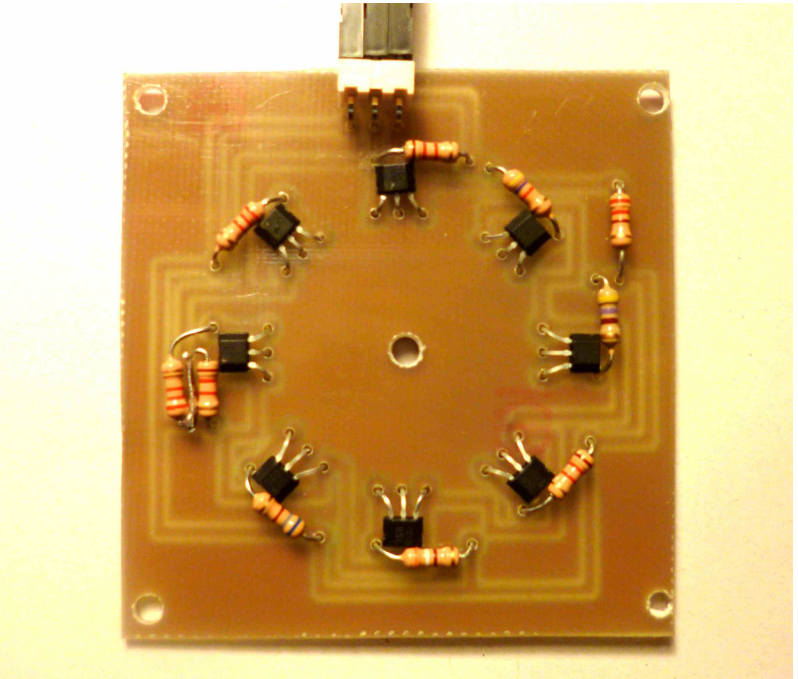
| Valeur | Sens | Angle |
|---|---|---|
| 144 | NNO | 337° |
| 150 | NNE | 22,5° |
| 156 | N | 0° |
| 174 | SSO | 202,5° |
| 200 | SSE | 157,5° |
| 223 | ESE | 112,5 |
| 250 | S | 180° |
| 267 | ENE | 67,5° |
| 287 | E | 90° |
| 327 | OSO | 247,5° |
| 369 | SO | 225° |
| 511 | SE | 135° |
| 539 | ONO | 292,5° |
| 662 | NO | 315 |
| 757 | O | 270° |
| 815 | NE | 45° |

| ohms | 220 | 4700 | 470 | 1200 | 390 | 670 | 3400 | 2200 | Valeur |
|---|---|---|---|---|---|---|---|---|---|
| 1200 | 0,78 | 4,04 | 1,42 | 2,53 | 1,24 | 1,83 | 3,75 | 3,28 | Tension |
| valeur pont diviseur | 156 | 815 | 287 | 511 | 250 | 369 | 757 | 662 | Valeur num |
| | N | NE | E | SE | S | SO | O | NO | |
| | 0,75 | | | | | | | | Tension |
| | | 1,33 | | | | | | | |
| | | | 1,11 | | | | | | |
| | | | | 1 | | | | | |
| | | | | | 0,86 | | | | |
| | | | | | | 1,62 | | | |
| | | | | | | | 2,67 | | |
| 0,72 | | | | | | | | 0,72 | |
| NNE | 150 | | | | | | | | Valeur Num |
| | ENE | 267 | | | | | | | |
| | | ESE | 223 | | | | | | |
| | | | SSE | 200 | | | | | |
| | | | | SSO | 174 | | | | |
| | | | | | OSO | 327 | | | |
| | | | | | | ONO | 539 | | |
| 144 | | | | | | | NNO | 144 | |

Exemple de réalisation du capteur en gravure à ''l'Anglaise''.



**Le virtuel ?**

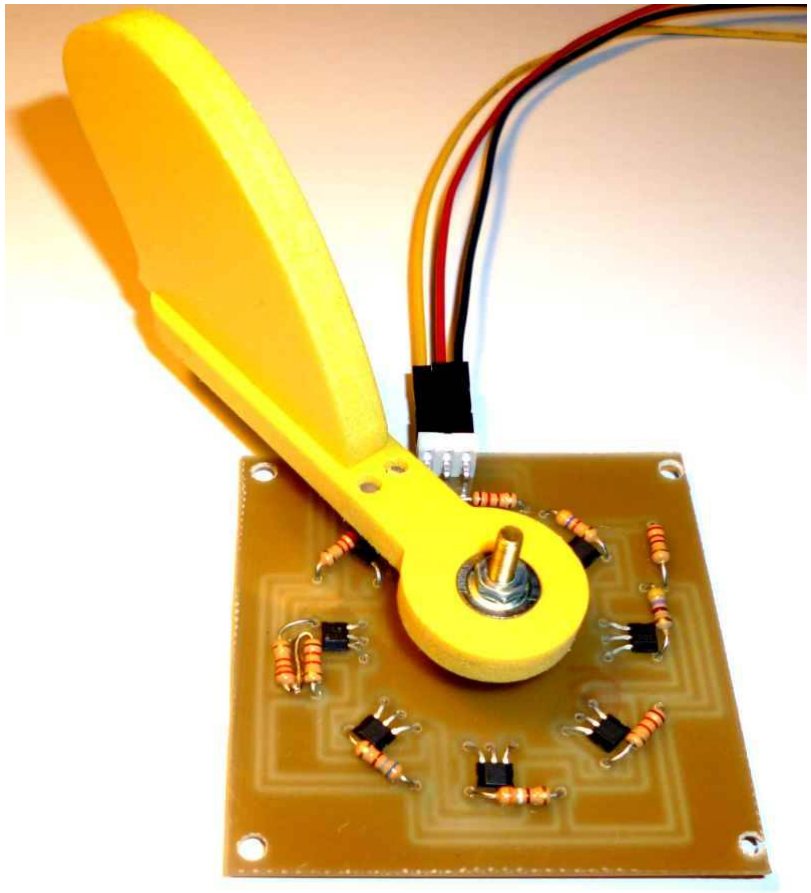


**La réalité !!! houps**

**Circuit réalisé avec l'excellent logiciel Target3001V16 discover, gratuit sans utilisation commerciale.**

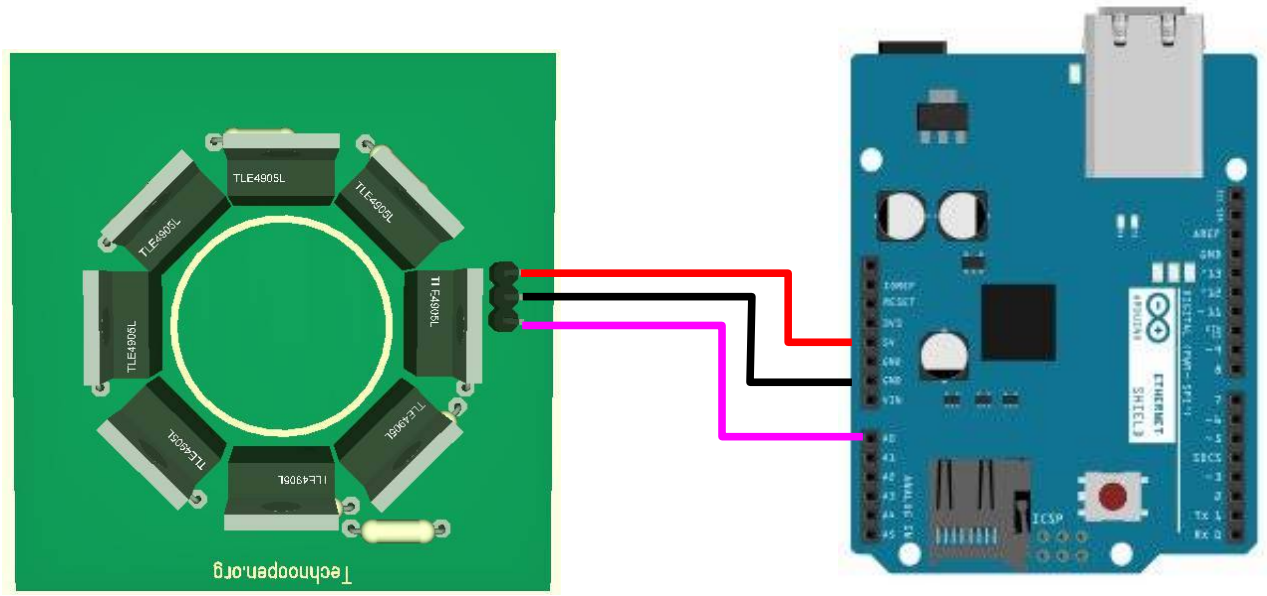
http://www.lextronic.fr/P4452-capteurs-pour-station-meteo.html

http://www.evola.fr/product_info.php/kit-capteurs-meteo-p-478



**Réalisation ''maison'' avec les capteurs à effet HALL**

**Capteur du commerce avec les contacts ILS (reed)**




**Le capteur à effet HALL est relié à une entrée analogique de l'Arduino ici la A0 et délivre 16 positions.**

# Montage final

# Programmation de l'Arduino pour lire la position du vent

direction_vent_bon | Arduino 1.0.5

Fichier Édition Croquis Outils Aide

direction_vent_bon §

```
if ((sensorValue>350) && (sensorValue<380)) {
Serial.println("Sud Ouest");
}
if ((sensorValue>740) && (sensorValue<780)) {
Serial.println(" Ouest");
}
if ((sensorValue>800) && (sensorValue<850)) {
Serial.println("Nord Est");
}
if ((sensorValue>167) && (sensorValue<=172)) {
Serial.println(" Nord");
}
if ((sensorValue>155) && (sensorValue<=160)) {
Serial.println(" Nord Nord Ouest");
}
if ((sensorValue>160) && (sensorValue<=165)) {
Serial.println(" Nord Nord Est");
}
if ((sensorValue>275) && (sensorValue<=280)) {
Serial.println(" Est Nord Est");
}
if ((sensorValue>230) && (sensorValue<=240)) {
Serial.println(" Est Sud Est");
}
if ((sensorValue>205) && (sensorValue<=212)) {
Serial.println(" Sud Sud Est");
}
if ((sensorValue>180) && (sensorValue<=185)) {
Serial.println(" Sud Sud Ouest");
}
if ((sensorValue>330) && (sensorValue<=340)) {
Serial.println(" Ouest Sud Ouest");
}
if ((sensorValue>540) && (sensorValue<=550)) {
Serial.println(" Ouest Nord Ouest");
}
}
```

