Premiers pas dans la domotique :

Comment commander à distance

des appareils

du bout des doigts ?

Quelques notes sur l'installation et la configuration *pas-à-pas*
d'un microcontrôleur Arduino, d'un Raspberry Pi et du logiciel OpenRemote
pour le contrôle d'un relais ou de volets roulants électriques
comme exemples simples d'applications domotiques faites soi-même.

D. Swinnen - Août 2014

Le raccordement du relais pour commander des lampes ou quoi que ce soit en tension 220V nécessite des précautions pour éviter des accidents qui peuvent conduire à de graves brulures, voire à la mort dans les cas les plus graves!

Faire des raccordements sur le réseau électrique n'est pas sans risque.

Si vous ne disposez pas des compétences adéquates en haute tension où si vous hésitez, faites appel à quelqu'un en disposant.

L'utilisation de ces instructions est à votre risque.

Introduction 4

1. Commander un relais depuis son smartphone 5

Liste du matériel nécessaire 5

Etape 1 : Installation Raspberry Pi avec IP fixe 5

Etape 2 : Configuration de DNS dynamique 8

Etape 3 : raccordement du relais et programmation Arduino 11

Etape 4 : tester votre montage depuis un réseau externe! 20

2. Commander un relais depuis son smartphone à l'aide du logiciel OpenRemote (partie 1) 22

Introduction : Brève description de OpenRemote et de son fonctionnement 22

Etape 1 : Installation de Java SE sur Raspberry Pi 25

Etape 2 : Installation de OpenRemote sur Raspberry Pi 26

Etape 3 : Création de la télécommande sous Android et / ou iOS à l'aide de OpenRemote 29

3. Commander un relais depuis son smartphone à l'aide du logiciel OpenRemote (partie 2) 40

Etape 1 : Installation du service VPN sur Raspberry Pi 40

Etape 2 : Test du VPN pour le contrôle du relais 44

4. Commander des volets roulants depuis son smartphone 46

Introduction 46

Etape 1 : Création d'un shield Arduino d'interface avec les télécommandes 48

Etape 2 : Programmation Arduino 51

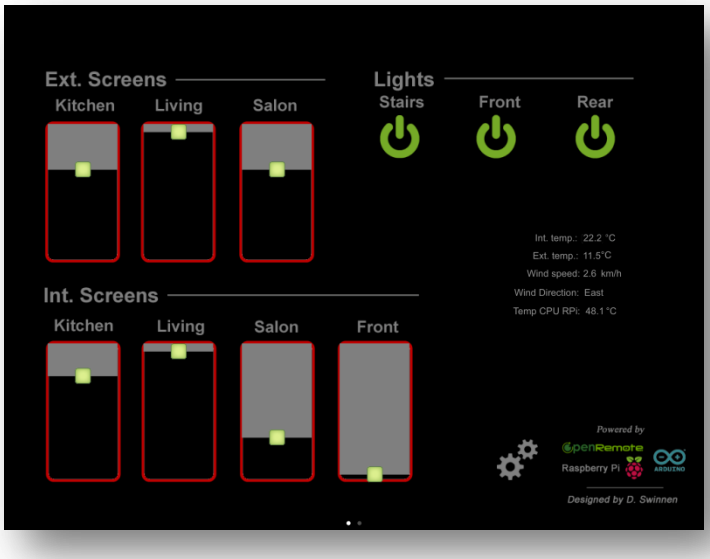
Etape 3 : Installation d'un curseur de position ("slider") qui permet de définir la hauteur des volets roulants 63

5. Perspectives 66

Introduction

Depuis 6 mois je me suis intéressé à la **domotique** et j'ai décidé de me plonger un peu dans le domaine. Il existe de nombreuses solutions commerciales, mais j'ai préféré créer mon petit système ou tout du moins une partie, non pas seulement car c'est probablement moins onéreux, mais car c'est tellement plus amusant de le faire par soi même (le fameux "**Do It Yourself**"). Comprendre le fonctionnement, chercher des solutions ou simplement découvrir sur le net que nous sommes nombreux à avoir les mêmes envies, les mêmes problèmes mais pas forcément les mêmes solutions. Apprendre quoi !

Mon projet, en deux mots, consiste à intégrer un système **domotique** simple dans ma maison pour commander quelques **éclairages**, quelques **prises de courant** et enfin pour commander l'ouverture ou la fermeture de **volets roulants** en fonction **des informations météo** trouvées sur le net.



Le tout, en essayant de ne pas exploser le budget et en étant convivial par simple « clics » sur ma tablette ou mon smartphone. Et surtout en étant sécurisé car on parle ici de commande de tension 220V (!)

En parcourant le net, j'ai choisi dans la mesure du possible de combiner du matériel et des logiciels dits « **Open Source** ». Je suis vite tombé sur des logiciels vraiment bien conçus tels que Domoticz et OpenRemote tournant sur un petit ordinateur au format carte de banque, le Raspberry Pi, et enfin des microcontrôleurs de type Arduino.
Scientifique de formation avec quelques notions de programmation et d'électronique, je ne savais pas ce qu'était un Arduino ni un Raspberry Pi, jusqu'à ce que je rencontre un autre Dominique qui m'a transmit son virus…. Je me suis d'ailleurs grandement inspiré de ses tutoriels, Wikis et Blogs mais aussi d'autres ressources comme par exemple le livre de Marco Schwartz (« Build Home Automation Systems with Arduino »). J'essaierai de ne pas oublier de continuer à mentionner mes sources !

Vous trouverez dans la suite de ces notes une petite série de tutoriels sur la **commande d'un simple** relais depuis un panneau de commande sur **smartphone ou tablette** avec **retour de l'information sur l'état du relais,** et fonctionnant depuis **n'importe où** (!). Ensuite, une application pour le **contrôle à distance de volets roulants**. Le tout avec un **minimum de programmation** pour ceux qui comme moi, sont des débutants bricoleurs en automation ou en domotique.

A vous d'adapter ces quelques notes à votre projet bien plus complexe…

1. Commander un relais depuis son smartphone

En suivant ces notes, il sera possible d'allumer une lampe (ou quoi que se soit) par l'intermédiaire d'un relais, par simple commande http avec votre navigateur web préféré, sur tablette smartphone ou ordinateur, et d'avoir l'état du relais (ici allumé ou éteint) par le retour d'un fichier XML. La commande de ce relais à distance fonctionnera depuis n'importe où, depuis vos vacances ou votre lieu de travail (pour autant que vous aillez accès au réseau GSM/internet adéquat).

Liste du matériel nécessaire

1 Raspberry Pi - env 37 EUR
1 Arduino Uno R3 (Atmega328 - assemblé) – env 22 EUR
1 Ethernet Shield pour Arduino – env 48 EUR
1 module relais – env 8 EUR
1 carte SD 4 Gb – env 8 EUR

La grande partie de la programmation de l'Arduino en serveur XML est inspirée de celle trouvée sur l'un des tutoriels de MC Hobby !

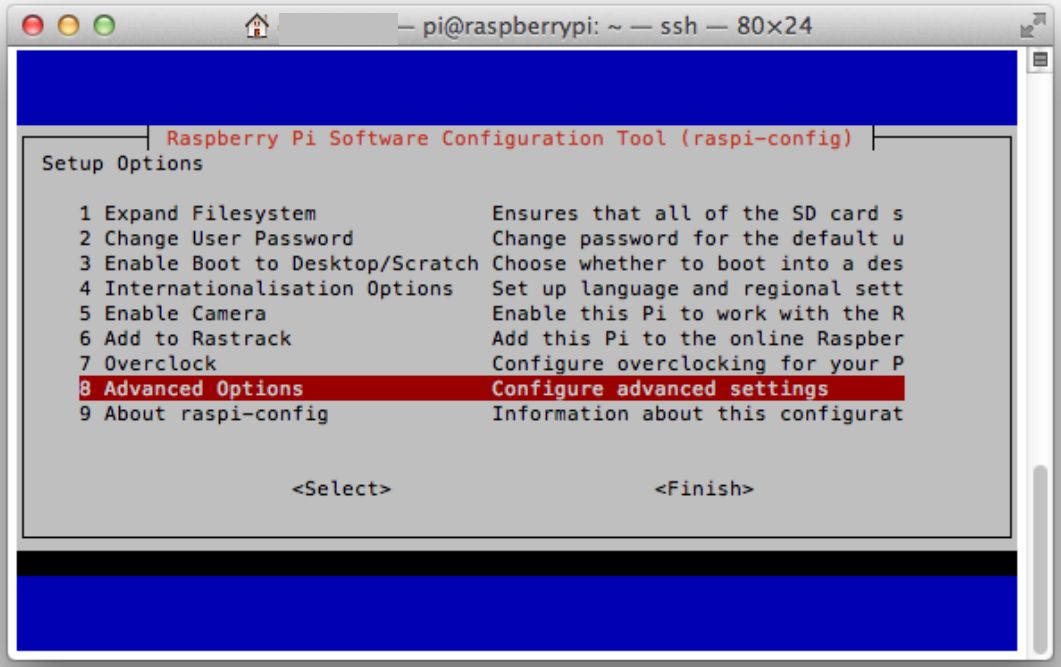
Etape 1 : Installation Raspberry Pi avec IP fixe
Etape 2 : Configuration de DNS dynamique
Etape 3 : Raccordement du relais et programmation Arduino
Etape 4 : Teste du montage depuis un réseau externe!

Etape 1 : Installation Raspberry Pi avec IP fixe

L'installation du logiciel OpenRemote sur Raspberry Pi est bien décrite ici. Ci-dessous, vous trouverez mes notes car j'ai dû m'y prendre à plusieurs fois, malgré les notes détaillées disponibles. Pour commencer, nous utiliserons une carte SD avec le système d'exploitation Wheezy Raspbian pour Raspberry Pi (à télécharger ici) qui peut être installé suivant le tutoriel et les instructions de MC Hobby. Alternativement, vous trouverez des cartes SD avec le système d'exploitation déjà préinstallé.

Une fois que celui-ci est installé sur la carte SD et après avoir démarré votre Raspberry Pi, je vous recommande de changer de mot de passe (utilisateur : « pi », mot de passe initial « Raspberry ») vers quelque chose de plus personnel (et sécurisé). Pour configurer le Raspberry Pi, tapez simplement

```
$ sudo raspi-config
```



Dans la suite de cette note, j'utiliserai l'interface SSH pour commander à distance la carte Pi. Pour cela, le serveur SSH doit être activé dans la configuration (dans le menu des options avancées). Une description détaillée de la mise en place du mode SSH, de son utilisation ainsi que de la description des clients SSH (par exemple PuTTY sur PC, Mac ou linux ou juste Terminal sur Mac) est décrite ici encore dans un des WiKi de MC Hobby.

Après votre configuration initiale, il est vraiment important de mettre votre système à jour. Tapez :

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Ensuite, il est nécessaire de configurer le Raspberry Pi avec un adresse IP fixe et non pas avec un adresse flottante attribuée bar votre serveur DHCP. Pour cela, if suffit d'éditer le fichier « interfaces » et d'ajouter quelques lignes avec un éditeur de texte (« nano » dans la suite de ces notes).

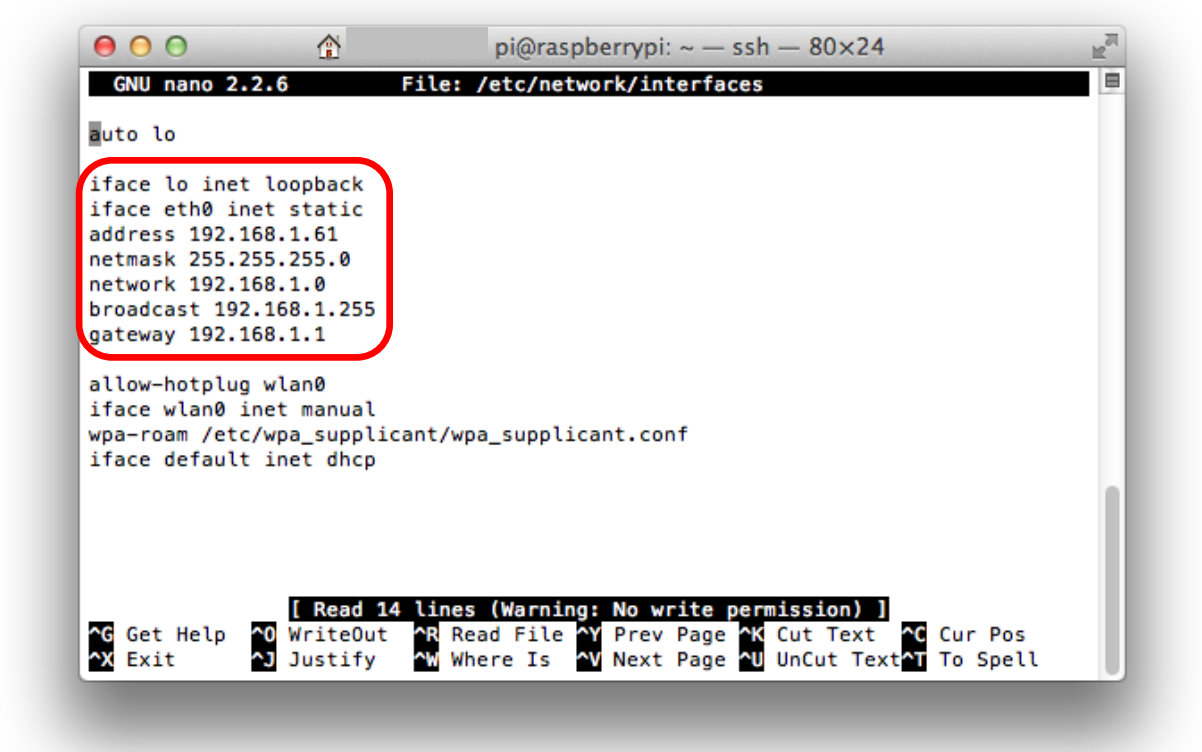
Tapez :

```
$ sudo bash
$ nano /etc/network/interfaces
```

Et changer la ligne « iface eth0 inet dhcp » par celles-ci (à adapter selon votre réseau) :

```
iface eth0 inet static
address 192.168.1.61
netmask 255.255.255.0
gateway 192.168.1.1
network 192.168.1.0
broadcast 192.168.1.255
```

La configuration ci-dessous utilise une connexion Ethernet (eth0), une adresse IP fixe 192.168.1.61 et 192.168.1.1 pour le routeur. Voici à qui ressemble le fichier une fois modifié :



Pour terminer, un « ctrl+X » suivit de « Y » et « enter » pour enregistrer la configuration IP fixe.

Pour redémarrer, tapez :

```
$ sudo reboot
```

Si comme moi vous changez de Pi ou de système d'exploitation sur une même adresse IP, vous aurez un message d'erreur au moment de la connexion SSH car la clé d'encryption pour l'adresse IP ne correspondra plus à celle de votre Pi (puisque vous avez changez d'OS ou de Pi). Pour résoudre ce problème, il suffit d'employer la commande suivante munie de l'adresse IP fixe de votre Raspberry Pi (192.168.1.61) dans la suite de mes notes:

```
$ ssh-keygen -R 192.168.1.61
```

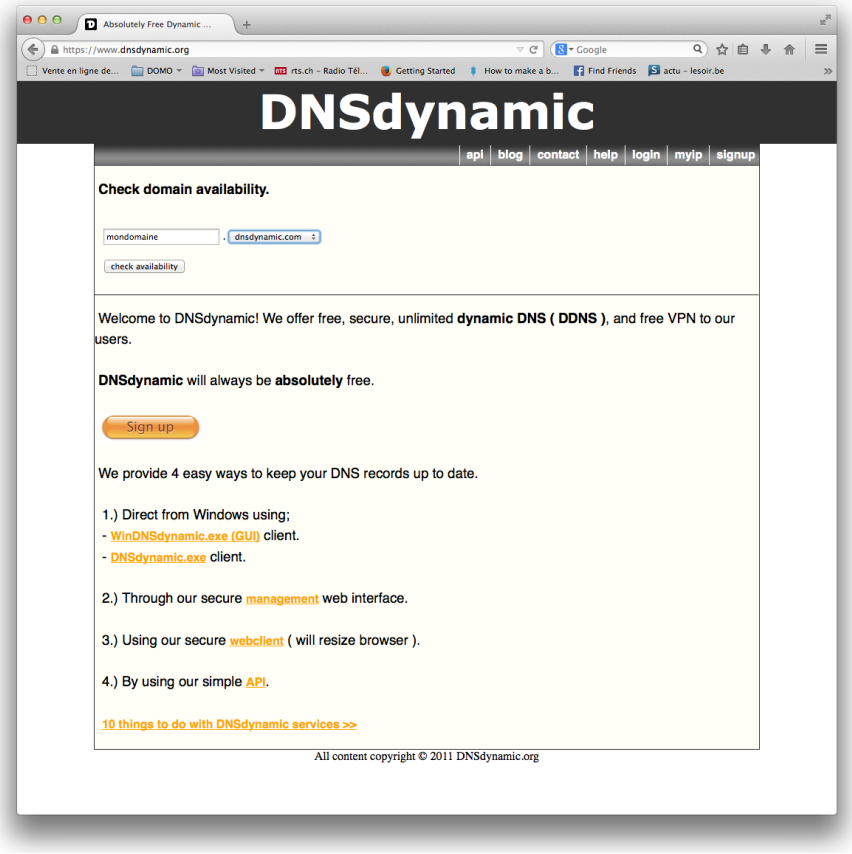
Etape 2 : Configuration de DNS dynamique

Le but ici est de mapper une adresse IP (souvent dynamique c-à-dire qui change au court du temps par l'opérateur réseaux comme Belgacom, Orange, etc.) avec un sous domaine (comme « dyndns.org » ou « dnsdynamic.com ». En clair il s'agit d'associer un nom de domaine fixe (par exemple « *mondomaine*.dyndns.org » pour une machine ayant une IP dynamique sur Internet.