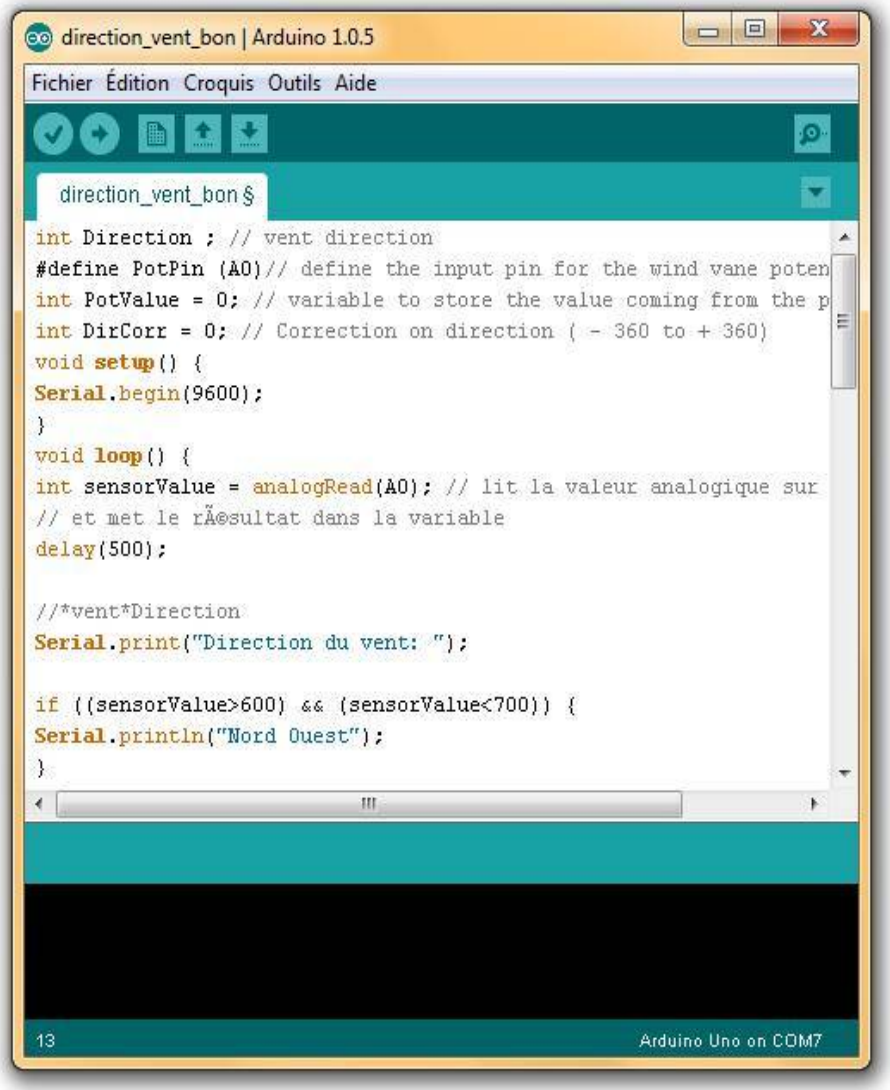
COM7

Envoyer

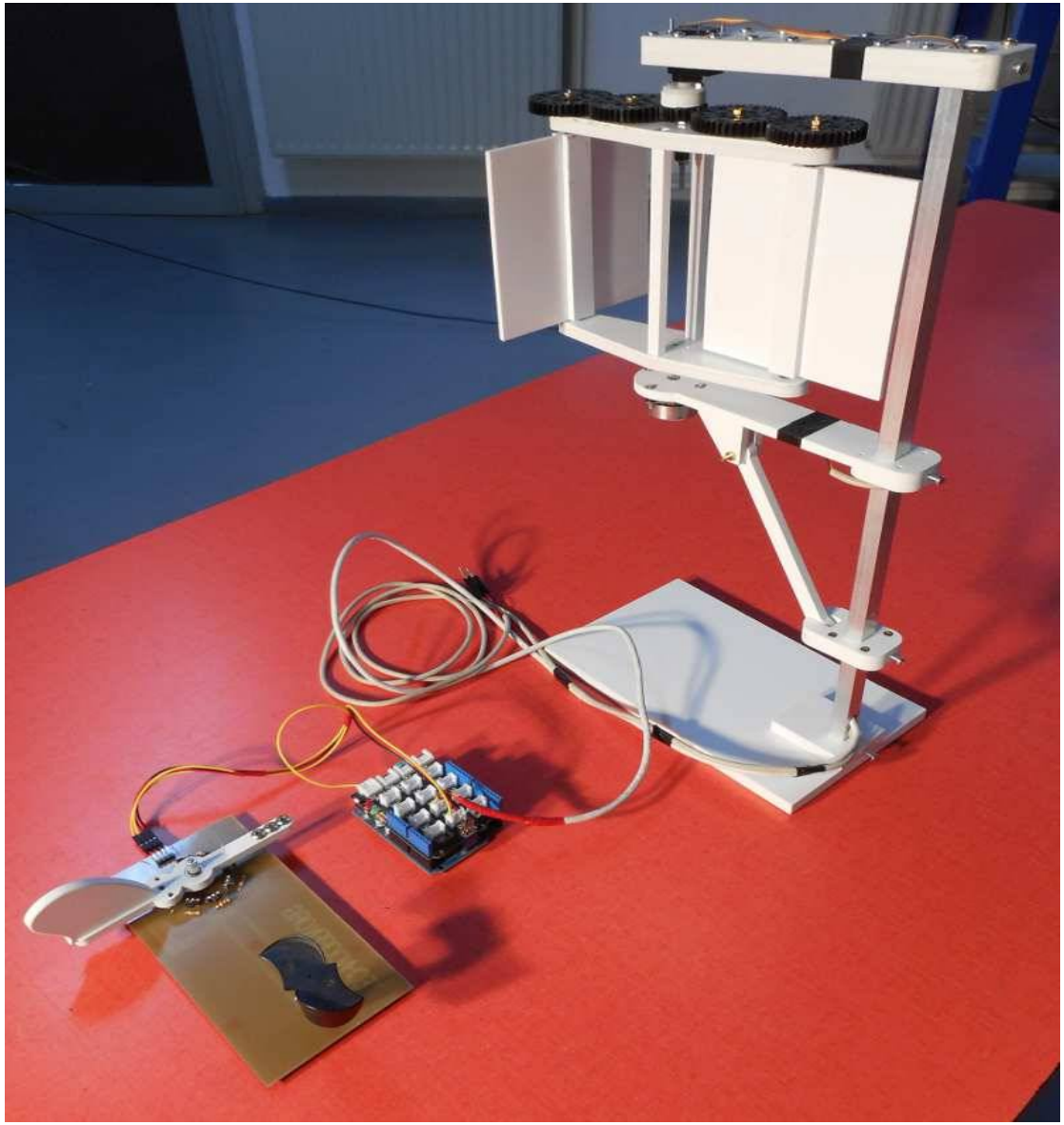
```
Direction du vent: Sud Est
Direction du vent:  Sud Sud Est
Direction du vent:  Sud Sud Est
Direction du vent:  Sud
Direction du vent:  Sud
Direction du vent:  Sud
Direction du vent:  Sud Sud Ouest
Direction du vent:  Sud Sud Ouest
Direction du vent: Sud Ouest
Direction du vent: Sud Ouest
Direction du vent: Sud Ouest
Direction du vent:  Ouest Sud Ouest
Direction du vent:  Ouest
Direction du vent:  Ouest
Direction du vent:  Ouest Nord Ouest
Direction du vent:  Ouest Nord Ouest
Direction du vent: Nord Ouest
Direction du vent:  Nord Nord Ouest
Direction du vent:  Nord Nord Ouest
Direction du vent:  Nord Nord Ouest
Direction du vent:  Nord Nord Ouest
Direction du vent:  Nord
Direction du vent:  Nord
Direction du vent:  Nord
Direction du vent:  Nord
Direction du vent:  Nord
Direction du vent:  Nord
Direction du vent:  Nord
Direction du vent:  Nord
```

Défilement automatique | NL & CR | 9600 baud

Il y a bien-sûr énormément de manières de programmer l'Arduino pour des résultats presque similaires. Le programme ci -dessus se borne à afficher sur le terminal la position. Il compare l'entrée numérique à des valeurs pré-enregistrées.




Exemple : ''Si la valeur est comprise entre 600 et 700 alors il s'affiche Nord Ouest''



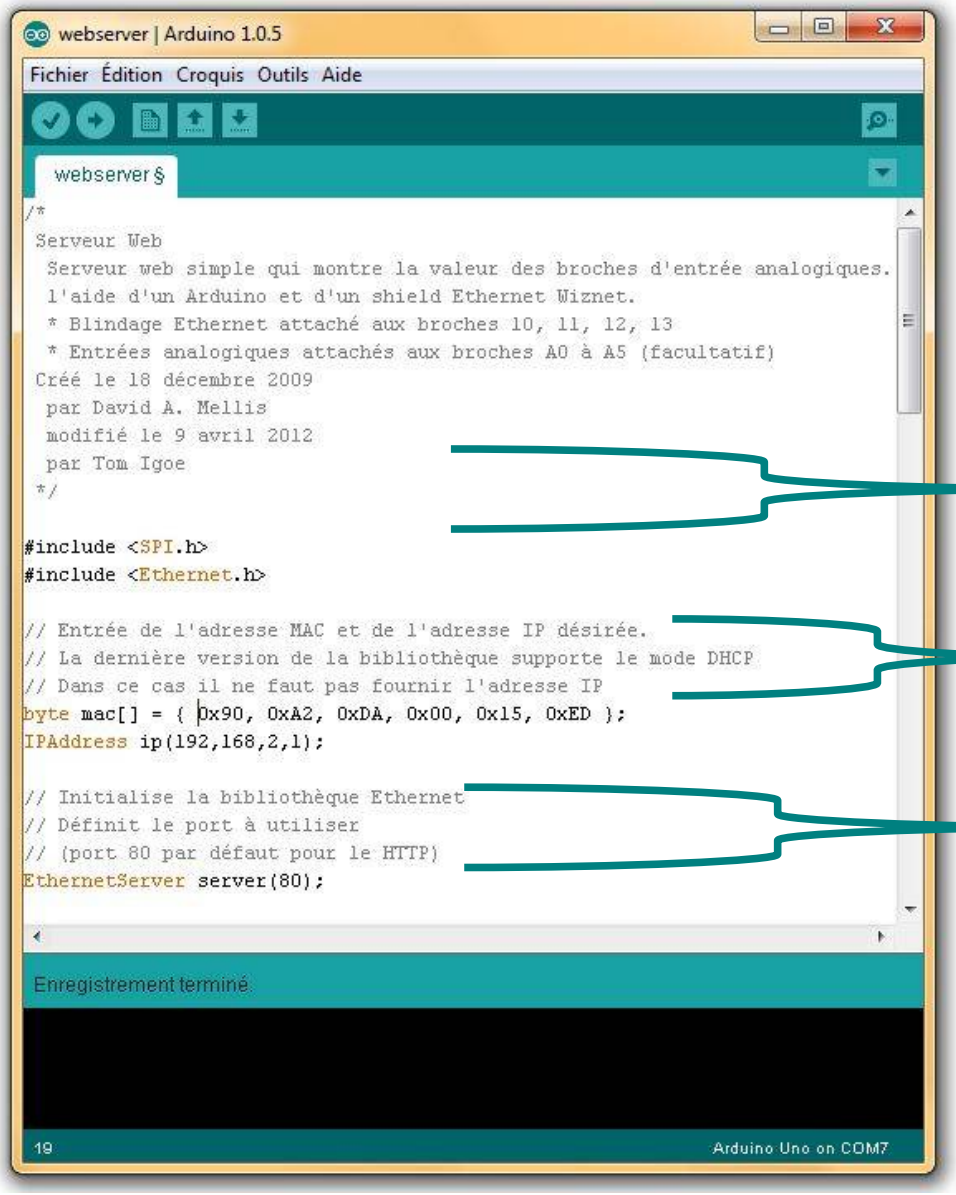
**Le capteur à effet HALL est relié à une entrée analogique de l'Arduino ici la A0 et délivre 16 positions. Exemple de fonctionnement avec la girouette qui informe de la position du vent et la position des pales commandé par le servo-moteur.**

Vidéo sur :

http://www.dailymotion.com/video/x27e1cw_regulation-eolienne-axe-vertical_tech

# Analyse d'un exemple : schield Ethernet

Nous allons maintenant ''décortiquer'' plusieurs exemples afin de comprendre comment afficher des informations en provenance de l'automate et comment le commander. L'exemple suivant a été écrit par David A. Mellis et Tom Igoe concepteurs de l'Arduino.




Commentaire sur le programme. Toute cette partie ne sera pas compilée par le logiciel.

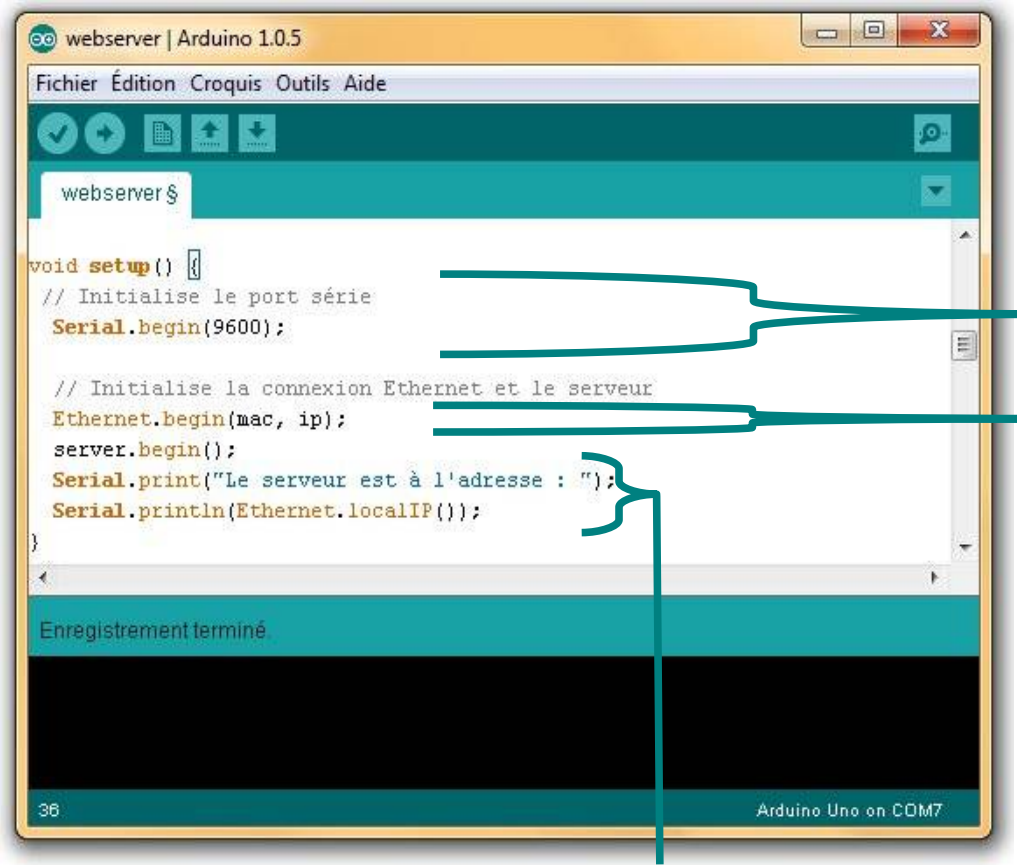
Partie déclarative : inclusion dans le programme de la bibliothèque pour le Bus de donnée SPI ainsi que la bibliothèque Ethernet qui gère la puce Wiznet.

Déclaration d'une variable et d'un tableau comportant l'adresse MAC du Shield Ethernet.
Déclaration d'une variable contenant l'adresse IP que vous voulez donner au Shield Ethernet.

Initialisation de la bibliothèque Ethernet et définition du port à utiliser
(port 80 par défaut pour le HTTP) et la visualisation sur le navigateur

Le programme de cette exemple dans sa version francisée est disponible dans la boîte à outils.
Ce programme est inspiré de la traduction de M Tavernier
http://www.tavernier-c.com

Boucle d'initialisation :
Setup (installation)




Initialisation et démarrage du port série

Configuration du fonctionnement du Shield pour la gestion de l'adresse IP.
- Si « ip » est indiqué le Shield va prendre en compte l'adresse IP fixe indiquée par vos soins plus haut.
- Si « ip » est absent de la ligne la carte va permettre au routeur de fournir une IP en DHCP.

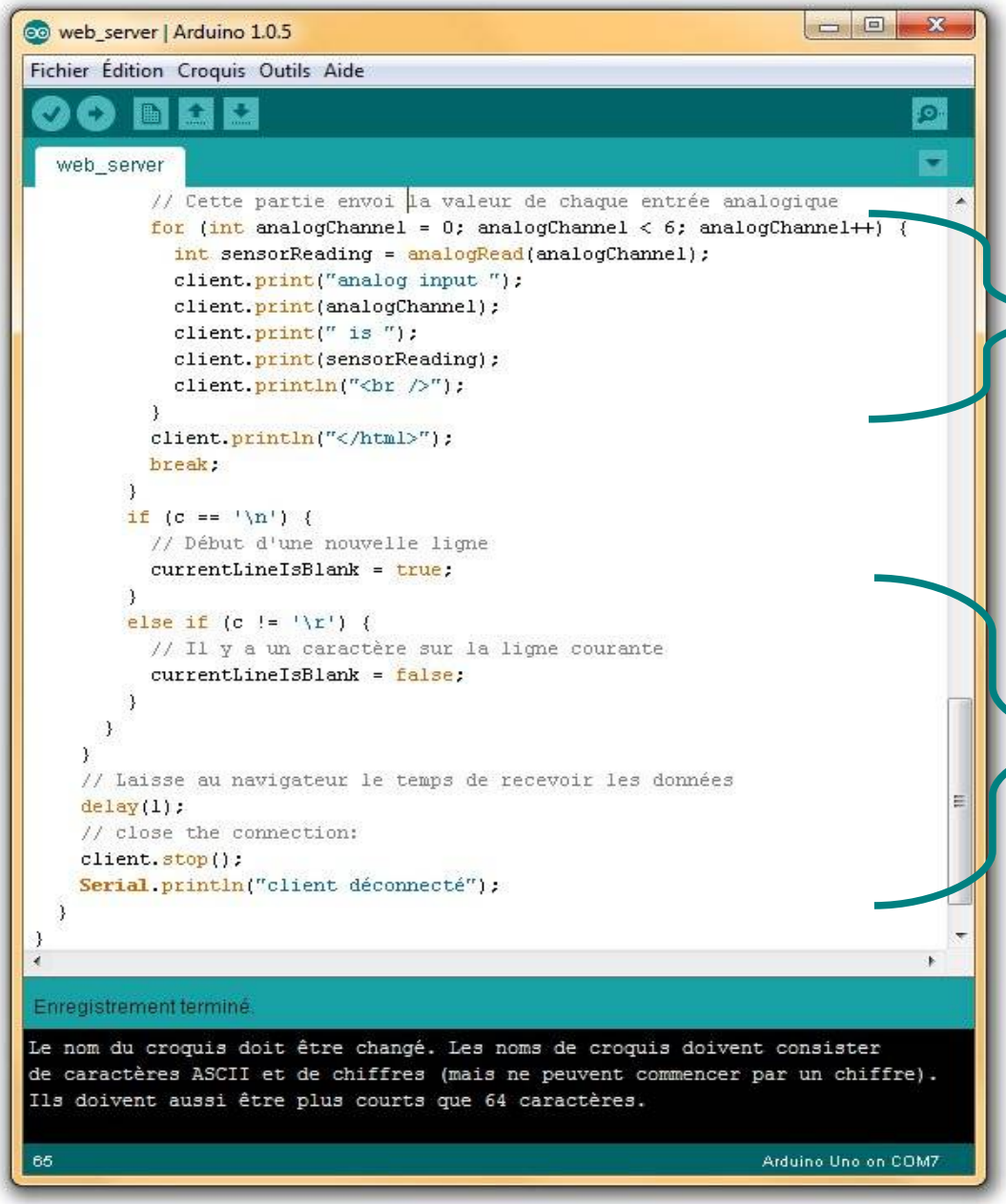
Cette instruction démarre le serveur et elle demande à l'Arduino d'afficher l'adresse IP sur le terminal. Cela va devenir primordial lorsque l'on a choisi le mode DHCP. Dans le cas contraire nous n'avons pas la possibilité de connaître l'adresse allouée par le routeur et le réseau local.