Pour y parvenir, l'adresse IP dynamique (*variable*) actualisée doit être régulièrement envoyée vers un serveur hébergé par un site de DNS dynamique. Celui-ci « traduira » l'adresse IP dynamique (ex. 187.143.43.5) par une adresse plus lisible mais surtout *constante*, comme par exemple par une adresse de type « *mondomaine*.dyndns.org ». Idéal pour monter un serveur chez soi, ou contrôler sa domotique depuis n'importe où.

Un article (http://korben.info/clone-dyndns-remplacer.html) reprend quelques options pour souscrire gratuitement à un service. Pour ma part, j'ai testé le service dns.dynamic.com (www.dnsdynamic.com) car il est simple et gratuit, même si les pages web sont assez peux intuitives.

Première étape, souscrire un compte sur le site de dnsdynamic.com. Choisir un nom d'utilisateur, un mots de passe et un nom de (sous)domaine, par exemple « *mondomaine*.dnsdynamic.com ».



C’est le Raspberry qui se chargera d’envoyer régulièrement l’adresse IP du routeur vers le serveur hébergé par le site de DNS dynamique.Pour cela, il faut installer un client qui fera le boulot automatiquement. J’ai trouvé un petit programme qui tourne sur le Raspberry, l’excellent ddclient (http://sourceforge.net/p/ddclient/wiki/Home/) dont l’installation est aisée et comprends quelques lignes de configuration.

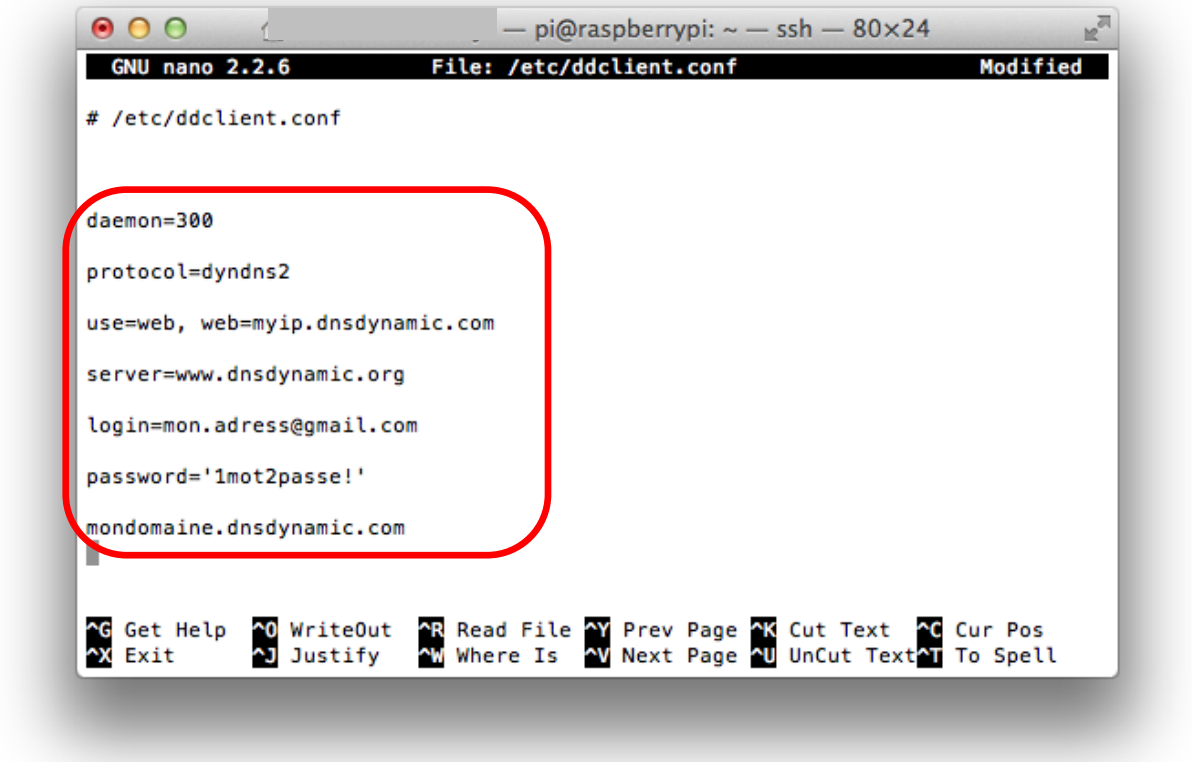
Créez le fichier /etc/ddclient.conf et du démarrage de l’application :

```
$ sudo apt-get install ddclient
$ sudo nano /etc/ddclient.conf
```

Dans ce fichier, copiez et adaptez les lignes suivantes selon votre configuration:

```
daemon=300
protocol=dyndns2
use=web, web=myip.dnsdynamic.com
server=www.dnsdynamic.org
login=mon.adress@gmail.com
password='1mot2passe!'
mondomaine.dnsdynamic.com
```

L’application ddclient enverra toute les 300 secondes (5 minutes) l’adresse IP « flottante » de votre routeur. Le fichier de configuration devient celui-ci (avec votre mot de passe et nom de domaine):



Pour démarrer l'application:

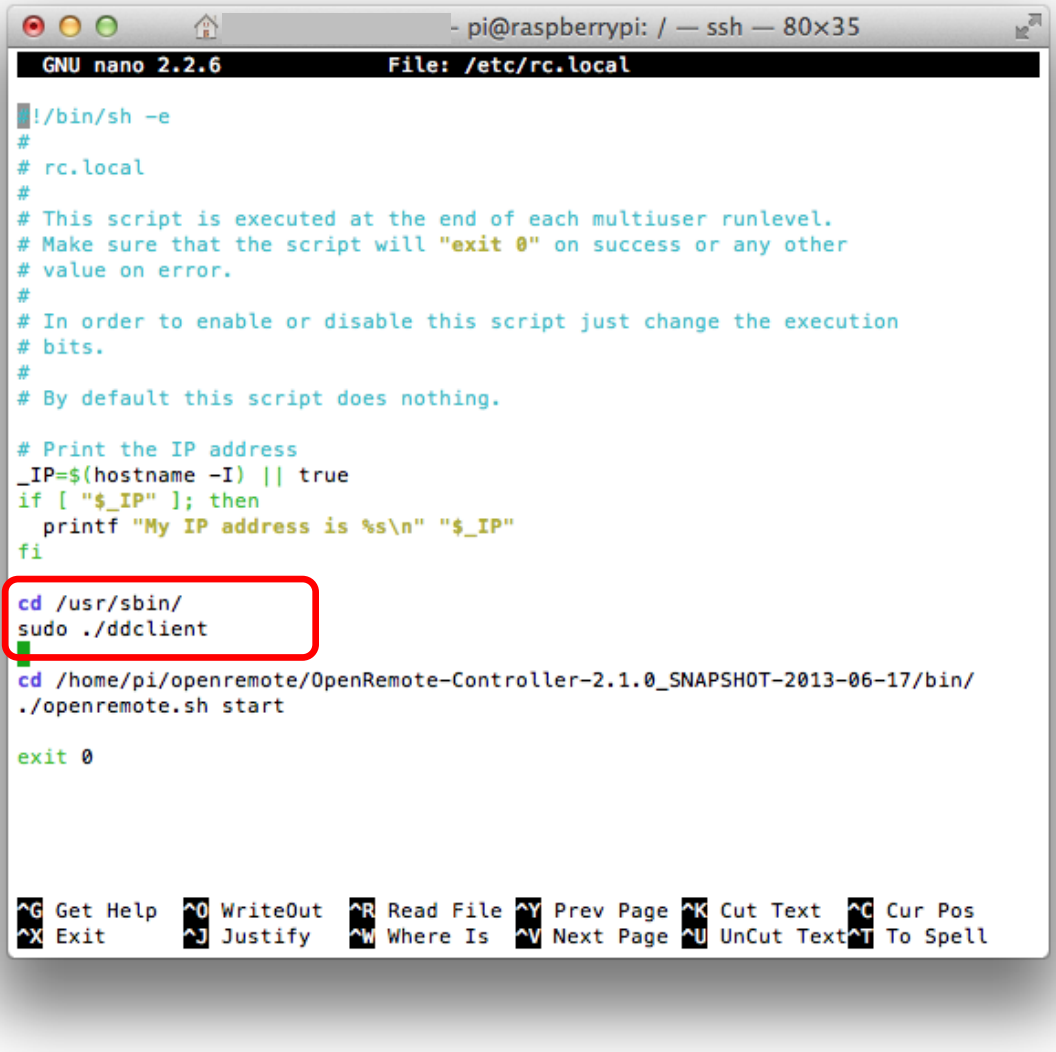
```
$ cd /usr/sbin/ddclient
$ sudo ./ddclient
```

Et enfin, pour que celle-ci puisse démarrer automatiquement lors de chaque démarrage de la carte Pi, éditez le fichier « rc.local » :

```
$ sudo nano /etc/rc.local
```

Et ajoutez les lignes suivantes juste avant la ligne « 'exit 0' » :

```
$ cd /usr/sbin/
$ sudo ./ddclient
```



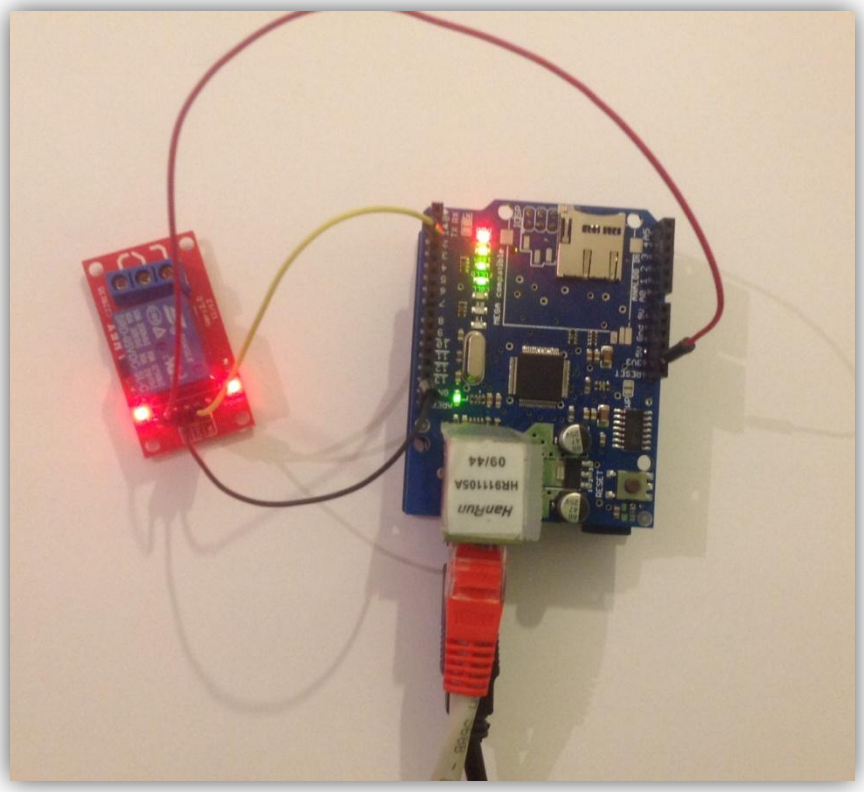
Enfin, pour que vous puissiez avoir accès à votre réseau domestique (ou une partie du moins comme par exemple l'Arduino qui commande le(s) relais), il faut ouvrir une porte dans votre routeur. Dans mon cas, j'ai ouvert la porte 9999 pour l'adresse de mon Arduino équipé d'un shield Ethernet à l'adresse IP 192.168.1.63. Lorsque le routeur sera configuré, l'Arduino pourra être connecter à l'adresse mondomaine.dnsdynamic.com:9999 , depuis n'importe où sur le web!

Ici, la configuration du routeur :

Name	Activated	Protocol	Public start port	Public end port	LAN start port	Local IP Address	Action
DIY_Web_Relay	Yes	TCP	9999	9999	80	192.168.1.63	

Etape 3 : raccordement du relais et programmation Arduino

Ici encore, toutes les informations concernant le montage du relais et son raccordement à la carte Arduino sont bien documentées sur le web, et en particulier sur le site WiKi de MC HOBBY (http://mchobby.be/wiki/index.php?title=Module_Relais).



Le code Arduino pour ce projet est amplement inspiré de celui développé par Dominique Meurisse pour l'installation d'un serveur web mesurant la température, la lumière ou la présence (http://Arduino103.blogspot.be/2014_04_01_archive.html). Je l'ai seulement adapté au contrôle d'un relais, avec les actions (« *0* », « *1* » ou « *status* » pour désactiver, activer ou recevoir l'état du relais par simple commande http).