Début de la boucle principale. Cette ligne initialise en quelque sorte la liaison entre le client (le navigateur) et le serveur (Arduino et le shield)

Si le client est présent il y a écrit dans la liaison série ''nouveau client ''

Initialisation de l'échange

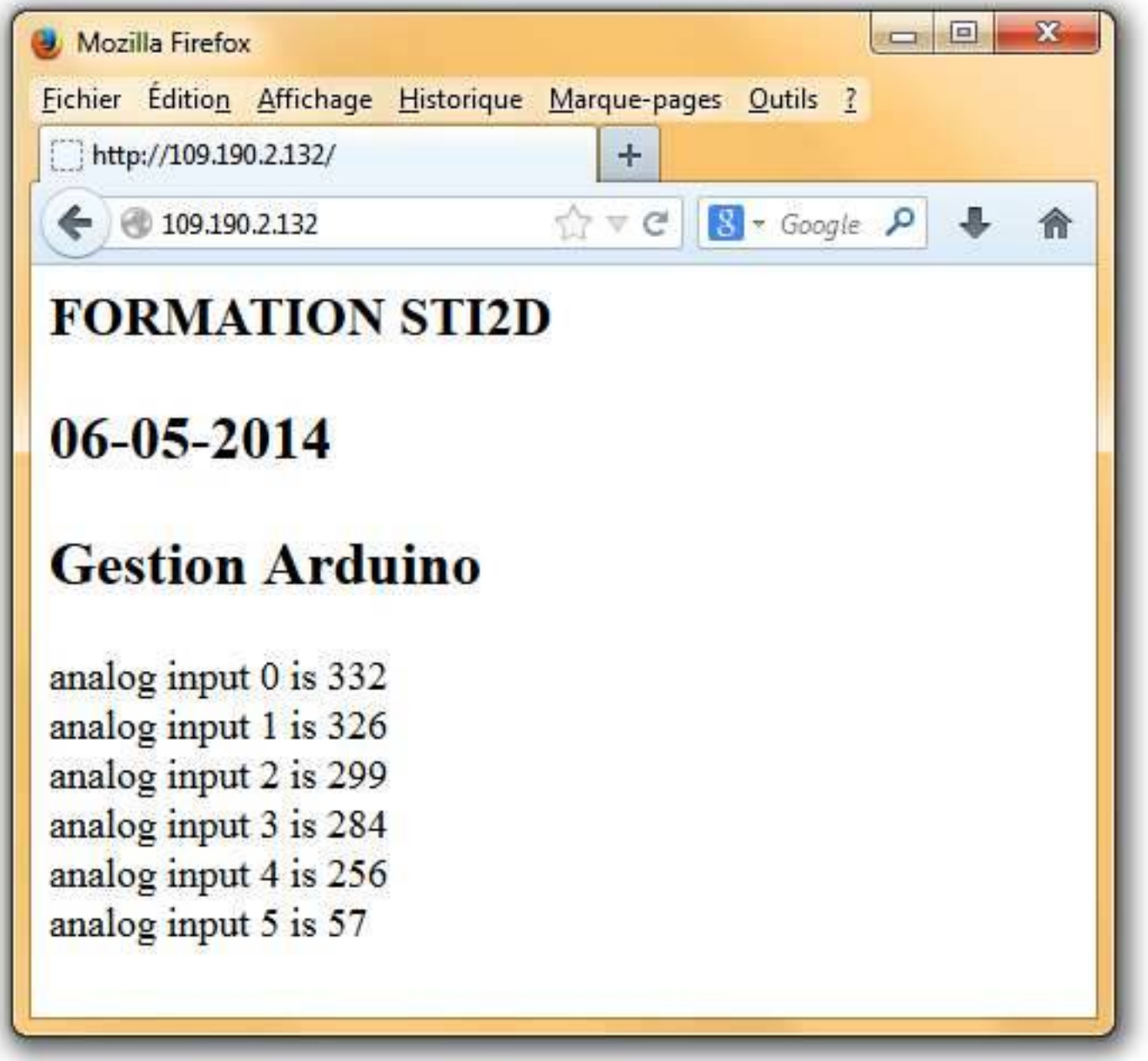
Si tout est ok, l'Arduino renvoie un en-tête http standard



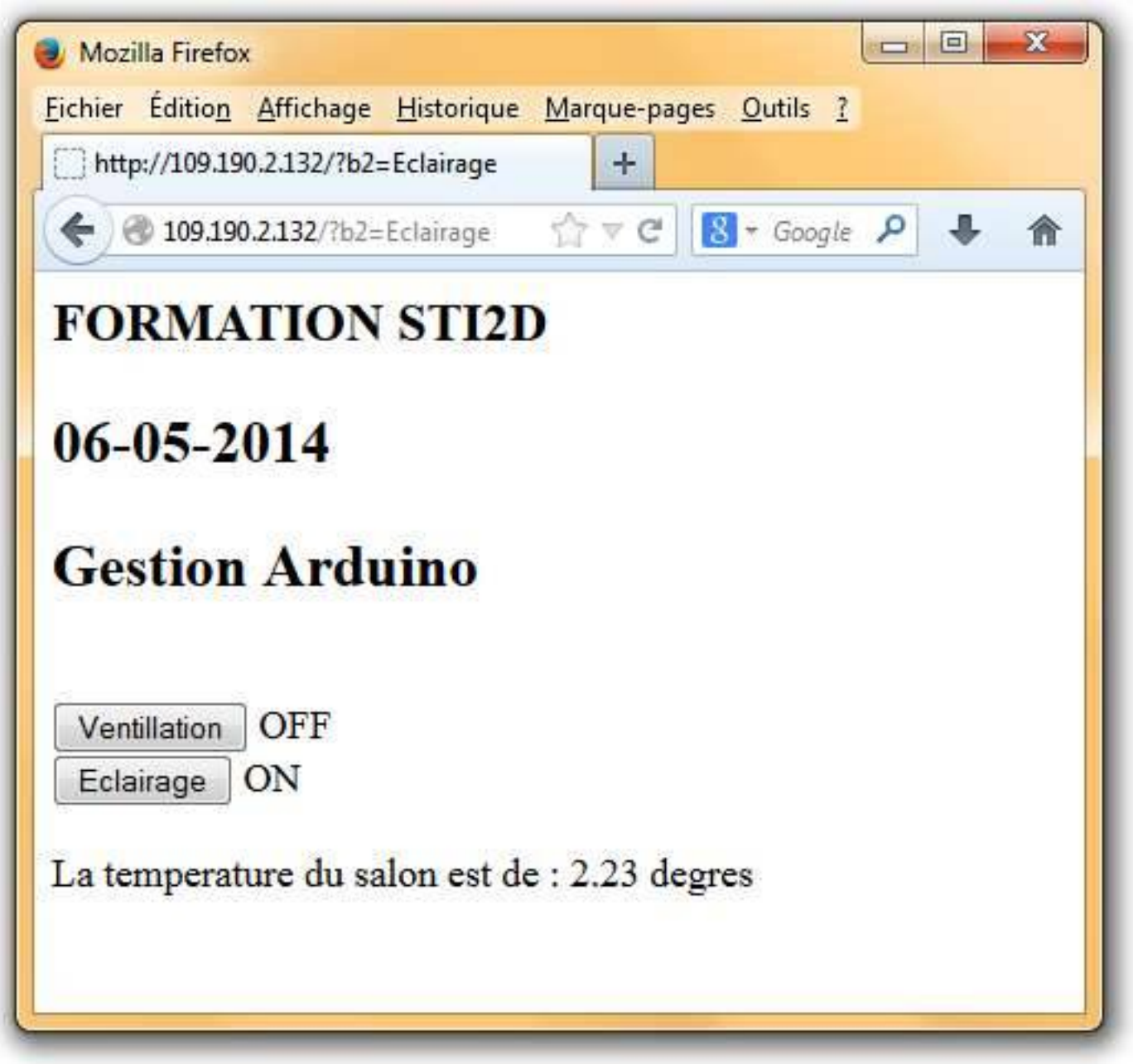

Cette partie envoie la valeur de chaque entrée analogique

Fermeture du programme

Le résultat

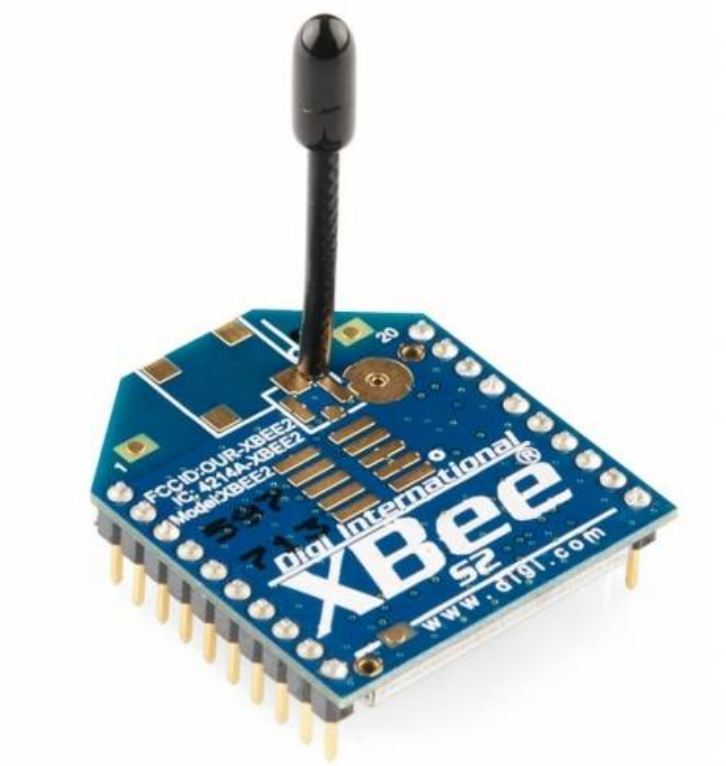

Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils ?
http://109.190.2.132/
109.190.2.132
Google
FORMATION STI2D
06-05-2014
Gestion Arduino
analog input 0 is 332
analog input 1 is 326
analog input 2 is 299
analog input 3 is 284
analog input 4 is 256
analog input 5 is 57