Dans l'exemple suivant, le shield Ethernet est configuré avec une adresse IP 192.168.1.63, le pin 2 est celui qui commande le relais. Le code que j'ai adapté pour le relais est le suivant :

```
//
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//                              DIY Web Relay
//                            D. Swinnen - July 2014
//
// This example illustrates how to control a simple relay by http command
// The relay is connected on PIN 2 of the Arduino Uno board
// The http command from a web browser as follow: "192.168.1.63:relay1_0",
"192.168.1.63:relay1_1" or "192.168.1.63:status"
//
// This code and application to control a relay are inspired and adapted from the ones
described bx D. Meurisse
// http://Arduino103.blogspot.be/2014_04_01_archive.html
// http://mchobby.be/data-files/pi-vigilance/WebServer_BasicNode-v01b.ino
//
//
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


#include <SPI.h>
#include <Ethernet.h>

//--- Outils de dÃ©bogage ----------------------------------------
#define GG_DEBUG

#ifdef GG_DEBUG
 #define DEBUG_PRINTLN(x) Serial.println(x)
 #define DEBUG_PRINTLN2(x,y) Serial.println(x,y)
 #define DEBUG_PRINT(x) Serial.print(x)
 #define DEBUG_PRINT2(x,y) Serial.print(x,y)
#else
 #define DEBUG_PRINTLN(x)
 #define DEBUG_PRINTLN2(x,y)
 #define DEBUG_PRINT(x)
 #define DEBUG_PRINT2(x,y)
#endif

//-- Identification du Noeud -------------------------------------
// Cette information est retournÃ©e dans les differentes reponses
//  WEB. Elles peuvent etre utile pour identifier la fonctionalite
//  du Noeud
#define NODE_ID "DIY_Web_Relay"

//--- Definition des erreurs HTTP --------------------------------
// 400 Bad Request -
 The request could not be understood by the server due to malformed syntax
// 401 Unauthorized - The request requires user authentication.
// 403 Forbidden  - The server understood the request, but is refusing to fulfill it.
```

```
//          Authorization will not help and the request SHOULD NOT be repeated.
// 404 Not Found  - The server has not found anything matching the Request-URI.
// 405 Method Not Allowed - The method specified in the Request-
Line is not allowed for the resource identified by the Request-URI.
const int HTTP_ERR_BadRequest    = 400;
const int HTTP_ERR_Unauthorized   = 401;
const int HTTP_ERR_Forbidden    = 403;
const int HTTP_ERR_NotFound     = 404;
const int HTTP_ERR_MethodNotAllowed = 405;
const int HTTP_ERR_ServerError   = 500;

int Relay_State = 0;
//int photocellPin = 0;
//int photocellReading;


//--- Request Methods ------------------------------------------
const byte REQUEST_METHOD_NONE = 0;
const byte REQUEST_METHOD_GET = 1; // Post & Head are ignored

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1,63);
EthernetServer server(80);

// Buffer pour la parsing de la Ligne courante
//  de la requete HTTP (on traite une lignea la fois)
String currentLine;
#define CURRENT_LINE_BUFF_SIZE 128

// Nom de l'action dans l'URL http://192.168.1.177/monAction?param1=1&param2=x
String requestAction;
#define REQUEST_ACTION_BUFF_SIZE 10


void setup() {

 pinMode(2, OUTPUT); //pin selected to control

 //Serial.begin(9600);
 DEBUG_PRINTLN( "Setuping...." );

 // RÃ©server la taille du buffer
 currentLine.reserve( CURRENT_LINE_BUFF_SIZE+1 ); // 128 caractÃ¨res + NULL
 requestAction.reserve( REQUEST_ACTION_BUFF_SIZE+1 ); // 6 caractÃ¨re max + NULL


 // start the Ethernet connection and the server:
 Ethernet.begin(mac, ip);
 server.begin();

 DEBUG_PRINT("server is at ");
 DEBUG_PRINTLN(Ethernet.localIP());
}



void loop() {
 //Serial.print("Lecture Analogique = ");
 //Serial.println(photocellReading);   // La valeur analogique brute
```

```
 //-- Gestion des connexion entrantes --
 manageRequests();
}


void manageRequests(){
 byte charCount;
 byte requestMethod = REQUEST_METHOD_NONE;


 // Ecouter les connexions entrantes
 EthernetClient client = server.available();
 if (client) {
  DEBUG_PRINTLN("new client");
  // Une requete HTTP fini avec une ligne blanche... il faut donc la detecter
  boolean currentLineIsBlank = true;
  // SI la reponse est une erreur (qui peut etre detectee dans le parsing du Header)
  // ALORS il ne faut pas essayer de traiter la requete HTTP
  boolean skipProcessing = false;

  currentLine = "";
  charCount  = 0;
  requestMethod = REQUEST_METHOD_NONE;
  requestAction = "";

  while (client.connected() && !(skipProcessing) ){
   if (client.available()) {
    char c = client.read();

    // Pour les curieux
    //Serial.print( '.' );
    //Serial.print( c );

    // Recomposer une String (jusqu'a la taille du buffer)
    if( charCount < CURRENT_LINE_BUFF_SIZE ) {
     currentLine += c;
     charCount++;
    }

    // SI nous arrivons Ã la fin d'une ligne (en recevant un caractere de fin
    // de ligne) et que la ligne est est vide (blank), la reception de requete http
    // est terminÃ©e
    // ALORS nous pouvons traiter la demande (sauf erreur deja detectee) :-)
    if (c == '\n' && currentLineIsBlank && !skipProcessing) {

     // Traitement de la requete client (Client Request)
     processRequest( client, requestMethod, requestAction );

     break;
    }

    //--- INSPECTION LIGNE DE LA REQUETE -----------------------------
    // SI on recoit un caractere de fin de ligne dans le flux de la requete
    // ALORS on inspecte le contenu de la ligne
    //  But: Extraire les infos utiles au traitement de la requete
    if (c == '\n') {
     // DEBUG
     DEBUG_PRINT( "Inspect: " );
     DEBUG_PRINT( currentLine ); //la ligne contient dÃ©ja un CR/LF
```

```
    // Detection de la mÃ©thode GET/POST/... sur la ligne
    //  GET /setparam?id=12 HTTP/1.1

    //  Seule la methode GET est detectee (les autres sont ignorees pour le moment)
    if( currentLine.indexOf( " HTTP/1.") >= 0 ){
      DEBUG_PRINTLN( " +-> Method Detection" );
      if( currentLine.indexOf( "GET " )==0 ){
       requestMethod = REQUEST_METHOD_GET;
      }

      DEBUG_PRINTLN( " +-> Query Parsing" );
      int iSlashPos = currentLine.indexOf( "/" );
      int iEndOfQuery = currentLine.indexOf( " ", iSlashPos+1 );
      int iQuestionMark = currentLine.indexOf( "?", iSlashPos+1 );
      // DEBUG_PRINTLN( iSlashPos );
      // DEBUG_PRINTLN( iEndOfQuery );
      // DEBUG_PRINTLN( iQuestionMark );
      if( iQuestionMark >= 0 ){
       // Requete du type
       //  http://192.168.1.177/read?ID=16
       //  http://192.168.1.177/?ID=16
       // Verif longueur maximal de requestAction
       if( (iQuestionMark-(iSlashPos+1)) > REQUEST_ACTION_BUFF_SIZE ){
        sendError( client, HTTP_ERR_ServerError, "Action name exceed max length" );
        DEBUG_PRINTLN( " +-> skip processing");
        skipProcessing = true;
        break;
       }
       // Copier ne nom de l'action
       requestAction = currentLine.substring( iSlashPos+1, iQuestionMark );
      }
      else {
       // Requete du type
       //  http://192.168.1.177/read
       //  http://192.168.1.177
       // Verif longueur maximal de requestAction
       if( (iEndOfQuery-(iSlashPos+1)) > REQUEST_ACTION_BUFF_SIZE ){
        sendError( client, HTTP_ERR_ServerError, "Action name exceed max length" );
        DEBUG_PRINTLN( " +-> skip processing");
        skipProcessing = true;
        break;
       }
       // Copier ne nom de l'action
       requestAction = currentLine.substring( iSlashPos+1, iEndOfQuery );
      }
      DEBUG_PRINT( "   Action: >" );
      DEBUG_PRINT( requestAction );
      DEBUG_PRINTLN( "<" );

      DEBUG_PRINTLN( " +-> Query checks" );
      // Il doit y avoir une action
      if( requestAction.length() == 0 ){
       sendError( client, HTTP_ERR_ServerError, "Action missing!" );
       DEBUG_PRINTLN( " +-> skip processing");
       skipProcessing = true;
       break;
      }

    }

    // Reinit du buffer pour la ligne suivante
    currentLine = "";
```

```
     charCount = 0;
     // Jusqu'a preuve du contraire, la prochaine ligne est vide
     currentLineIsBlank = true;
    }
    else if (c != '\r') {
     // SI on recoit un caractere...
     // ALORS la ligne n'est pas vide.
     //
     // NB: Ignorer le caractere de Line Feed (\r qui est souvent juste devant le
retour chariot \n)
     currentLineIsBlank = false;
    }

   }

  }

  // Laisser le temps au web browser pour recevoir les donnees
  delay(1);
  // Fermer la connexion:
  client.stop();
  DEBUG_PRINTLN("client disonnected");
 }
}

// Description:
//  Renvoi un header HTML avec le code d'erreur PUIS clos la
//  connexion ethernet
// Parameter:
//  HtmlErrorCode - Code d'erreur HTML
//  sInfo - information complÃ©mentaire pour la page d'erreur
//
void sendError( EthernetClient client, int HtmlErrorCode, String sInfo){
  char c;
  DEBUG_PRINTLN( "sendError: flush input stream" );

  // Jeter le reste de la Requete HTTP Ã  la poubelle
  while( client.available() ){
   c = client.read();
   // Pour les curieux...
   //DEBUG_PRINT( '.' );
   //DEBUG_PRINT( c );
  }
  // Donner du temps au browser pour recevoir la reponse
  delay( 1 );

  DEBUG_PRINTLN( "sendError: sending error response" );
  // Envoyer un HEADER http
  client.println("HTTP/1.1 ");
  client.print( HtmlErrorCode );
  client.println(" ERROR");
  client.println("Content-Type: text/html");
  client.println("Connection: close");
  client.println();
  // Envoyer le contenu HTML
  client.println("<!DOCTYPE HTML>");
  client.println("<html>");
  client.print( "NODE_ID (" );
  client.print( NODE_ID );
  client.println( ")<br />" );
  client.print("HTML ERROR CODE (");
  client.print( HtmlErrorCode );
```

```
  client.println(")<br />");
  client.print("Info: ");
  client.println( sInfo );
  client.println("</html>");

  DEBUG_PRINTLN("sendError: sent!");
}

//------------------------------------------------------------------------
// processRequest
//------------------------------------------------------------------------
// Description:
//  Fait un traitement de la requete du client.
//  C'est ici qu'il faut ajouter les nouvelles entree.
//
// Note:
//  Si on arrive dans ce code, l'URL est verifiee et correctement formatee
//
void processRequest( EthernetClient client, byte RequestMethod, String RequestAction
){
  if( RequestMethod != REQUEST_METHOD_GET ){
   sendError( client, HTTP_ERR_MethodNotAllowed, "Method Not Allowed" );
   return;
  }
  // dÃ©tection de l'action et du traitement Ã  faire.

  if( RequestAction.equals( "help" ) ){
     // send a standard http response header
     client.println("HTTP/1.1 200 OK");
     client.println("Content-Type: text/html");
     client.println("Connection: close");
     client.println();
     client.println("<!DOCTYPE HTML>");
     client.println("<html>");
     client.print( "NODE_ID (" );
     client.print( NODE_ID );
     client.println( ")<br />" );
     client.println( "Voici la liste des actions disponibles:<br />" );
     client.println("help - HTML, affiche ce message d aide<br />" );
     client.println("status - XML, lecture de l'état du relais<br />" );
     client.println("relay1_0 relais eteint - PIN 2 LOW<br />" );
     client.println("relay1_1 relais allume - PIN 2 HIGH<br />" );
     client.println("</html>");
  }

  else
  if( RequestAction.equals( "relay1_1" ) ){

    Relay_State=1;
    digitalWrite(2, HIGH);  // set pin 4 high
    Serial.println("stop");
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/xml"); //retourne du XML
    client.println("Connection: close");
    client.println();
    client.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
    client.print("<node id=\""); client.print(NODE_ID); client.println("\">" );
    client.println("</node>");


  }
```

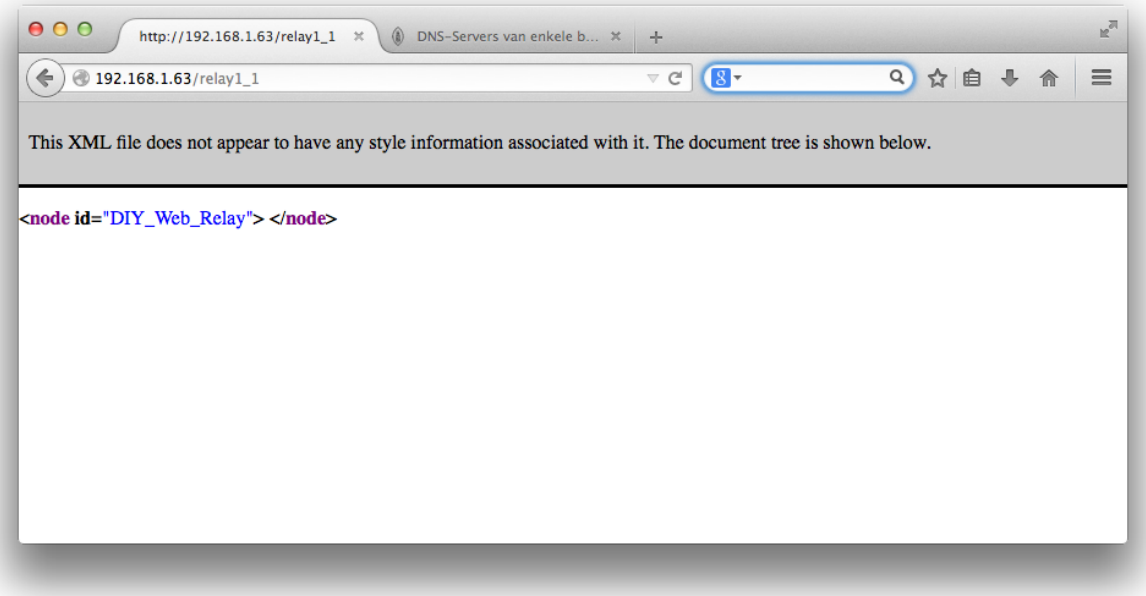
```
  else
  if( RequestAction.equals( "relay1_0" ) ){

    Relay_State=0;
    digitalWrite(2, LOW);  // set pin 4 high
    Serial.println("down");
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/xml"); //retourne du XML
    client.println("Connection: close");
    client.println();
    client.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
    client.print("<node id=\""); client.print(NODE_ID); client.println("\">" );
    client.println("</node>");
  }



  else
  if( RequestAction.equals( "status" ) ){

    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/xml"); //retourne du XML
    client.println("Connection: close");
    client.println();
    client.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
    client.print("<node id=\""); client.print(NODE_ID); client.println("\">" );
    client.print("<relay1>");
    client.print(Relay_State);
    client.print("</relay1>");
    client.println("</node>");
  }

//  else
//  if(Order=="$%7Bparam%7D"){
//
//   client.println("HTTP/1.1 200 OK");
//   client.println("Content-Type: text/xml"); //retourne du XML
//   client.println("Connection: close");
//   client.println();
//   client.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
//   client.print("<node id=\""); client.print(NODE_ID); client.println("\">" );
//   client.print("<relay1>");
//   client.print(Relay_State);
//   client.print("</relay1>");
//   client.println("</node>");
//  }

  // ELSE final... quand le nom de l'action (actionRequest) est inconnu!
  else {
   String errMsg = String( "Action inconnue (" );
   errMsg.concat( RequestAction );
   errMsg.concat( ")" );
   sendError( client, HTTP_ERR_NotFound, errMsg );
   return;
  }

}
```

Le relais devrait pouvoir être commandé simplement en entrant les ordres http suivants dans un navigateur web (dans le réseau local):

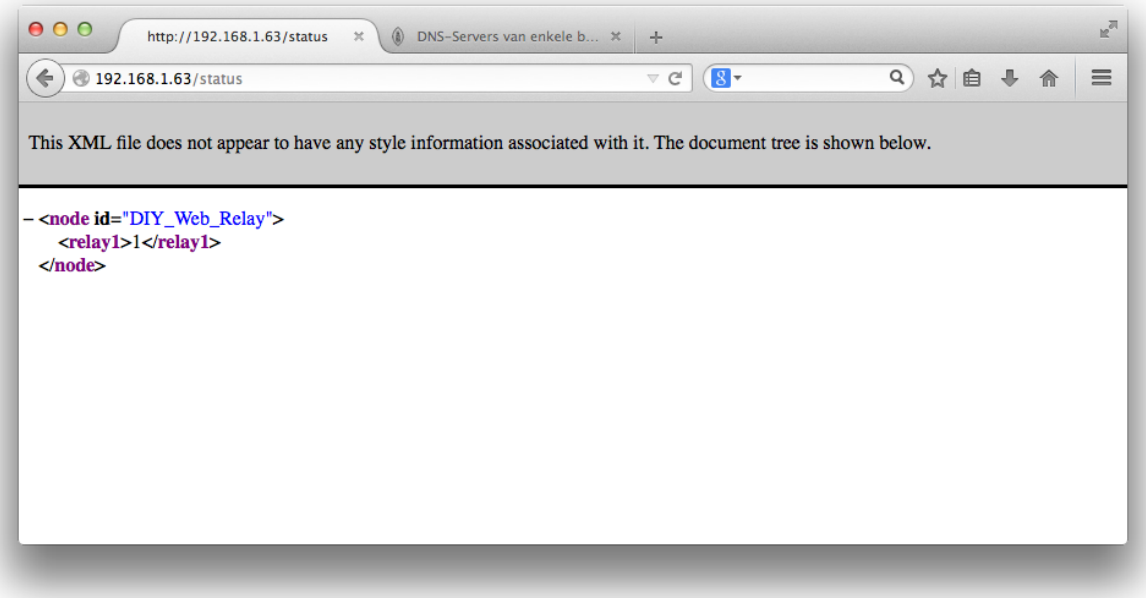
```
http://192.168.1.63/relay1_1
```

ou

```
http://192.168.1.63/relay1_0
```



Et voici le retour d'état au format XML. Veuillez noter que ce retour XML sera précieux pour la suite car il permet d'informer un système tiers de l'état du relais. Le logiciel OpenRemote, par exemple, sera capable de lire ce fichier et d'informer l'utilisateur de l'état du relais sur son interface Web (tablette, smartphone, etc…).



Etape 4 : tester votre montage depuis un réseau externe!

Pour tester votre montage depuis l'extérieur, n'oubliez pas qu'il faut sortir de votre domaine réseau « domestique ». En d'autres termes, il faut se connecter au réseau via un accès web autre que celui de votre routeur, comme par exemple à l'aide de votre smartphone *en désactivant le WiFi* pour une connexion via l'opérateur téléphonique.

Entrez la commande suivante dans votre explorateur web (ou *mondomaine* est le nom de votre domaine choisi lors de la souscription du service de dns dynamique comme dnsdynamic.com par exemple) :

```
http://mondomaine.dnsdynamic.com:9999/relay1_1
```

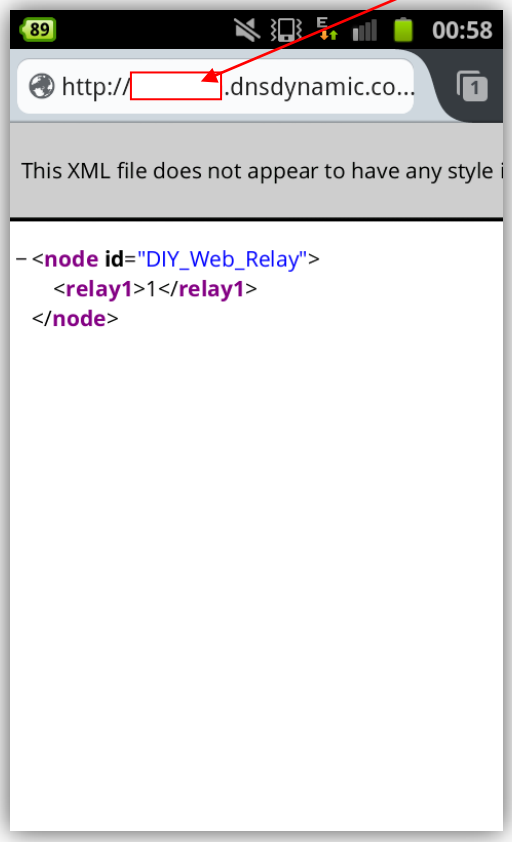
ou encore

```
http://mondomaine.dnsdynamic.com:9999/status
```

Voici une copie d'écran de mon smartphone (connecté uniquement sur le réseau GSM) lorsque la commande ci-dessus est entrée dans mon navigateur Web (ici FireFox pour Android):

Nom de votre domaine,

par ex. « *mondomaine* »



En résumé, grâce au port 9999 ouvert sur le routeur qui relie la carte Arduino et son Shield Ethernet au réseau Internet externe et grâce au client de DNS dynamique qui relie une adresse de domaine fixe et gratuite (mondomaine.dnsdynamic.com) à l'adresse flottante de votre routeur, il est maintenant possible de contrôler le relais depuis n' importe où ! Vous suivez ?

Cependant, devoir entrer des commandes de type http n'est pas très pratique, surtout quand il y en à plusieurs relais ! Dans le tutorial suivant, j'utilise le logiciel open source OpenRemote qui permet d'avoir une interface vraiment sympa pour la commande du relais par un simple click depuis la plage….

Et tout cela fonctionne, sans devoir programmer une seule ligne de code html, PHP, java, etc….

2. Commander un relais depuis son smartphone à l'aide de OpenRemote (partie 1)

En suivant ces notes, vous découvrirez une interface de commande pour contrôler un relais (ou quoi que se soit) par simple « clic » sur smartphone ou tablette (iOS ou Android), toujours avec l'information de l'état du relais (ici allumé ou éteint) et en plus synchronisé sur toutes les autres « télécommandes » comme sur la tablette de Madame et le smartphone de Monsieur... Et pas avec un look de bricolage ou de geek s'il vous plaît !

Cette première partie permettra de commander le relais dans son propre réseau local, mais pas encore en dehors. Cela fera partie de la seconde partie de ces notes.

Introduction : Brève description du logiciel OpenRemote et de son fonctionnement
Etape 1 : Installation de Java SE sur Raspberry Pi
Etape 2 : Installation du logiciel OpenRemote sur Raspberry Pi
Etape 3 : Création de la télécommande sous Android et / ou iOS à l'aide de OpenRemote

Le matériel utilisé dans ce projet est identique à celui utilisé pour le tutoriel précédent.

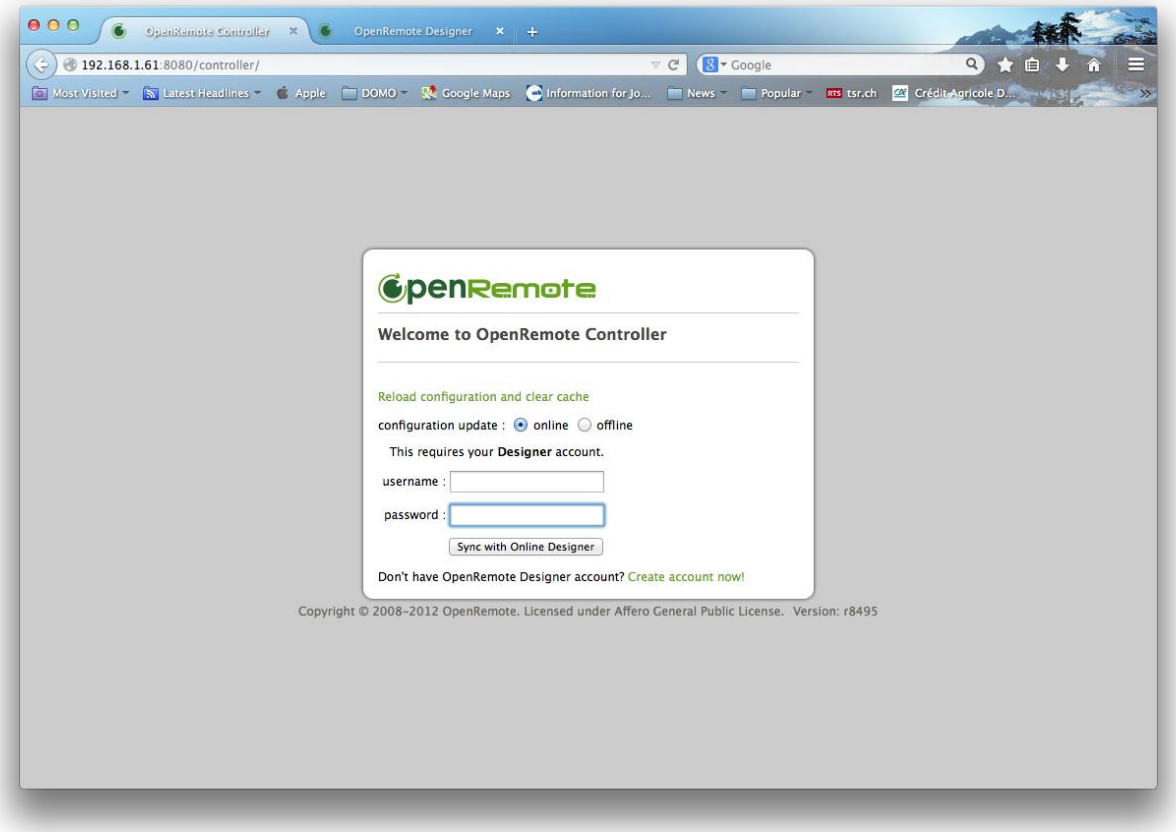
Introduction : Brève description du logiciel OpenRemote et de son fonctionnement

OpenRemote est un logiciel en Open Source (www.openremote.org) qui permet de commander toute sorte de périphériques en particuliers domotiques, quelques soient leurs protocoles, depuis votre tablette ou téléphone (iOS ou Android).

OpenRemote consiste principalement en 3 modules:

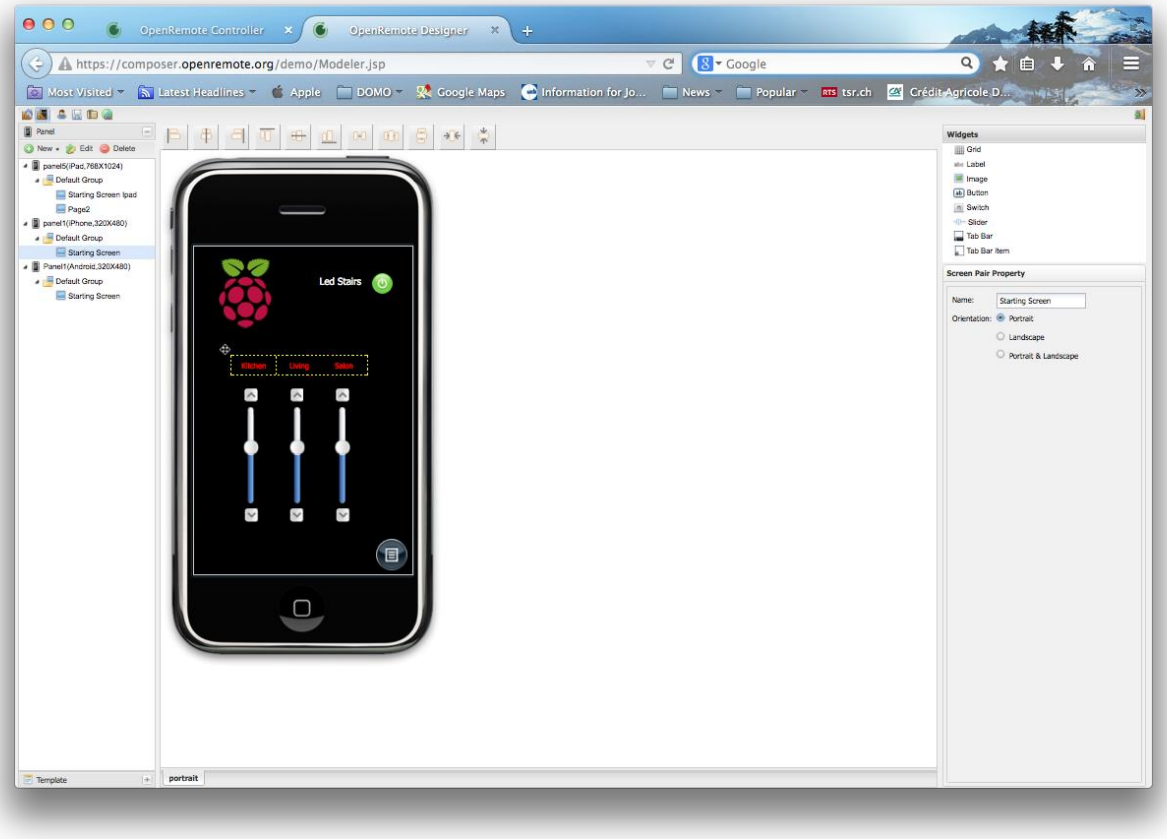
- "**OpenRemote Controller**" qui est le module principal permettant la relation entre les périphériques (par exemple domotiques) et les interfaces utilisateurs ou panneaux de contrôle comme une tablette ou un smartphone, ou plusieurs à la fois.
- "**OpenRemote Designer**" est un outil qui permet de créer et configurer le « Controller » (logiques, automations, calendrier, évènement basé sur des senseurs et/ou scripts, etc.) et de créer (« designer ») des interfaces utilisateurs ou panneaux de contrôle pour une tablette ou un smartphone. C'est un outil dont les données de design sont enregistrées dans le « cloud » c.-à-d. sur un serveur de OpenRemote (un compte utilisateur – gratuit - est donc nécessaire), mais également sur le « Controller » hébergé par le Raspberry Pi.
- " **OpenRemote Control Panels**" qui permet de contrôler et interagir avec des périphériques à travers le "Controller" .

L’accès au « Controller » est disponible depuis une interface web à l’adresse IP du Raspberry Pi, suivit du port 8080, soit dans notre exemple 192.168.1.61:8080



La partie « Designer » est quant à elle également disponible à l’adresse (https://composer.openremote.org/demo/Modeler.jsp). Ci-dessous une illustration de l’interface web pour créer un panneau de commande (ou télécommande) sur un iPhone.

Remarquez la présence d’un simple bouton poussoir vert command un relais, qui deviendra rouge lorsque le relais sera ouvert, ou 3 curseurs de position (“sliders”) qui commanderont les volets électrique roulants.

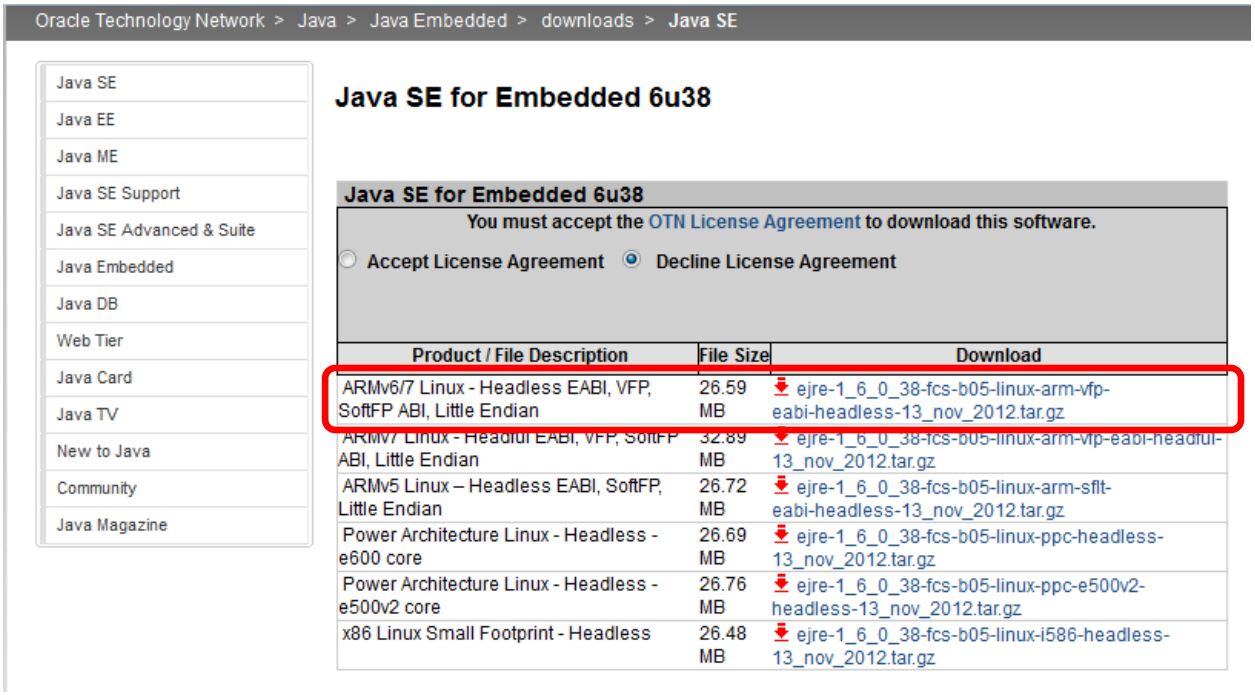


Enfin, l'interface pour tablette ou smartphone, le « OpenRemote Control Panels », se trouve sur l'App Store (iOS) ou le Google Play (Android). Cherchez « OpenRemote ».

Sur iPhone ou sur un smartphone Android:

Etape 1 : Installation de Java SE sur Raspberry Pi

Pour que le logiciel OpenRemote fonctionne, il faut installer Java, et pas n'importe quelle version ! Ici encore, j'ai suivi les instructions du site du logiciel que je détaille dans la suite car elles n'étaient pas très intuitives pour un novice comme moi… J'ai trouvé une aide à l'installation de Java, sur le site de Oracle. La version à installer est la version « ARM v6/7 » qui se trouve ici. Pour pouvoir télécharger la version de Java, vous devez accepter les termes de la licence.



Une fois le fichier téléchargé, copiez le sur le votre Pi.

Pour créer un dossier où sera placé le fichier (par exemple « /home/pi/java », tapez :

```
$ mkdir /home/pi/java
```

Pour le copier le fichier téléchargé sur mon PC ou MAC à partir de du dossier où se trouve le fichier téléchargé, à l'aide du client SSH du PC ou MAC, tapez:

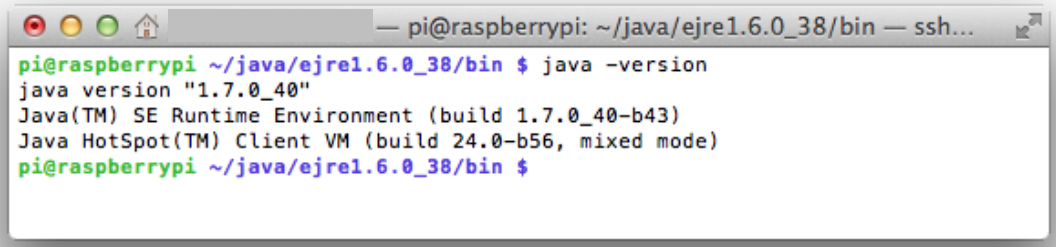
```
$ scp ejre-1_6_0_38-fcs-b05-linux-arm-vfp-eabi-headless-13_nov_2012.tar.gz
pi@192.168.1.61:/home/pi/java
```

Ensuite pour décompresser le fichier .gz, les commandes suivantes :

```
$ cd /home/pi/java
$ tar -zxvf *.gz
```

Pour démarrer java,

```
$ cd /home/pi/java/ejre1.6.0_38/bin/
$ java –version
```



Pour retrouver l'endroit ou est installé java :

```
$ which java
```

Enfin, pour être sûr que les applications trouveront correctement les fichiers nécessaires au fonctionnement du « Java virtual machine » (quelques applications Java tentent de localiser le Java VM en utilisant la variable JAVA_HOME). Vous êtes maintenant prêt à installer OpenRemote !

```
$ export JAVA_HOME=/usr
```

Etape 2 : Installation du logiciel OpenRemote sur Raspberry Pi

Les fichiers peuvent être directement téléchargés sur le site http://download.openremote.org.