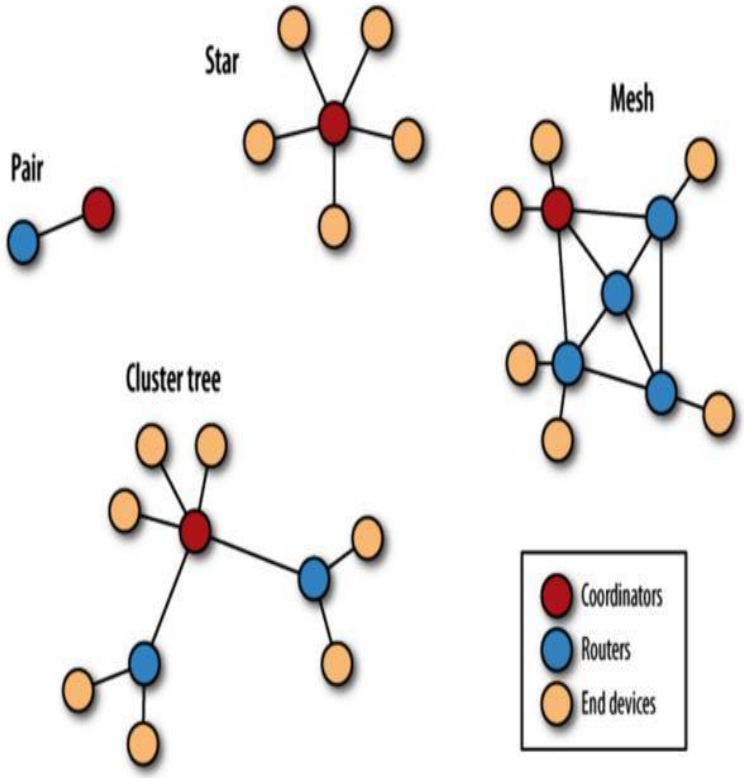
Un autre résultat avec la commande Get



# Intéressons nous maintenant à la liaison Xbee

# Transmission radio
# ''Module Xbee''




Le nom provient sans doute de l'image qu'il peut y avoir de plusieurs petits modules connectés ensemble comme une colonie d'abeilles. **ZigBee** est un protocole de communication qui s'appuie sur le travail du groupe IEEE 802.15.4 et définit par le groupe de professionnels ZigBee Alliance. **XBee** est une marque, un produit (Digi international) qui utilise le protocole ZigBee.

## Présentation

Principales caractéristiques du XBee :

- fréquence porteuse : 2.4Ghz
- portées variées : assez faible pour les XBee 1 et 2 (10 - 100m), grande pour le XBee Pro (1000m)
- faible débit : 250kbps
- faible consommation : 3.3V, 50mA
- entrées/sorties : 6 10-bit ADC input pins, 8 digital IO pins
- sécurité : communication fiable avec une clé de chiffrement de 128-bits
- faible coût : ~ 25€
- simplicité d'utilisation : communication via le port série
- ensemble de commandes AT et API
- flexibilité du réseau : sa capacité à faire face à un nœud hors service ou à intégrer de nouveaux nœuds rapidement
- grand nombre de nœuds dans le réseau : 65000
- topologies de réseaux variées : maillé, point à point, point à multi-points

## Modèles

Plusieurs produits XBee existent. Il y a deux catégories de XBee :
**-la série 1,** les modules de la série 1 ont souvent un "802.15.4" qui s'ajoutent à leurs noms.
**-la série 2,** Les modules de la série 2 sont disponibles en plusieurs versions : **XBee ZNet 2.5** (obsolète), le **ZB** (l'actuel) et le **2B** (le plus récent). Vous avez aussi des **XBee Pro**, qui font la même chose, mais avec de plus grandes capacités, notamment la portée qui semble pouvoir aller jusqu'à 1000 mètres ! Pour en savoir plus, télécharger le tableau de comparaisons des modules XBee :

http://www.digi.com/pdf/chart_xbee_rf_features.pdf.

Caractéristiques des modules série 1 et série 2

| | Series 1 | Series 2 |
|---|---|---|
| Typical (indoor/urban) range | 30 meters | 40 meters |
| Best (line of sight) range | 100 meters | 120 meters |
| Transmit/Receive current | 45/50 mA | 40/40 mA |
| Firmware (typical) | 802.15.4 point-to-point | ZB ZigBee mesh |
| Digital input/output pins | 8 (plus 1 input-only) | 11 |
| Analog input pins | 7 | 4 |
| Analog (PWM) output pins | 2 | None |
| Low power, low bandwidth, low cost, addressable, standardized, small, popular | Yes | Yes |
| Interoperable mesh routing, ad hoc network creation, self-healing networks | No | Yes |
| Point-to-point, star topologies | Yes | Yes |
| Mesh, cluster tree topologies | No | Yes |
| Single firmware for all modes | Yes | No |
| Requires coordinator node | No | Yes |
| Point-to-point configuration | Simple | More involved |
| Standards-based networking | Yes | Yes |
| Standards-based applications | No | Yes |
| Underlying chipset | Freescale | Ember |
| Firmware available | 802.15.4 (IEEE standard), DigiMesh (proprietary) | ZB(ZigBee 2007), ZNet 2.5 (obsolete) |
| Up-to-date and actively supported | Yes | Yes |

- les modules des séries 1 et 2 ne sont pas compatibles entre eux,

- la portée et la consommation sont sensiblement les mêmes,

- le nombre d'entrées et sorties est différent. La série 2 ne possède pas de sorties analogiques PWM.

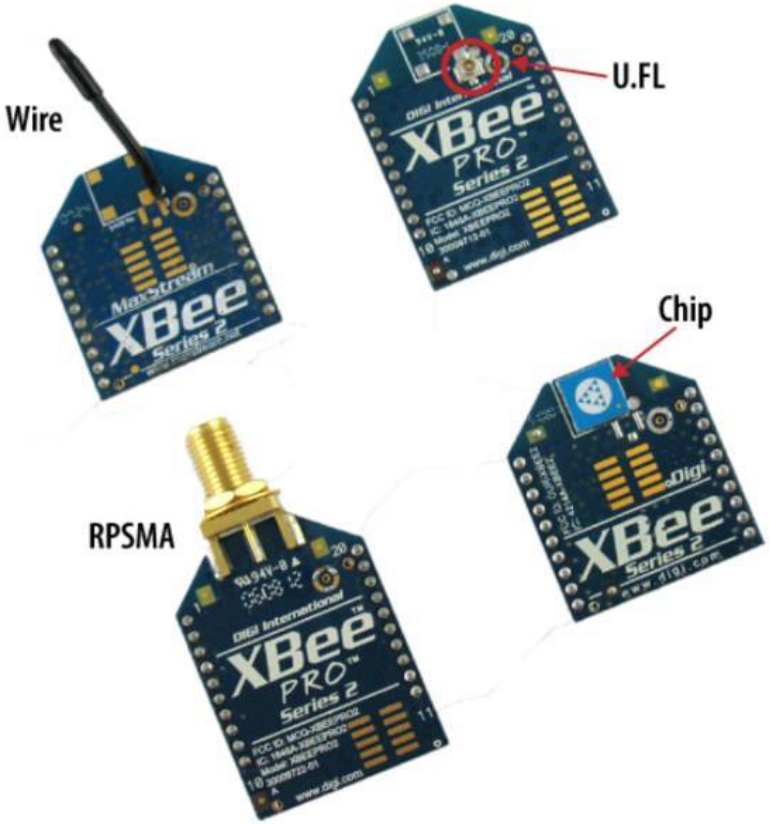
Les modules Xbee ne sont pas que de simples modems, ils sont constitués à partir d'un véritable micro-controleur qui permet directement de traiter des informations. Selon le modèle (indiqué dans le tableau ci-dessous) ils possèdent plusieurs entrées et sorties numériques comme analogiques.


Série 1


Série 2

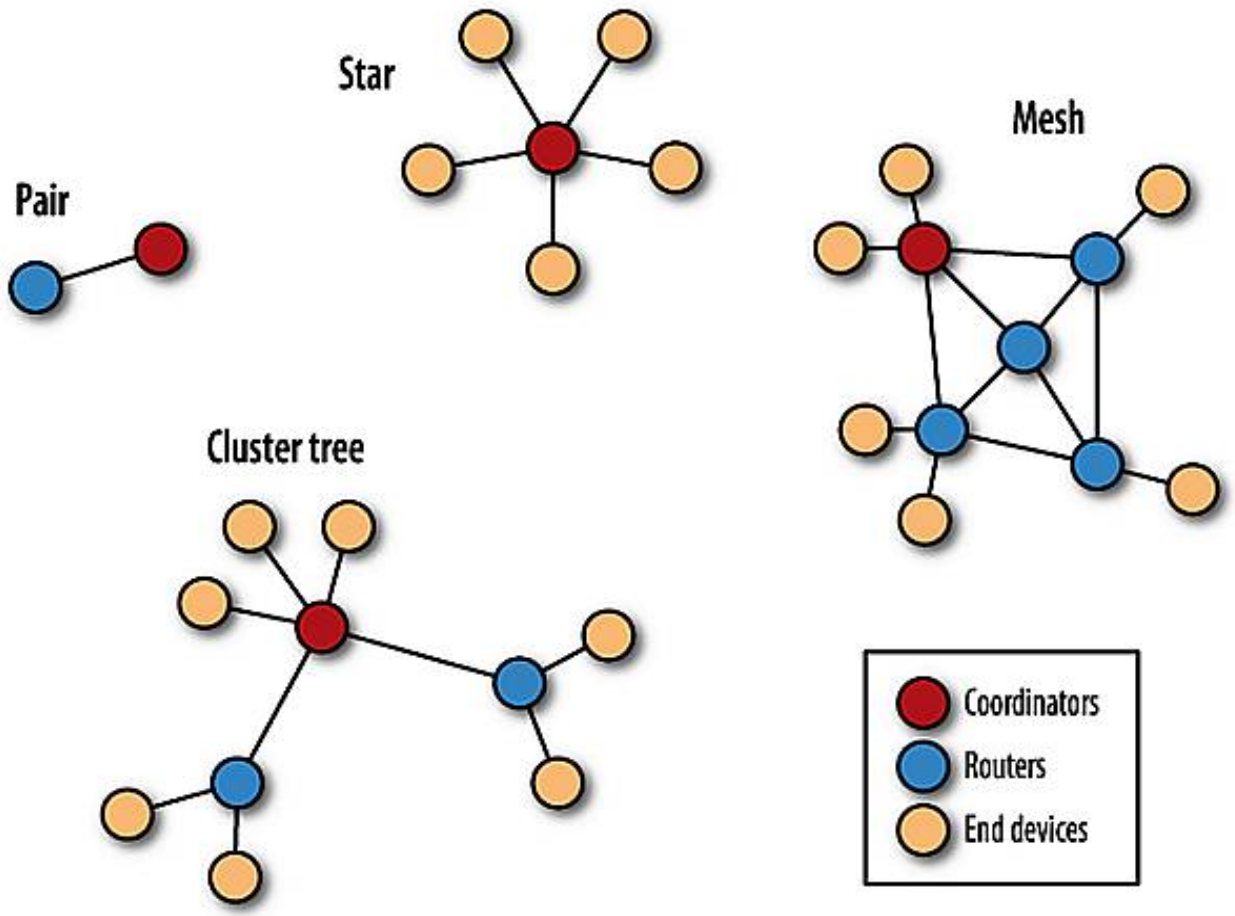
## Types d'antennes disponibles

- wire : simple, radiations omnidirectionnelles ;
- chip : puce plate en céramique, petite, transportable (pas de risques de casser l'antenne), radiations cardioïdes (le signal est atténué dans certaines directions) ;
- U.FL : une antenne externe n'est pas toujours nécessaire;
- RPSMA : plus gros que le connecteur U.FL, permet de placer son antenne à l'extérieur d'un boîtier.



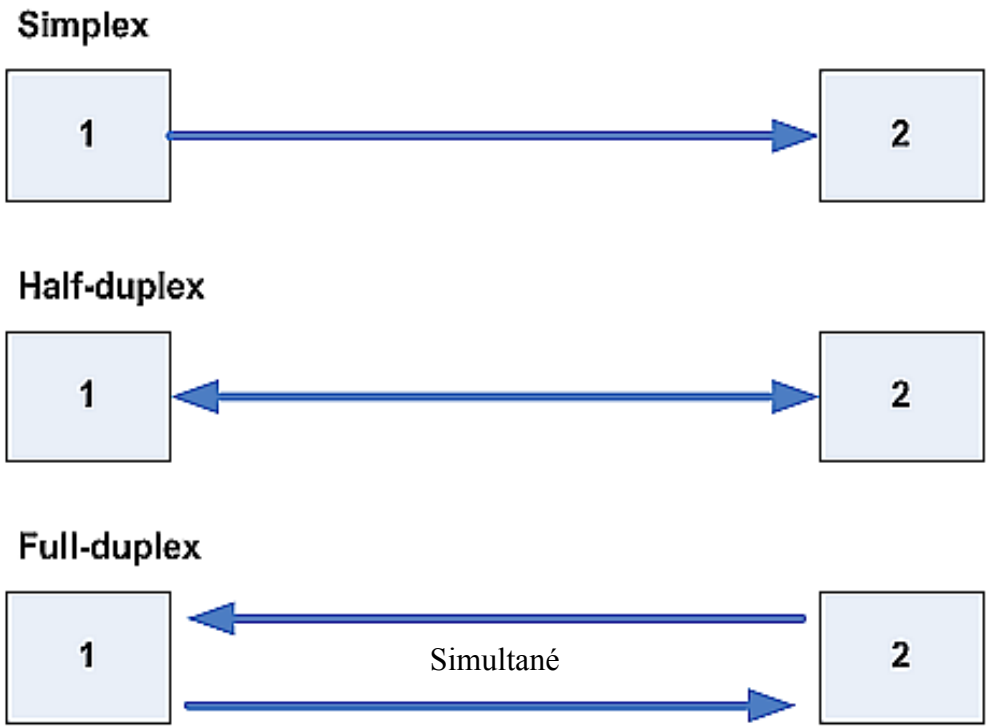
Types de réseaux possibles avec les modules Xbee

Les modules Xbee peuvent travailler sur plusieurs types de réseaux. Ils seront le plus souvent utilisés par paires dans les applications avec Arduino.



## Types de liaisons

Les modules XBee permettent de recevoir et d'émettre des données en même temps, on dit qu'ils sont full duplex, contrairement à la radio FM qui envoie les informations dans un seul sens (simplex) et au talkie-walkie qui ne permet pas à deux émetteurs de parler en même temps (half-simplex). On dit aussi que le XBee est un transceiver qui est la contraction de TRANSmitter (émetteur) et de reCEIVER (récepteur).

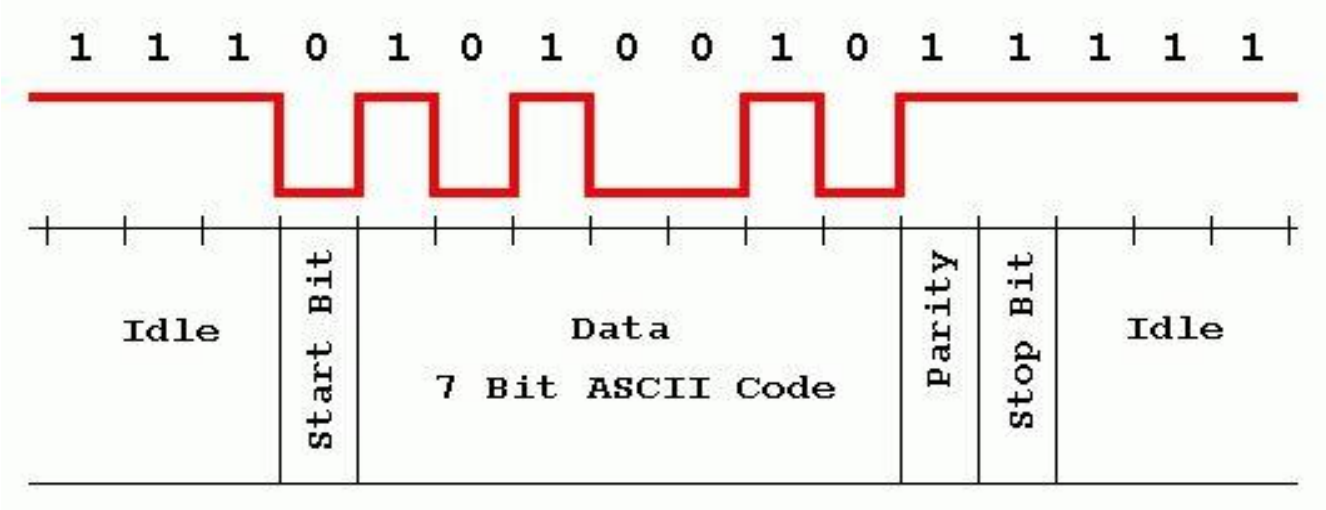



Liaison série

Dans une liaison en série, les données sont envoyées bit par bit sur la voie de transmission. Toutefois, étant donné que la plupart des processeurs traitent les informations de façon parallèle, il s'agit de transformer des données arrivant de façon parallèle en données en série au niveau de l'émetteur, et inversement au niveau du récepteur. Ces opérations sont

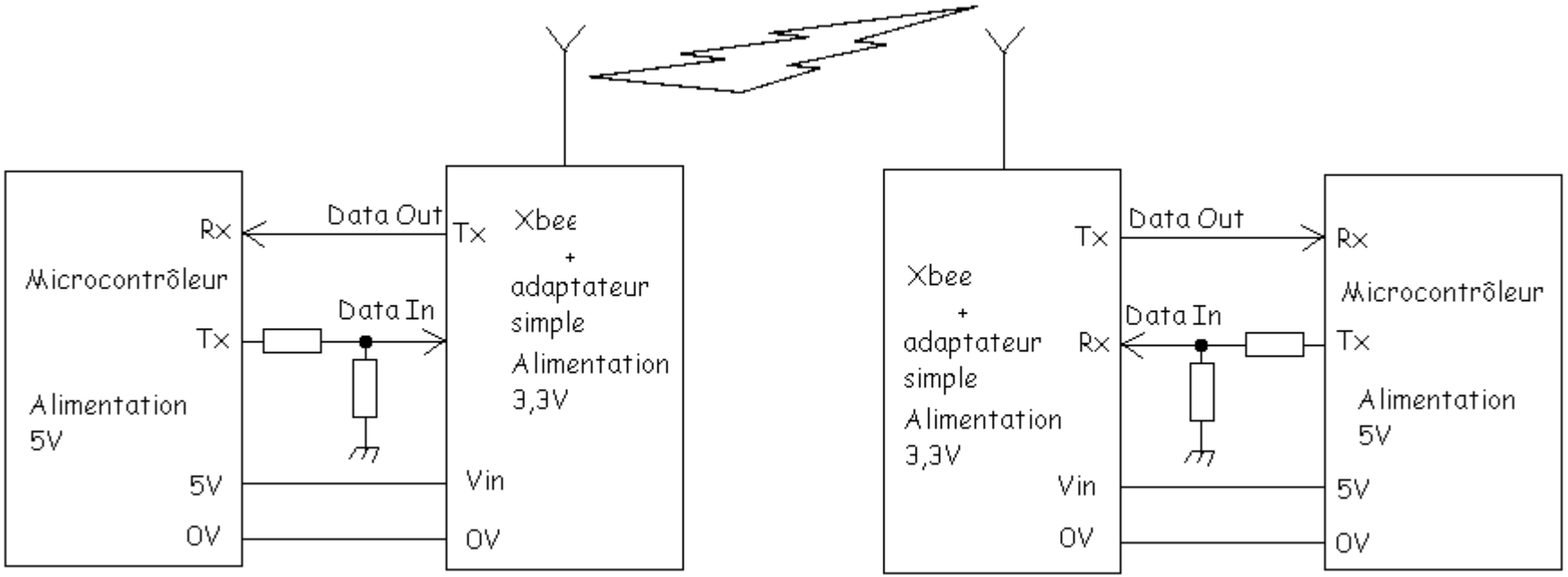
réalisées grâce à un contrôleur de communication (la plupart du temps une puce UART, Universal Asynchronous Receiver Transmitter). Il fonctionne grâce à un registre à décalage. Le registre de décalage permet, grâce à une horloge, de décaler le registre (l'ensemble des données présentes en parallèle) d'une position à gauche, puis d'émettre le bit de poids fort (celui le plus à gauche) et ainsi de suite."