Pour cela, j'utilise les commandes ci-dessous :

```
$ mkdir /home/pi/openremote
```

Et sur votre client SSH de votre PC ou MAC où se trouvent les fichiers téléchargés:

```
$ scp OpenRemote-Controller-2.1.0_SNAPSHOT-2013-06-17.zip
pi@192.168.1.61:/home/pi/openremote
```

Et à partir du votre client SSH de votre Pi:

```
$ cd /home/pi/openremote
$ unzip *.zip
$ cd OpenRemote-Controller-2.1.0_SNAPSHOT-2013-06-17/
```

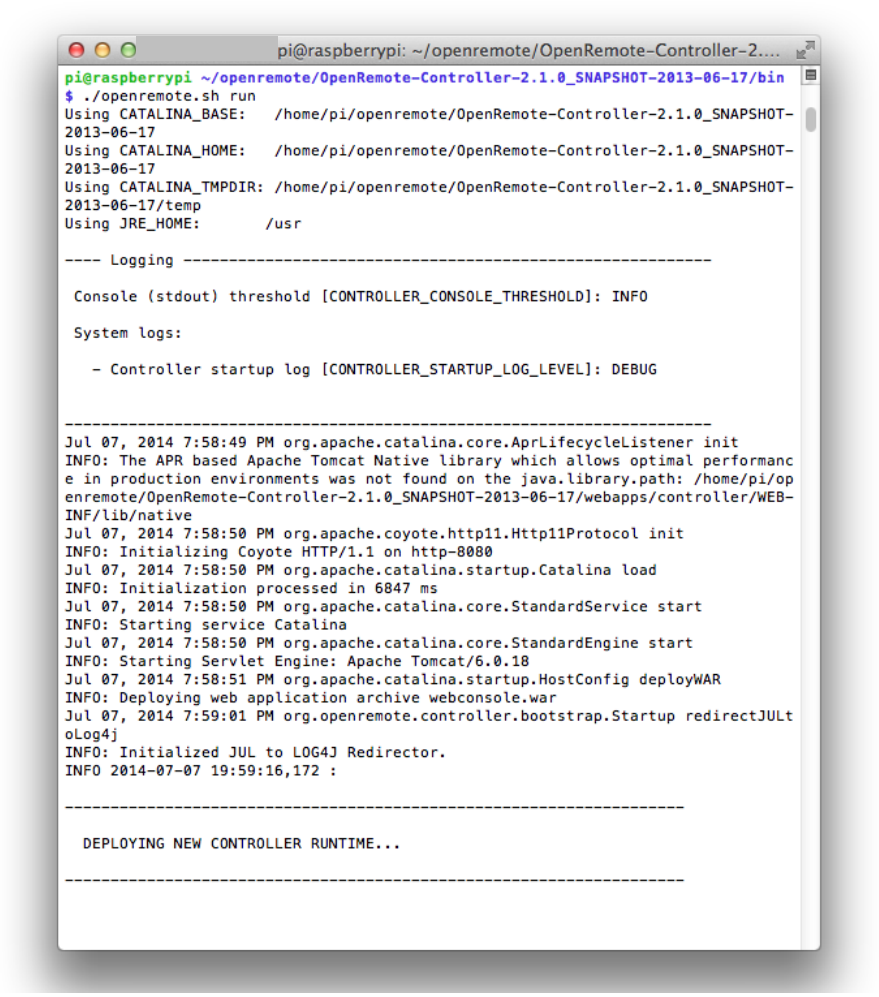
Et pour que le fichier openremote.sh puisse être exécuté :

```
$ cd bin/
$ sudo chmod +x openremote.sh
```

Enfin, nous y sommes presque, pour démarrer manuellement OpenRemote en mode SSH (qui facilite la lecture des messages d'exécution et erreur à travers le client SSH) :

```
$ ./openremote.sh run
```

Voici les messages que j'obtiens au démarrage du serveur OpenRemote à travers le client SSH (ici terminal depuis un Mac).



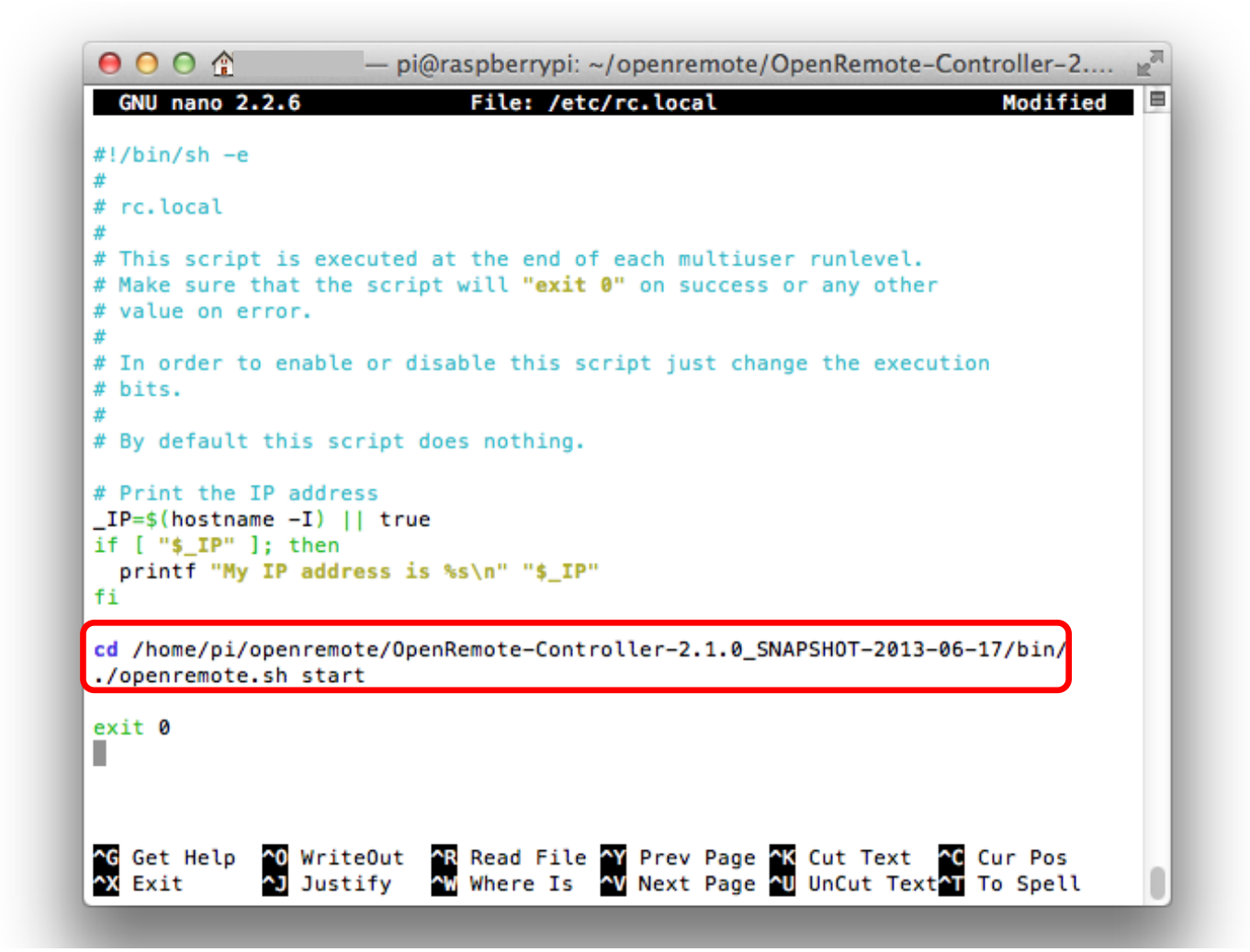
Pour que le serveur OpenRemote puisse démarrer automatiquement à chaque 'reboot', il faut éditer le fichier « rc.local » :

```
$ sudo nano /etc/rc.local
```

Et ajoutez les lignes suivantes juste avant la ligne « 'exit 0' » :

```
cd /home/pi/openremote/OpenRemote-Controller-2.1.0_SNAPSHOT-2013-06-17/bin/
./openremote.sh start
```

Voici à qui ressemble le fichier une fois modifié :

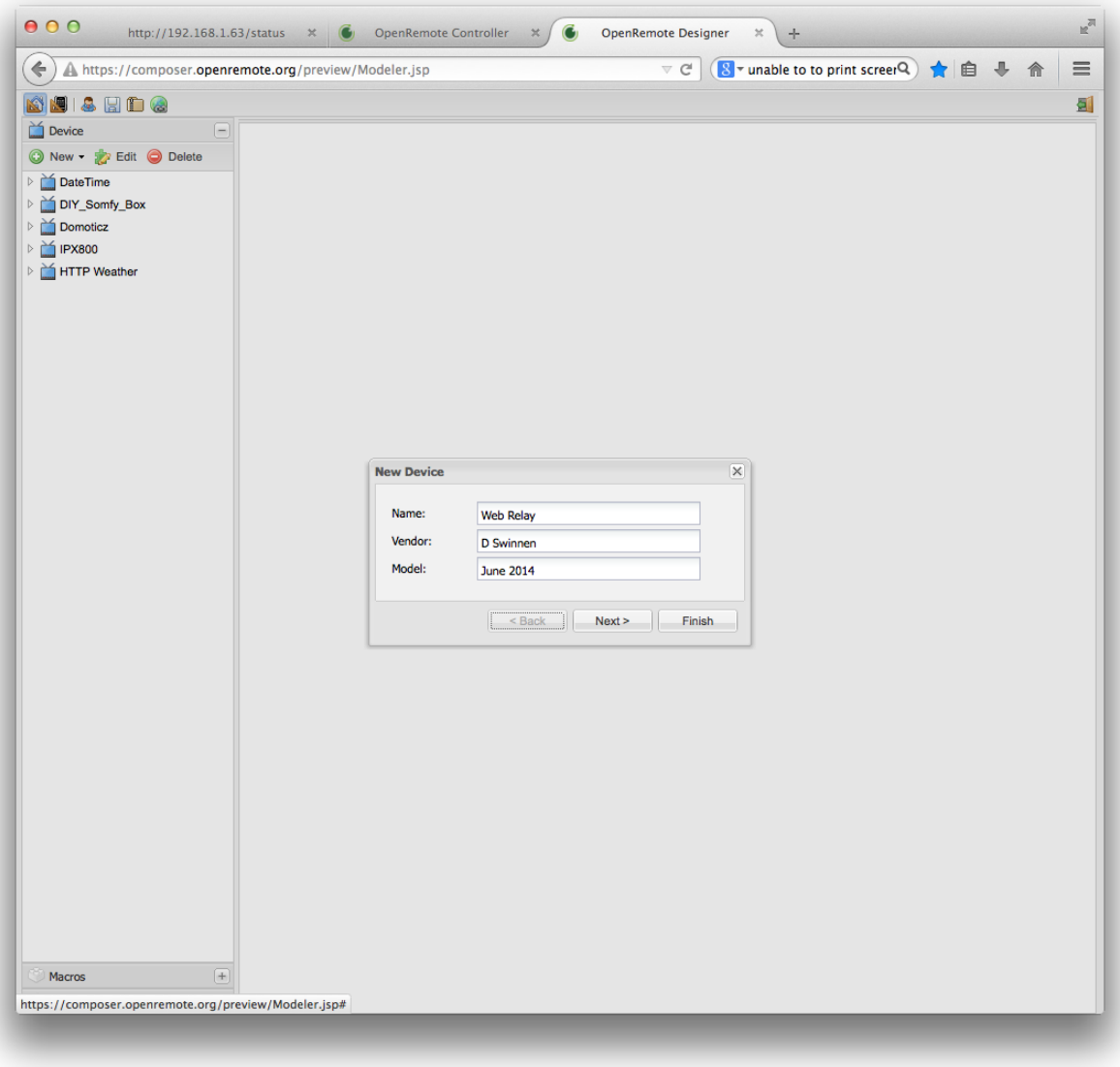


Le serveur OpenRemote démarrera ainsi automatiquement à chaque 'reboot' et pourra être consulté à son adresse IP fixe (ici dans cet exemple à l'adresse 192.168.1.61).

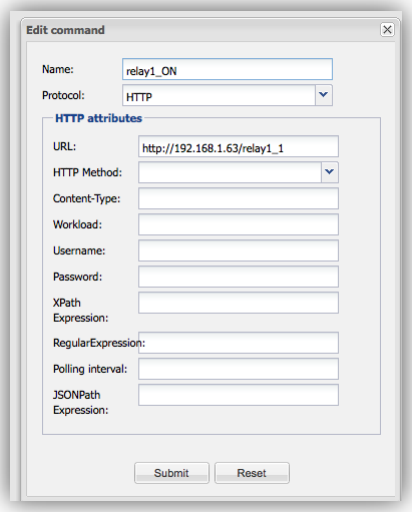
Etape 3 : Création de la télécommande sous Android et / ou iOS à l'aide du logiciel OpenRemote

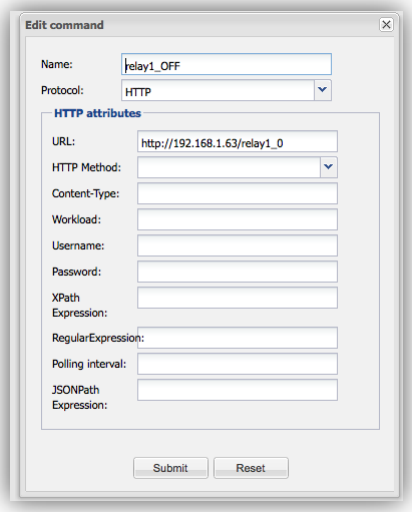
Vous trouverez de nombreuses aides et informations sur le site de OpenRemote ou son forum. J'ai essayé de représenter les étapes clés ci-dessous et en particulier le retour de l'information au format XML dans OpenRemote.

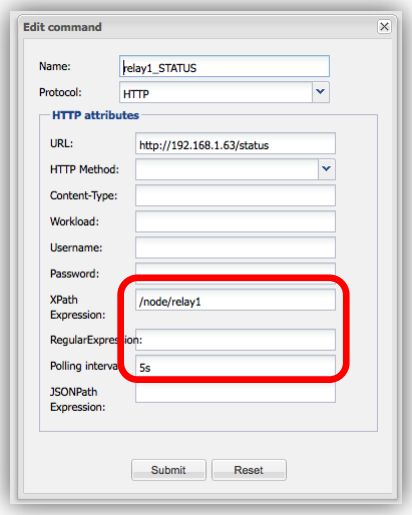
Tout d'abord, créer un nouveau périphérique (« Device »), ici nommé « Web Relay ».



Ensuite, créez 3 nouvelles commandes http selon la synthaxe qui a été utilisée dans le premier tutoriel: la commande http « allumé », « éteint» et la commande http « retour d'état » :

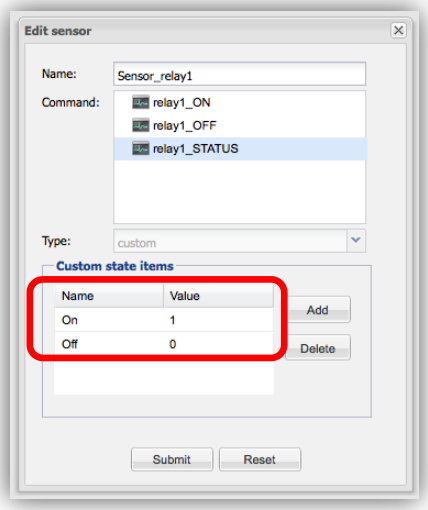




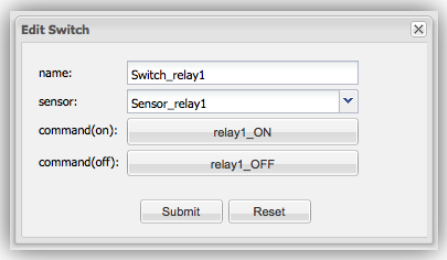


Notez le paramètre XPath (« /node/relay1 ») utile au « parsing » du fichier retour d'état au format XML, c-à-dire utile à retrouver l'information de l'état (allumé / éteint) du relais dans le fichier XML, ainsi que la fréquence à laquelle OpenRemote questionnera le statut d'état du relais à l'Arduino (paramètre « Polling interval » choisi ici de 5 secondes.