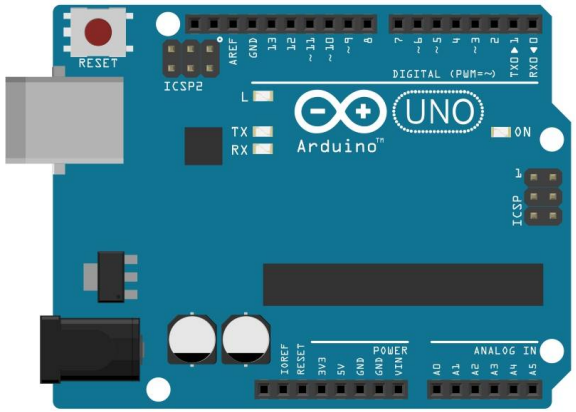
(source : http://www.commentcamarche.net/contents/transmission/transnum.php3 )

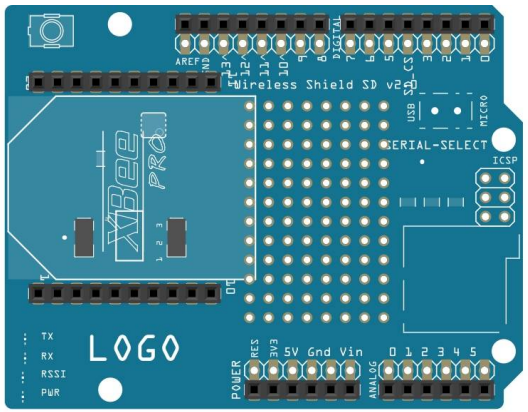


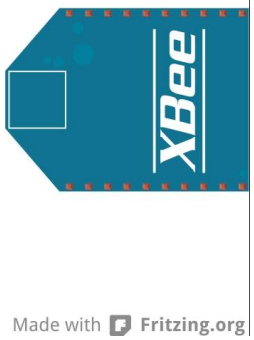

"Toute l'astuce d'une liaison série asynchrone repose sur la forme des signaux envoyés ; signaux qui permettent une synchronisation du récepteur sur chaque caractère reçu. Examinez la figure ci-dessus qui représente la transmission asynchrone de l'octet 01010010 (caractère ASCII "R", valeur décimale 82). Au repos (idle) la ligne de transmission est à l'état logique haut. La transmission débute par le passage à 0 de cette ligne pendant une période de l'horloge de transmission ce qui constitue le bit de start (ce qui signifie début ou départ). Les bits du mot à transmettre sont ensuite envoyés derrière ce bit de start comme dans une transmission série synchrone et, après le dernier bit utile, la ligne passe à nouveau à l'état haut pendant une ou deux périodes d'horloge pour constituer ce que l'on appelle le ou les bits de stop." (source : http://www.tavernier-c.com/serie.htm)

# Structure d'une liaison




X2

# Exemple de programme Xbee à Xbee

Exemple 1 : Structure d'un programme

Commande d'une sortie

**Code émetteur :**

```
/*
Lit une entrée numérique sur la broche 2, affiche le résultat sur le port série
*/
int boutonPoussoir = 2; // crée la variable boutonPoussoir ''bouton poussoir'' et l'associe à la broche N°2 de l'Arduino

void setup() { //lance l'initialisation
  Serial.begin(9600); // démarrage de la liaison série à 9600 bauds '' vitesse de base du mode transparent des moules Xbee''
  pinMode(boutonPoussoir, INPUT); // configure la variable boutonPoussoir en entée ''donc la broche N°2''
        }

void loop() { // boucle principale
 // read the input pin:
 int etatPoussoir = digitalRead(boutonPoussoir); // lit la broche d'entrée ''boutonPoussoir'' et crée une variable etatPoussoir ''état du bouton''
 // print out the state of the button:
        if (etatPoussoir == HIGH){ // Si l'état du bouton est à l'état haut
```

```
            Serial.println(etatPoussoir, DEC); //affiche sur le port série l'état de la variable ''état du bouton''
                                }
            if (etatPoussoir == LOW){ // Si l'état du bouton est à l'état bas
            Serial.println(etatPoussoir, DEC); //affiche sur le port série l'état de la variable ''état du bouton''
            Serial.flush (); // permet de vider le tampon du port série ''pas toujours nécessaire''
          }
  delay(1); //Retard entre deux lectures pour la stabilité
          }
```

**Code récepteur :**

```
/*
reçoit une commande du Xbee émetteur sur le port série et commande une LED en sortie
*/
int message = 0; // crée la variable message de type Interger et la positionne à 0
int brocheLed = 8; // crée la variable brocheLed et l'associe à la broche N°8 de l'Arduino.

void setup() { //lance l'initialisation
Serial.begin(9600); // démarrage de la liaison série à 9600 bauds '' vitesse de base du mode transparent des moules Xbee''
  pinMode(brocheLed, OUTPUT); // configure la variable brocheLed en sortie
}
```

```
void loop() { //démarrage de la boucle principale
    if (message != 0){ // si message est différent de 0
      //char inChar = (char)message;
    //Serial.println(inChar);
    Serial.println(message); //écrire sur le port série message
    }
    if (message == '1'){
      digitalWrite(brocheLed, HIGH); // si la variable message est parfaitement = à 1 activer la broche N°8
//brocheLed//
    }
    if (message == '0'){  // si la variable message est parfaitement = à 0 désactiver la broche N°8 //brocheLed//
      digitalWrite(brocheLed, LOW);
    }
}
void serialEvent() {
/*
SerialEvent se produit chaque fois que des nouvelles données proviennent de la liaison RX matériel. Cette routine est exécutée entre chaque boucle de temps (), donc elle peut retarder la réponse. Plusieurs octets de données peuvent être disponibles.
*/
```

```
  while (Serial.available()) { // cette fonction permet de savoir combien de caractères ont été mémorisés dans le tampon, 128 maximum.
    message = Serial.read(); // lit le premier caractère contenu dans le tampon
    Serial.flush (); // permet de vider le tampon du port série ''pas toujours nécessaire''
  }
}
```

Exemple 2 : Structure d'un programme

Variation de la luminosité d'une led commandée par un potentiomètre.

```
#define brochePot 0 // Création de la variable brochePot, qui définit la broche analogique 0
#define brocheLed 3
int tamponSerie = -1;
char tableauChaine[6];
int valChainePos = 0;
int ancieneVal = 0;
void setup()  {
                pinMode(brocheLed, OUTPUT);
                Serial.begin(9600);
              }
void loop() {
                tamponSerie = Serial.read();
/*
Envoie une nouvelle valeur seulement si elle diffère de la précédente d'au moins 5 afin de ne pas saturer le réseau
*/
                int valeurPot = analogRead(brochePot);
                if( abs(valeurPot - ancieneVal) > 5) {Serial.println(valeurPot);
```

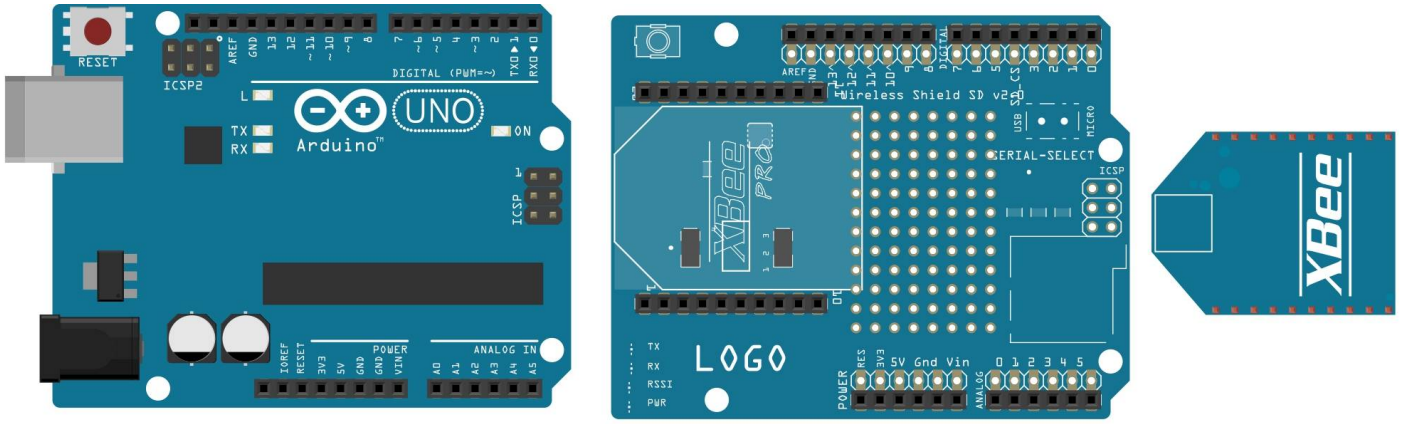
```
    ancieneVal = valeurPot;
  }
if((tamponSerie >= '0') && (tamponSerie <= '9')){tableauChaine[valChainePos] = tamponSerie;
valChainePos ++; // Si une donnée numérique est disponible, la mémoriser
 }
/*
Si un caractère de fin de ligne est détecté, traiter la chaîne et écrire la valeur sur la sortie qui commande la LED puis remettre la chaîne à zéro
*/
if(tamponSerie == '\r'){
int valLum = atoi(tableauChaine); //convertit la chaîne en entier

valLum = map(valLum, 0, 1023, 0, 255); // Réduction de la plage d'entrée de 0-1023 à 0-255
analogWrite(brocheLed, valLum);

for (int c = 0; c < valChainePos; c++){ tableauChaine[c] = 0; // Remise à zéro de la chaîne de caractères
}
    valChainePos = 0;
  }
}
```

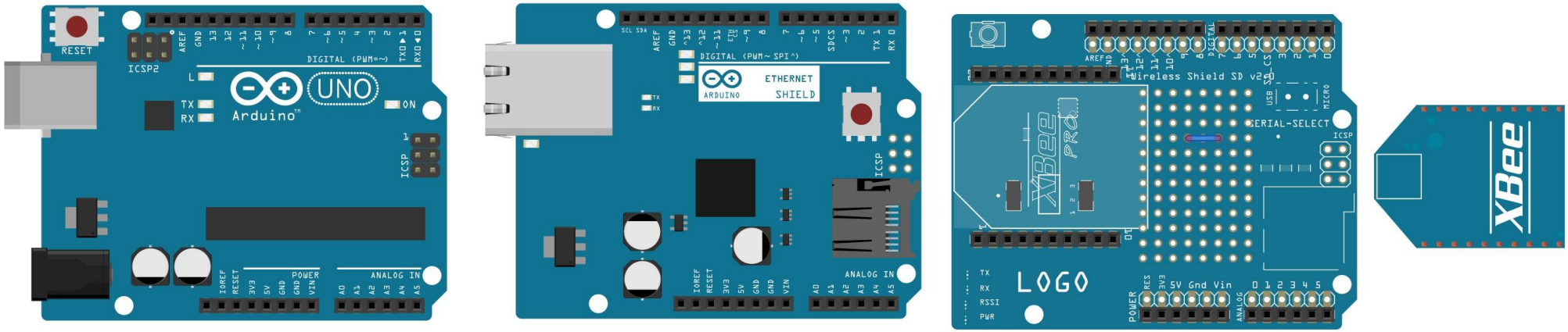
# Exemple de programme complet

Le but maintenant est de récupérer avec un premier Arduino :

- la vitesse du vent (Broche N°2 en entrée)
- la positon de la girouette (Broche analogique N°A0 en entrée)
- vitesse de l'alternateur ( Broche N°4 en entrée)

La broche N°3 et utilisée en sortie pour la commande de la girouette.



Exemple programme Complet

Communication Xbee vers Xbee (mode transparent) et récupération info Xbee vers Ethernet Webwerver

```
  /*
  Exemple Web Server & Xbee
  Une entrée analogique (A0) sur Arduino ethernet
  Une entrée numérique (N°4) sur Arduino Xbee-ethernet récepteur
  Une entrée numérique (N°2) sur Arduino Xbee émetteur
  Utilisant l'Arduino Wiznet Ethernet shield.
  Le shield Ethernet  est attaché aux broches 10, 11, 12, 13
  Le premier code à été crée le 18 Dec 2009
  par David A. Mellis modifié le 9 Apr 2012 par Tom Igoe
  Modifié et adapté par Johan Rungette le 25 05 2014
  */

// partie déclarative
#include <SPI.h>
#include <Ethernet.h>
// associe la broche à l'interrupteur
const int brocheInter = 4;
// associe la broche 10 à l'activation du shield ethernet (0=activé, 1=pause)
const int pauseEthernet = 10;
// associe la broche 8 à la variable brocheLed
const int brocheLed =  8;
// Force état de la variable etatInter à 0
```

```
int etatInter = 0;
// Force état de la variable etatInter à 0
int message = 0;

/*
Entrée de l'adresse MAC et de l'adresse IP désirée.
La dernière version de la bibliothèque supporte le DHCP.
Dans ce cas il ne faut pas fournir l'adresse IP fixe.
*/

byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xF7, 0xB2 }; // A adapter en fonction de l'adresse mac de votre shield
byte ip[] = { 192, 168, 2, 1 }; // A adapter en fonction de votre réseau (donné par la ''box'')
// initialise la bibliothèque Ethernet
// définit le port à utiliser
// (port 80 par défaut pour le HTTP)
EthernetServer server(80);

void setup() {
    //met la broche de la Led en sortie
  pinMode(brocheLed, OUTPUT);

  // met la broche de l'inter en entrée
  pinMode(brocheInter, INPUT);

 // initialise le port série
  Serial.begin(9600);
```

```
  // initialise la connexion Ethernet et le serveur
  Ethernet.begin(mac); // si vous rajoutez ip (mac, ip) vous forcez l'adresse à celle désignée plus haut
  server.begin(); // démarrage du serveur ethernet
  Serial.print("Le serveur est à l'adresse : "); // avec la ligne suivante cela permet de récupérer l'adresse IP donnée par
le serveur.
  Serial.println(Ethernet.localIP());
 }

void loop() {

    digitalWrite(pauseEthernet, HIGH); //Désactive le schield Ethernet
    if (message != 0){ // si message est différent de 0
    //char inChar = (char)message; à utiliser si vous avez besoin d'une chaîne de caractère
      Serial.println(inChar);
      Serial.println(message); //écrire sur le port série ce qui se trouve dans la variable message
    }
    if (message == '1'){
        digitalWrite(brocheLed, HIGH); // si la variable message est parfaitement = à 1 activer la broche N°8
(brocheLed)
        }
    if (message == '0'){  // si la variable message est parfaitement = à 0 désactiver la broche N°8 (brocheLed)
        digitalWrite(brocheLed, LOW);
        }
    digitalWrite(pauseEthernet, LOW); //active le schield Ethernet
    delay(2);
```

```
 // cette ligne écoute des clients Ethernet éventuels
EthernetClient client = server.available();
if (client) {
  Serial.println("Nouveau client");
  // la requête http se termine par une ligne vide
  boolean currentLineIsBlank = true;
  while (client.connected()) {
    if (client.available()) {
      char c = client.read();
      Serial.write(c);
       */
       lorsque l'on arrive à la fin de la ligne et que la ligne est vide, la requête http est terminée
       il est alors possible de répondre
       /*
      if (c == '\n' && currentLineIsBlank) {
       // envoi une en-tête http standard en réponse au client (navigateur)
        client.println("HTTP/1.1 200 OK");
        client.println("Content-Type: text/html");
        client.println("Connnection: close");
        client.println();
        client.println("<!DOCTYPE HTML>");
        client.println("<html>");
        client.println("<meta charset=""utf-8"" />"); // impose la police de caractère utf8 (accents)
        client.println("<meta http-equiv=\"refresh\" content=\"2\">"); // Rafraîchie la page toutes les 2 secondes
```

```
// Cette partie envoi la valeur de l'entrée analogique 0
 {int entreeAnalogique = 0;
 int lectureCapteur = analogRead(entreeAnalogique);
 client.print("l'entrée analogique ");
 client.print(entreeAnalogique);
 client.print(" est à ");
 client.print(lectureCapteur);
 client.println("<br />");

// Cette partie envoi la valeur de l'entrée numérique 4
 int lectureInter = digitalRead(brocheInter);
 client.print("l'entrée numérique ");
 client.print(brocheInter);
 client.print(" est à ");
 client.print(lectureInter);
 client.println("<br />");

 int affichageLed = digitalRead(brocheLed);
 client.print("la LED broche ");
 client.print(brocheLed);
 client.print(" est à ");
 client.print(affichageLed);
 client.println("<br />");
```

```
        // Cette partie allume la led en broche N°8
          int etatInter = 0;
          etatInter = digitalRead(brocheInter);
          // Vérifie si l'interrupteur est appuyé
          // S'il est à l'état haut passer la broche de la led à l'état haut
          if (etatInter == HIGH) {
            digitalWrite(brocheLed, HIGH);
            }
          else {
            // Sinon passer la broche de la led à l'état bas
            digitalWrite(brocheLed, LOW);
            }
        }
        client.println("</html>");
        break;
        }
        if (c == '\n') {
        // début d'une nouvelle ligne
        currentLineIsBlank = true;
        }
          else if (c != '\r') {
          // il y a un caractère sur la ligne courante
        currentLineIsBlank = false;
        }
    }
  }
```

```
  // laisse au navigateur le temps de recevoir les données
   delay(1);
   // fermeture de la connexion avec le client (navigateur)
   client.stop();
   Serial.println("client déconnecté");
  }

    void serialEvent(); {
  /*
SerialEvent se produit chaque fois que des nouvelles données proviennent de la liaison RX matériel. Cette routine est exécutée entre chaque boucle de temps (), donc elle peut retarder la réponse. Plusieurs octets de données peuvent être disponibles.
  */
    while (Serial.available()) { // cette fonction permet de savoir combien de caractères ont été mémorisés dans le tampon, 128 maximum
      message = Serial.read(); // lit le premier caractère contenu dans le tampon
      Serial.flush (); // permet de vider le tampon du port série ''pas toujours nécessaire''
      }
  }
}
```

A suivre … sur http://technoopen.org/



Johan Rungette

WEBOGRAPHIE :


http://www.aeroseed.com/innovation/eolienne.php
http://www.mon-club-elec.fr/
http://jeromeabel.net
http://www.tavernier-c.com
http://blog.nicolargo.com
http://fritzing.org/home
http://fr.wikipedia.org
http://fr.flossmanuals.net
http://vimeo.com/52000418
http://www.eolprocess.com
http://www.ibfriedrich.com
http://fr.openclassrooms.com
http://atelier-meteo-arduino-lycee-vincendo.blogspot.fr
http://technoopen.org/


Merci

# Table des matières