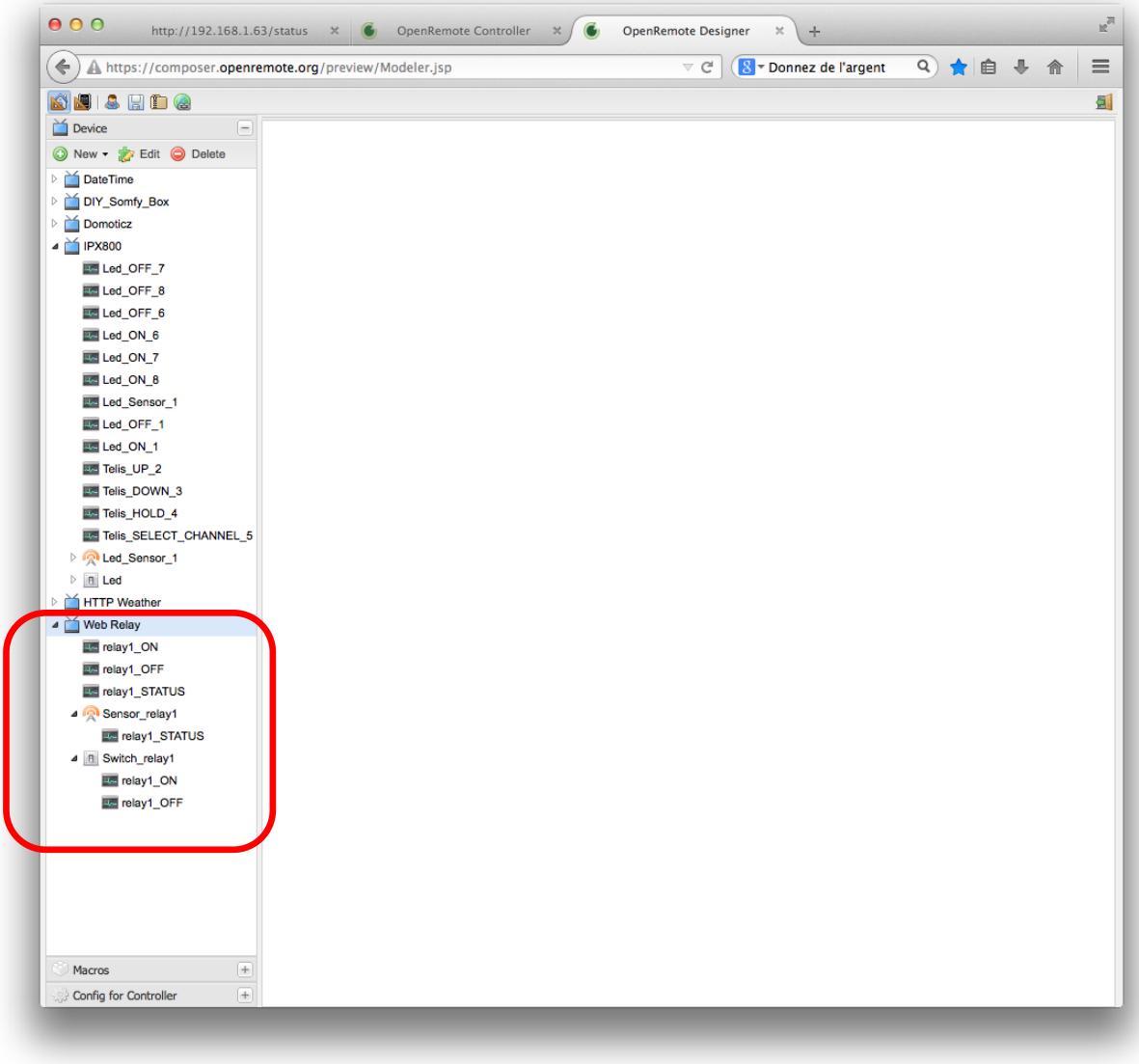
Ensuite, créez un « senseur » qui aura comme rôle de sonder l'état du relais à l'aide de la commande précédemment créée. Le type de senseur utilisé ici est « custom». Renseignez les variables « On » et « Off » par un simple « 1 » et « 0 »



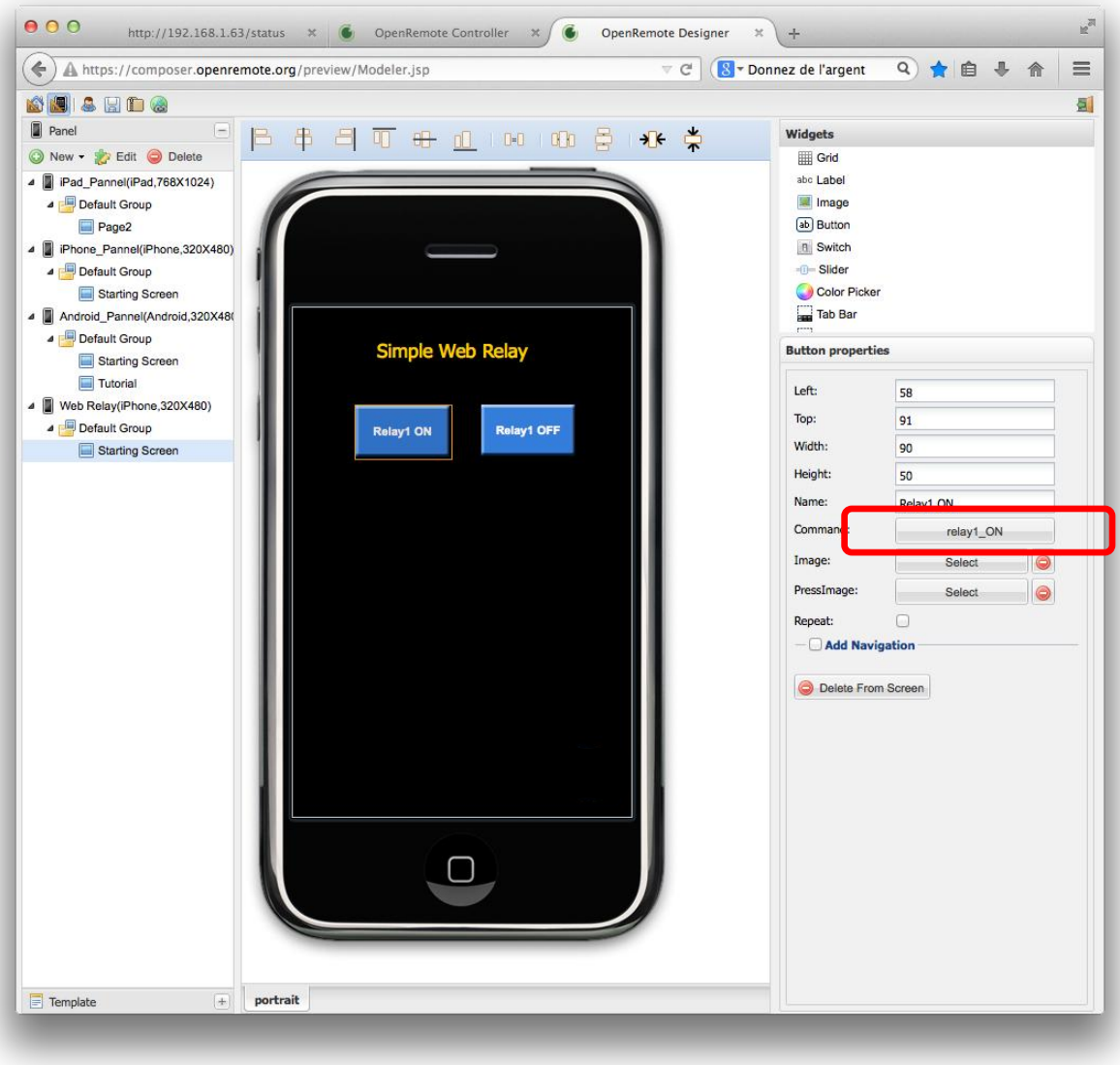
Pour terminer et rapidement illustrer le logiciel OpenRemote avec un exemple concret et simple, créez un interrupteur (« Switch ») :



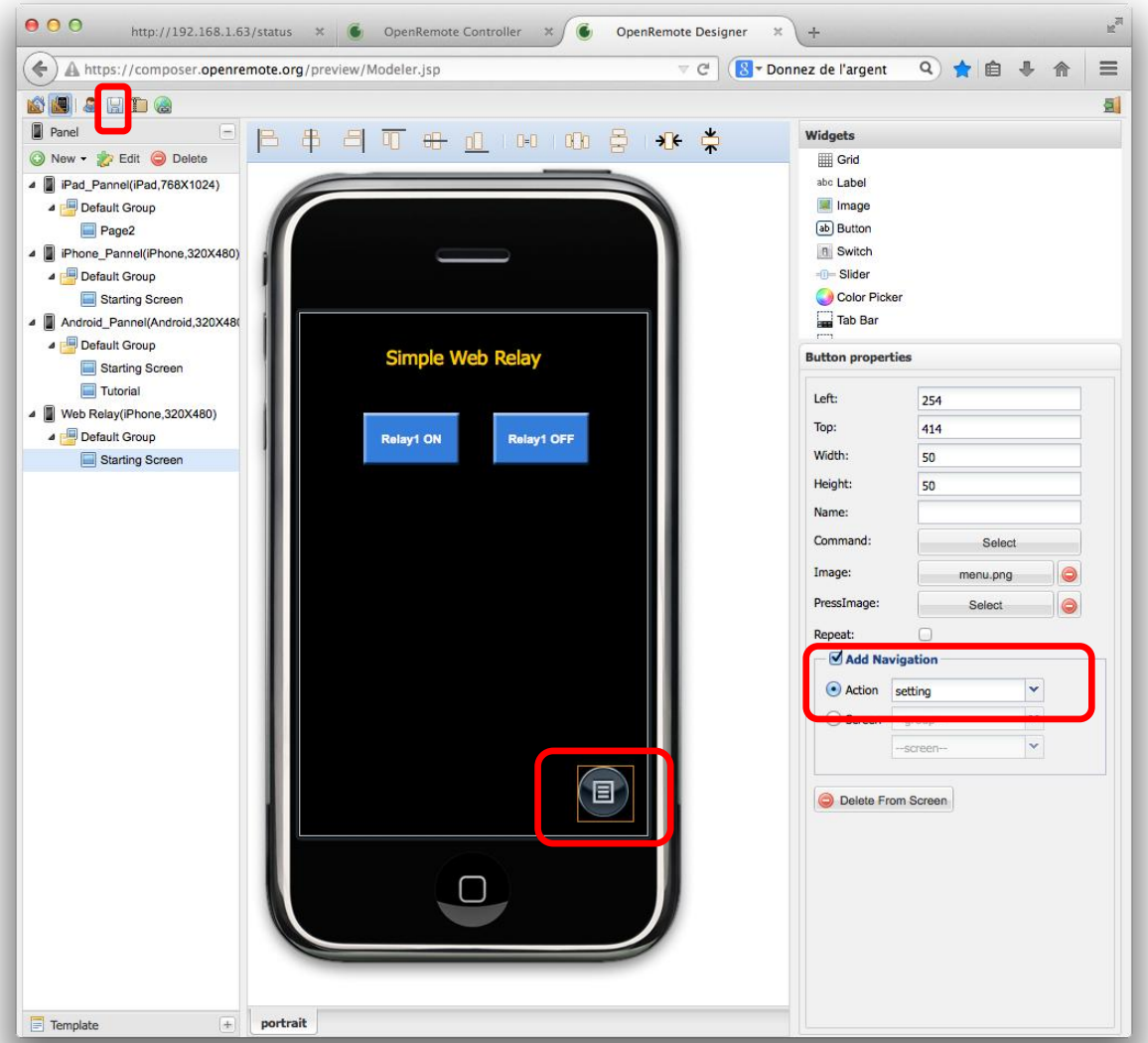
Au final, on obtient un série de 3 commandes http, un senseur pour évaluer l'état du relais et un interrupteur pour actionner les commandes allumer ou éteindre.



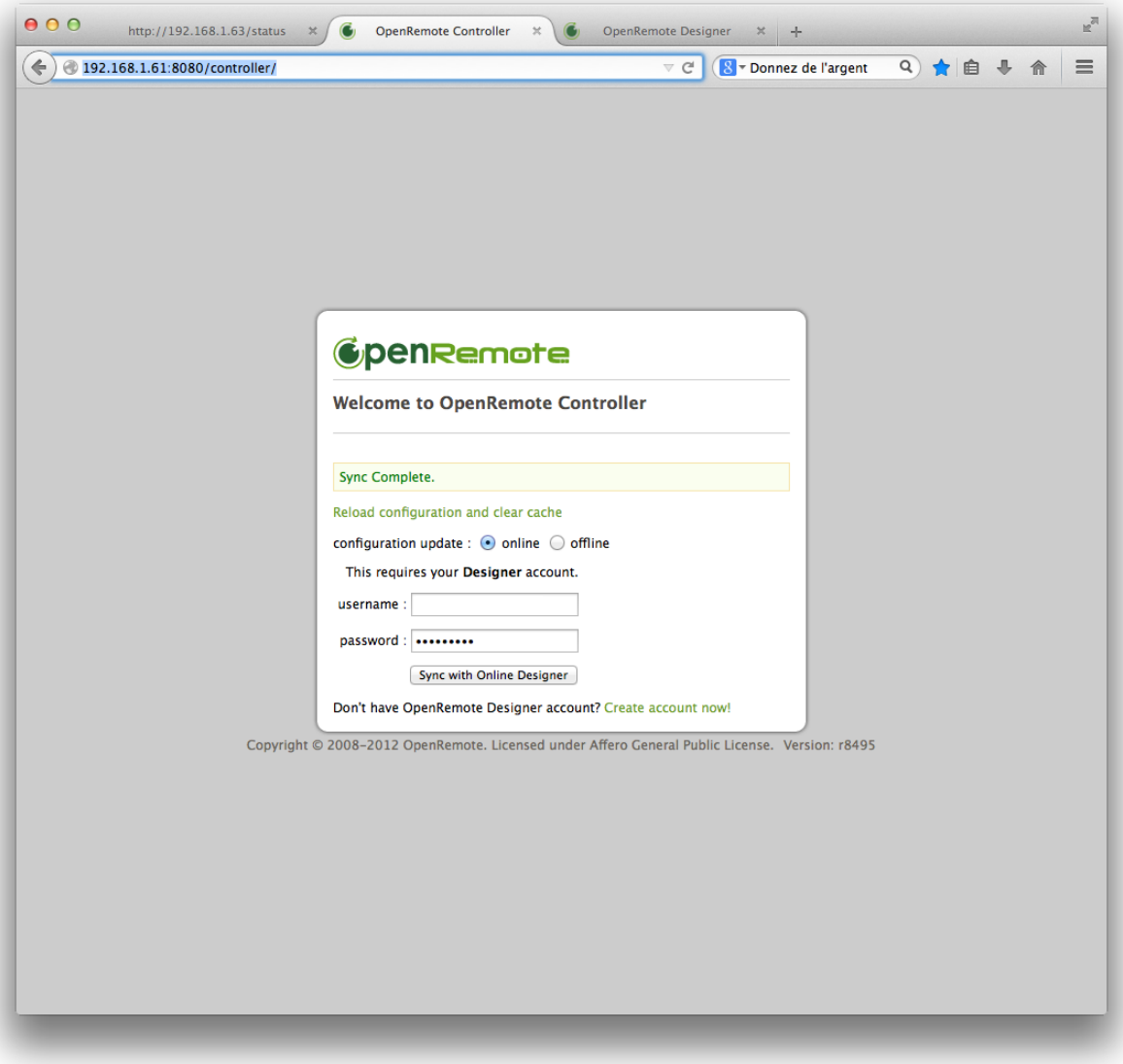
Maintenant, vous êtes prêts pour le design de l'interface tablette ou smartphone. Dans l'exemple qui suit, j'ai choisi de construire une interface simple pour iPhone. Créez tout d'abord un écran (« Pannel ») appelé ci-dessous « Web Relay ». Ensuite, en glissant sur l'écran de l'iPhone les différents objets tel que « label » et« button » sans oublier d'y associer les commandes « on » et « off » respectives. Voici que l'on obtient :



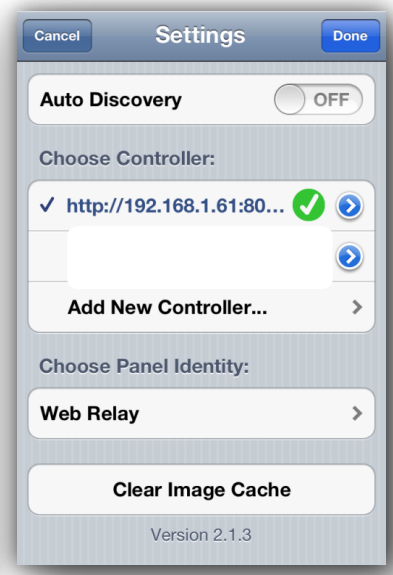
Des icones bien plus sympas, trouvées sur le net ou crées soi-même, peuvent rendre l'interface encore plus aboutie. Un conseil : ajoutez un bouton de type « setting » car il est bien utile pour paramétrer l'adresse du serveur OpenRemore lors du premier démarrage ou des essais ultérieurs ! Pour cela il suffit de glisser un bouton sur l'écran de l'iPhone et de sélectionner l'option « Add Navigation » et de choisir comme « Action » l'option « setting » :



Une fois que cette configuration effectuée dans la partie « Designer » du logiciel OpenRemote est enregistrée (à l'aide de la petite icône disquette), il faut la synchroniser avec la partie « Controller » qui pour rappel est hébergée sur le Rasberry Pi. Pour cela, accédez à la page du « Controller » à l'adresse http://192.168.1.61:8080/controller/ avec votre identifiant et mot de passe pour y avoir accès :

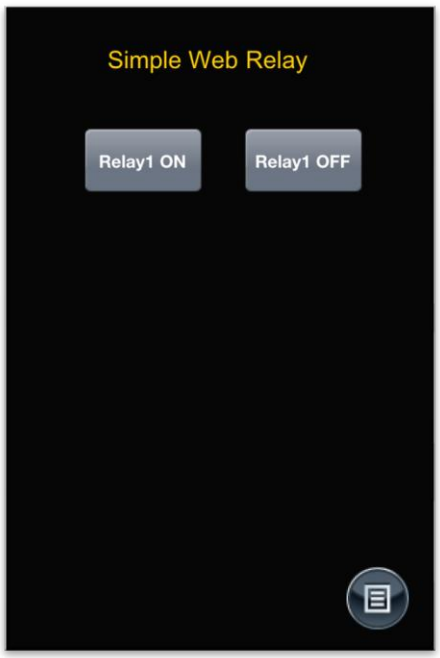


Du coté du « client » c-à-dire des interfaces tablette ou smartphone, if faut installer OpenRemote (sur iOS dans les illustrations qui suivent) :

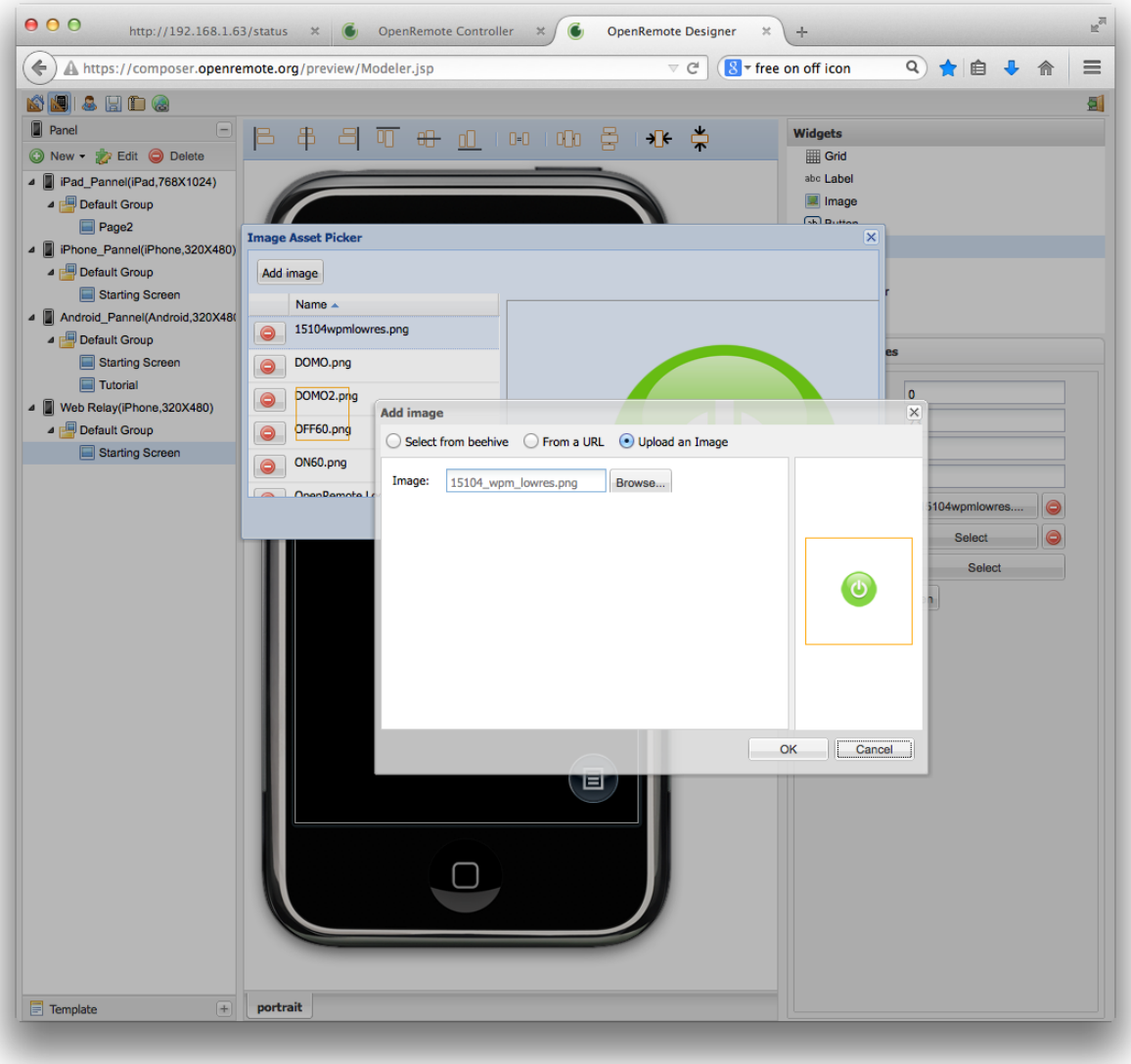


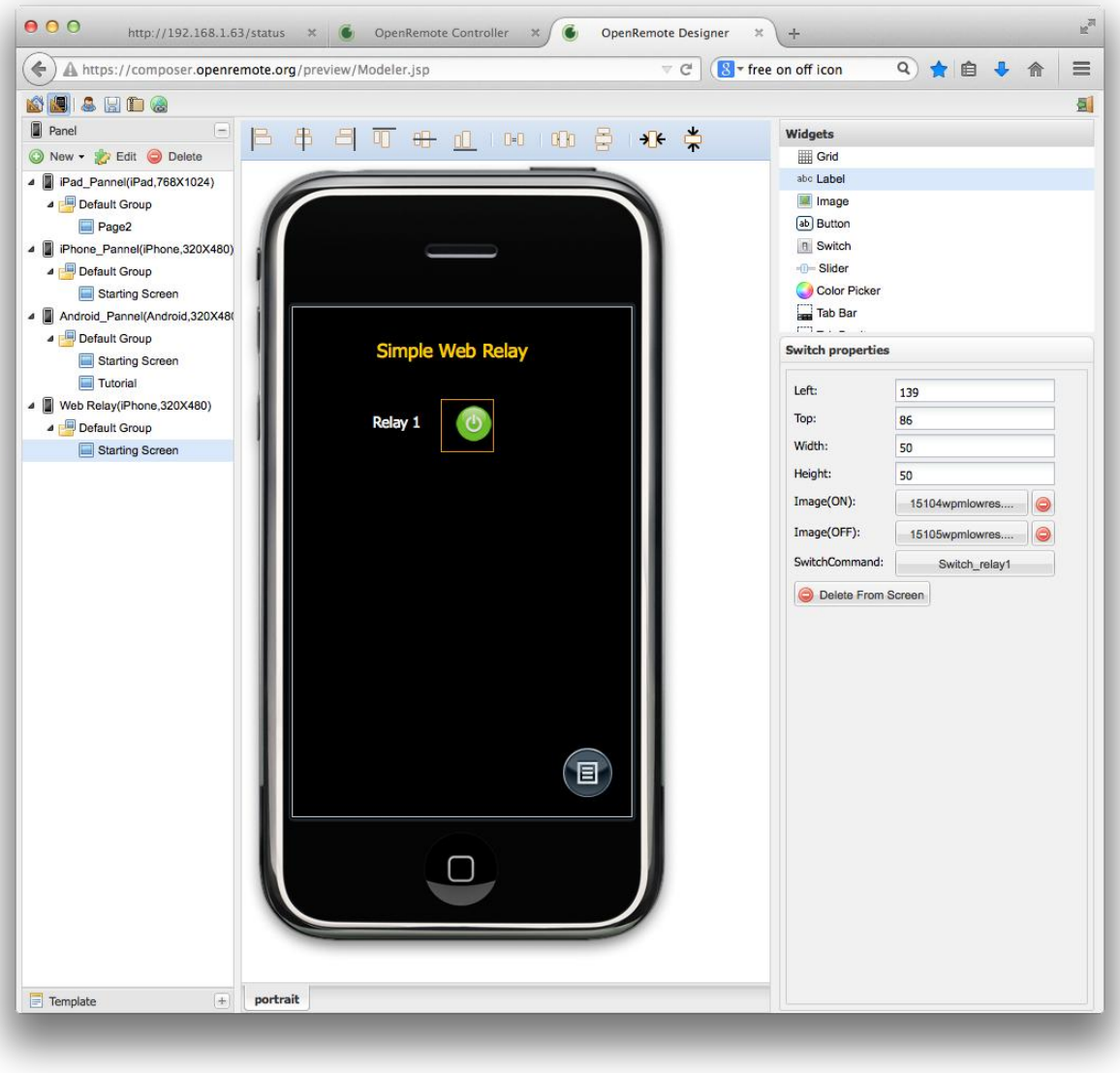
Une fois installé, configurer le « Controller » avec son adresse IP (port 8080) (l'adresse utilisée jusqu'à présent http://192.168.1.61:8080) et sélectionner le nom de panneau (« Pannel Identity ») crée précédemment et appelée « Web Relay ».

Ca y est, deux boutons sur le smartphone permettent maintenant de commander le relais !

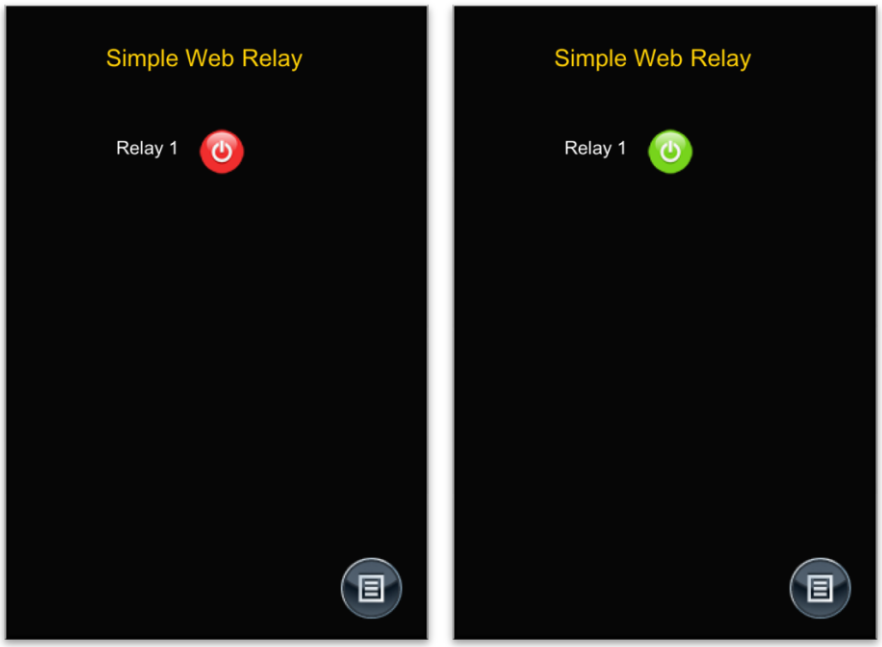


Avant de terminer, changeons ces deux affreux boutons par un simple bouton dont la couleur changera en fonction de l'état du relais. Effacez les deux butons bleu, et remplacez les par un interrupteur (« switch ») dont le paramètre de commande sera «Switch_relay1 » et les deux icones « ON » et « OFF » peuvent être trouvées sur le net facilement : (par exemple les icones trouvées sur http://www.freestockphotos.biz/stockphoto/15104):





Voici le résultat, lorsque le relais est en position « OFF », l'icône est rouge et lorsqu'elle est en mode « ON » elle est verte…



Comme exercice, commander le relais avec une commande http depuis l'extérieur, et observez le changement automatique de couleur de l'interrupteur sur votre smartphone après quelques secondes ! Ou encore, installez OpenRemote sur un deuxième smartphone ou une tablette, et observez l'état du relais synchronisé sur les deux smartphone. Pas mal non ?

Par contre, cette configuration ne fonctionnera pas en dehors de votre réseau domestique. En effet, le « Controller », c-à-dire le Raspberry Pi, n'est pas (encore) accessible depuis l'extérieur. Dans le tutoriel suivant, nous installerons facilement un serveur VPN sur le Pi qui permettra de contrôler le relais avec l'interface OpenRemote depuis l'extérieur.

3. Commander un relais depuis son smartphone à l'aide du logiciel OpenRemote (partie 2)

Déclencher l'arrosage du jardin ou encore l'ouverture d'un volet roulant électrique par l'intermédiaire d'un relais, par simple « clic » sur son smartphone, c'est bien. Mais pourquoi pas depuis loin de chez soi, comme depuis un lieu de vacances ? En suivant ces notes, vous pourrez y arriver en contrôlant le relais à travers un réseau VPN personnel.

Etape 1 : Installation du VPN sur Raspberry Pi
Etape 2 : Test du VPN pour le contrôle du relais

Le matériel utilisé dans ce projet est identique à celui utilisé pour le tutoriel précédent.

Etape 1 : Installation du service VPN sur Raspberry Pi

Pour que logiciel OpenRemote puisse fonctionner sur notre smartphone / tablette depuis n'importe où, il faut installer un accès VPN sur Raspberry et permettre son accès depuis l'extérieur par votre routeur. Aux premiers abords, cela semble compliqué, mais ce ne l'est pas vraiment grâce aux explications que l'on peut trouver sur le net. J'ai choisi le logiciel serveur VPN PPTP (http://Raspberrypihelp.net/tutorials/21-pptp-vpn-server-Raspberry-pi) car il semble être un des plus compatibles avec iOS et/ou Android. Même s'il peut être est un peu moins sécurisé ou performant que d'autre comme OpenVPN, c'est amplement suffisant pour l'application du relais commandé illustré dans ces notes.

Avant d'installer le serveur VPN, il ne devrait pas avoir de message d'erreur après avoir entré l'instruction suivante

```
$ sudo modprobe ppp-compress-18
```

Pour l'installation du serveur VPN, tapez :

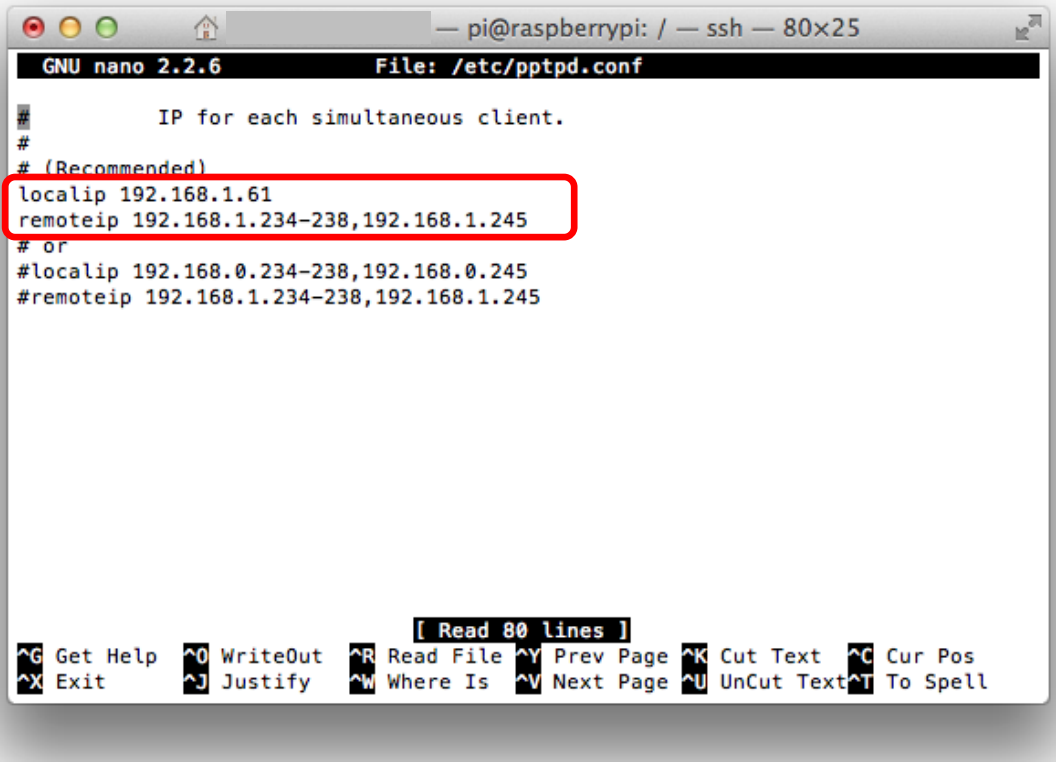
```
$ sudo apt-get install pptpd
```

Et pour configurer le serveur :

```
$ sudo nano /etc/pptpd.conf
```

A la fin du fichier, adaptez les lignes localip et remoteip

```
localip 192.168.1.61
remoteip 192.168.1.234-238,192.168.1.245
```

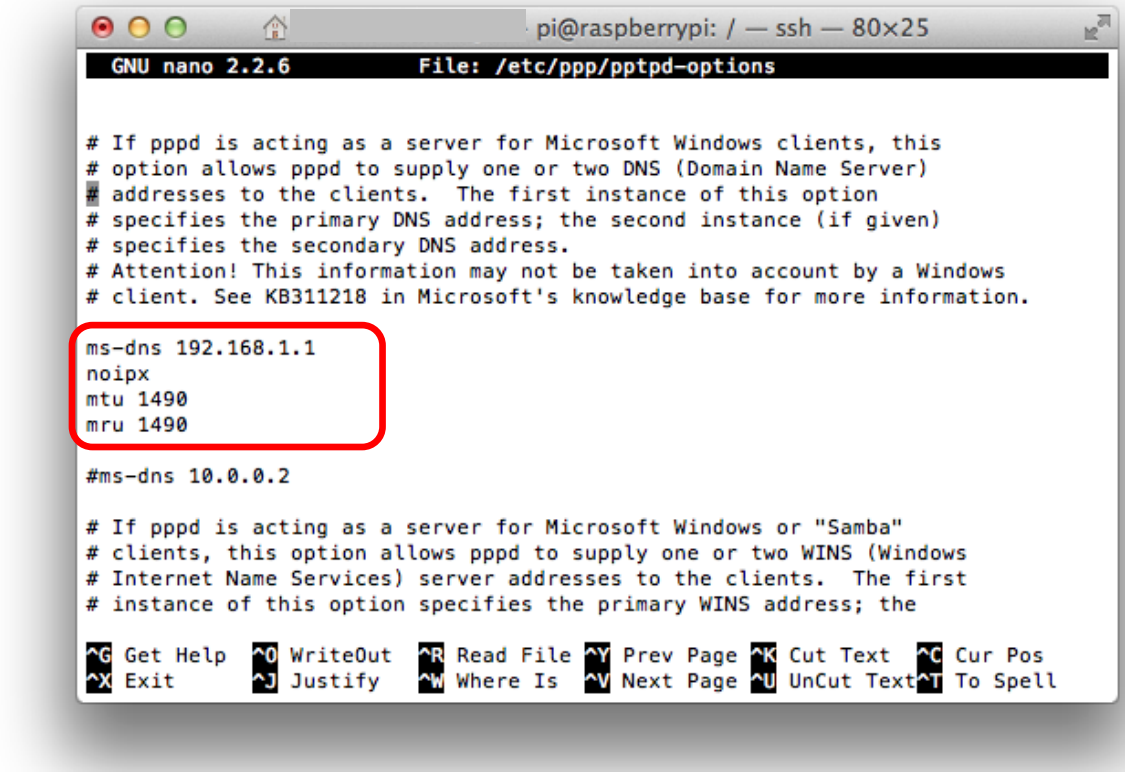


Ensuite ajouter les 4 lignes suivantes à la fin du fichier :

```
ms-dns 192.168.1.1
noipx
mtu 1490
mru 1490
```

Et ensuite la commande :

```
$ sudo nano /etc/ppp/pptpd-options
```

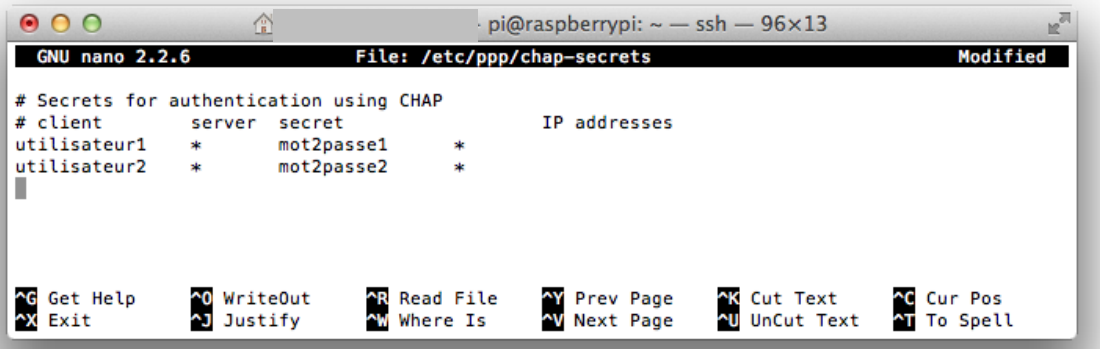


Enfin, créez des utilisateurs avec des mots de passe d'accès avec la syntaxe suivante:

`username[TAB]*[TAB]password[TAB]*`

```
$ sudo nano /etc/ppp/chap-secrets
```

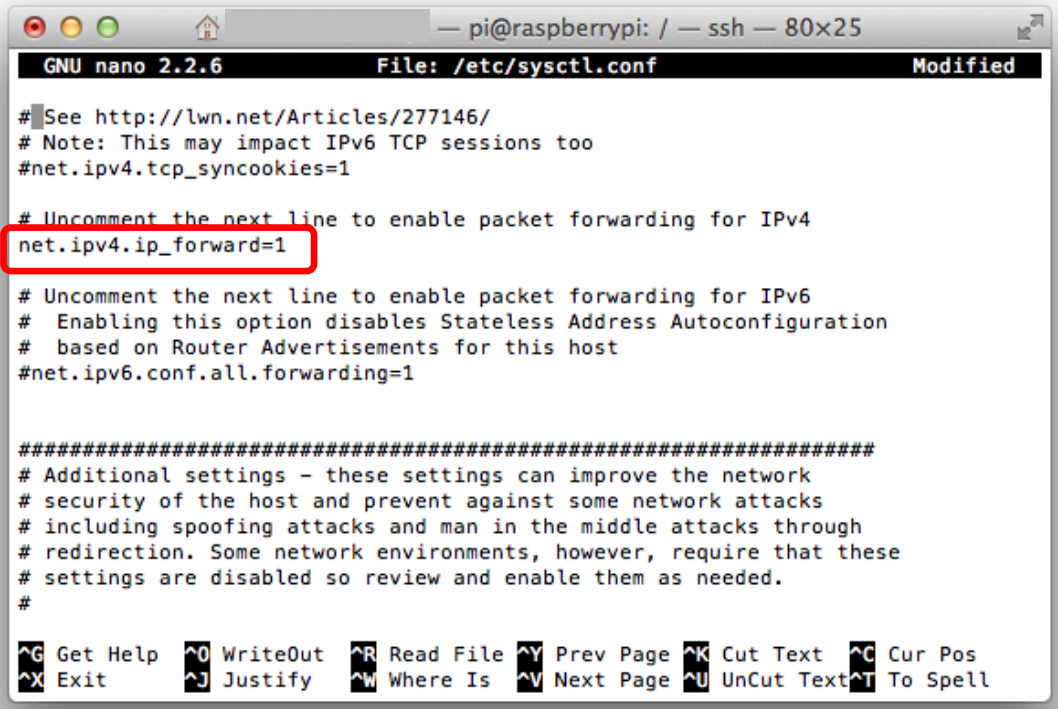
Ici, en exemple le fichier pour deux utilisateurs ainsi que de leur mot de passe respectif :



Ensuite, tapez:

```
$ sudo nano /etc/sysctl.conf
```

Cherchez la ligne « net.ipv4.ip_forward=1” “ et enlevez le « # » (et changez le 0 en 1 si besoin) :



Ces quelques lignes pour terminer :

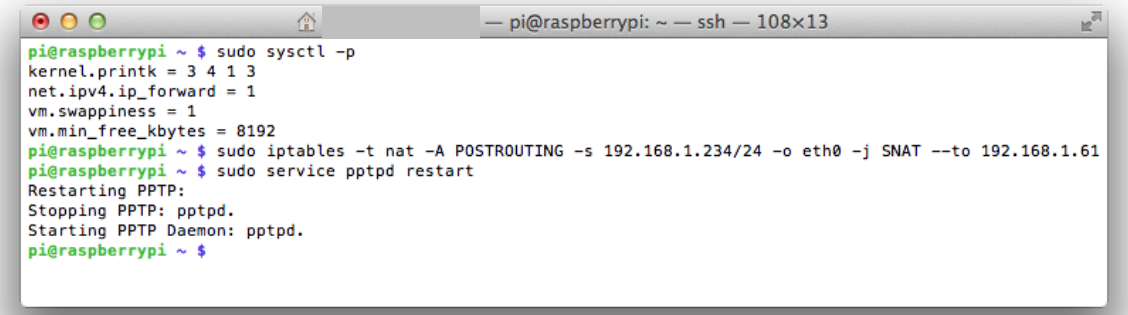
```
$ sudo sysctl -p

$ sudo iptables -t nat -A POSTROUTING -s 192.168.1.234/24 -o eth0 -j SNAT --
to 192.168.1.61
```

Et enfin, pour démarrer le serveur VPN :

```
$ sudo service pptpd restart
```

Et éventuellement pour arrêter le service serveur VPN :

```
$ sudo service pptpd stop
```



Dernière étape, il faut ouvrir un port pour le serveur VPN dans le routeur (port 1723). Ici, la configuration du routeur :

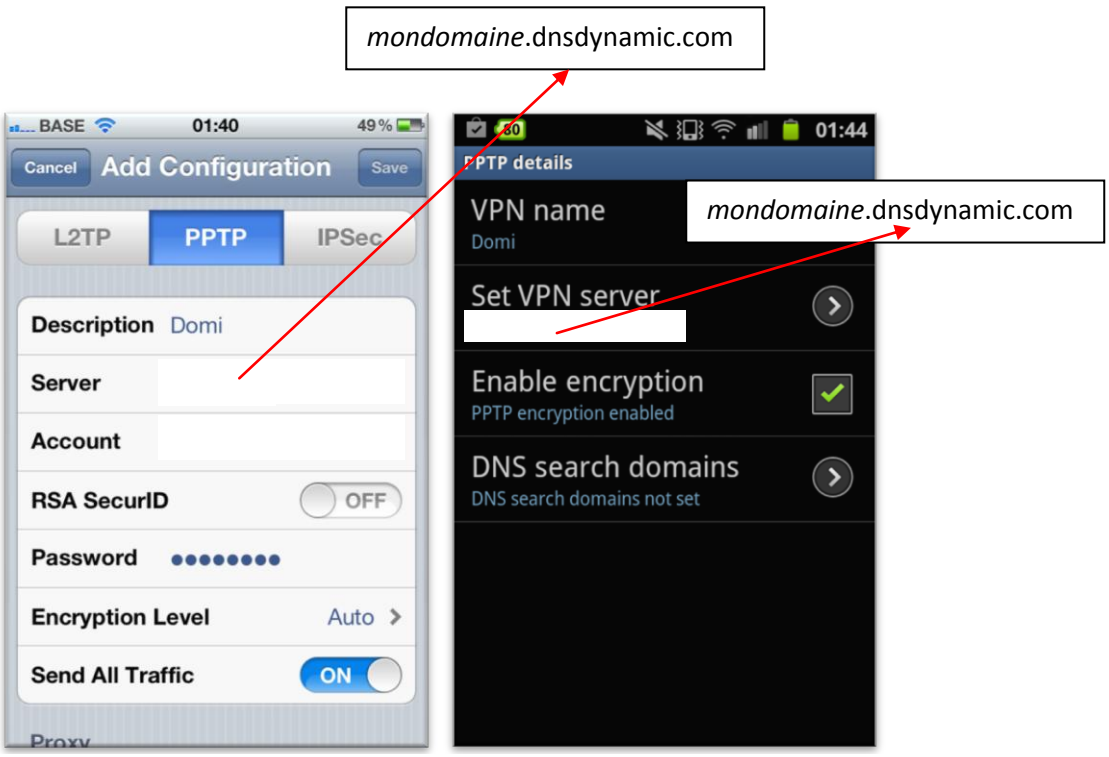
Name	Activated	Protocol	Public start port	Public end port	LAN start port	Local IP Address	Action
VPN	Yes	TCP	1723	1723	1723	192.168.1.61	

Etape 2 : Test du VPN pour le contrôle du relais

PPTP VPN peux être facilement configuré sur Android ou iOS. Voici quelques captures d'écran pour aider sa configuration. Noter le logo VPN dans un des coins supérieurs de l'écran du smartphone lorsque la connexion VPN est établie. Veuillez également noter que la connexion VPN est rompue sur iOS lorsque l'iPhone ou l'iPad bascule automatiquement en mode veille. Les paramètres VPN d'un smartphone Android sont les mêmes.

Pour accéder aux settings VPN d'un iPhone :

```
* Settings -> General-> VPN -> Add VPN Configuration
* Select PPTP
* Description: Give the VPN channel configuration a name
* Server: is the dyndns-address of the router
* Account: is the user name used in the VPN server configuration
* Password is the password used in the VPN server configuration
* RSA-SecureID is OFF
* Encryption Level: Auto
* Send all Traffic ON
* Proxy OFF
```



A partir d'un réseau extérieur, lorsque le VPN est correctement installé (et le service est actif), vous avez accès à votre réseau domestique, en particulier à votre Raspberry Pi (ici à l'adresse IP « domestique » 192.168.1.61) simplement en entrant « *mondomaine*.dnsdynamic.com ».

Une fois la connexion VPN établie, toujours dans un réseau extérieur à votre réseau domestique, en entrant l'adresse de l'Arduino suivi de la commande « relay1_1 » ou « relay1_0 » dans un navigateur Web c-à-dire les commandes suivantes, vous devriez pouvoir contrôler le relais !

```
http://192.168.1.63/relay1_1
```

ou

```
http://192.168.1.63/relay1_0
```

Et pour terminer, si les lignes de commandes http ci-dessus fonctionnent depuis l'extérieur, en gardant la configuration du serveur OpenRemote sur le smartphone ou la tablette avec l'adresse locale utilisée dans l'exemple que nous suivons qui n'est autre que celle de du réseau « domestique », soit :

```
http://192.168.1.61:8080/controller
```

la « télécommande » de l'interface du logiciel OpenRemote pourra commander votre relais depuis la plage ou la montagne (pour autant que vous aillez un réseau téléphonique avec transfert de data dans les parages) !

4. Commander des volets roulants depuis son smartphone

Introduction

Le but de ce projet est de commander des volets roulants équipés de télécommandes sans fils à partir de l'interface du logiciel OpenRemote.

Dans mon cas précis, j'ai voulu contrôler mes volets roulants de marque Somfy avec une interface domotique sympa. Des solutions commerciales existent (par exemple l'utilisation d'interface domotique tels que Somfy Box Tahoma , HomeWizard ou encore la Zibase), mais elles sont relativement fermées au développement. Récemment un récepteur/transmetteur USB compatible Raspberry Pi et protocole Somfy RTS a vu le jour (voir le RFXtrx433E USB 433.92MHz Transceiver).

Si la soudure ne vous tente pas, en combinant ce dernier récepteur/transmetteur avec le logiciel open source Domoticz fonctionnant avec le Raspberry Pi vous trouverez une solution domotique vraiment bien conçue. Si sous continuez à lire ces lignes, vous avez le virus comme moi du « je peux le faire moi-même car c'est tellement plus amusant » ….

Comme beaucoup d'autres, j'ai opté pour une solution ou les télécommandes sont reliées physiquement avec un Arduino car le protocole d'échange radio fréquence est protégé avec un code roulant, même si certains semblent avoir interprété et compris ce code RTS.

Côté hardware, je n'ai rien inventé, mais je me suis plutôt inspiré des solutions développées sur le net :

- Building a Somfy Controller with Arduino and Vera
- DiY Somfy Blind Control
- Arduino et Télécommande Somfi Telis 4 RTS (volets roulants)
- DIY : Réaliser son interface « bidirectionnelle » ZwaveSomfy
- Control Somfy RTS with Openremote
- How To - Control Sunshades - Somfy-Telis with Raspberry Pi
- Shades Automation - howto

Dans les quatre premiers exemples, les auteurs raccordent une télécommande Somfy à un microcontrôleur de type Arduino, et dans le 3 derniers, à un Raspberry Pi. Dans tous les cas, la solution est l'assemblage d'une interface (un « shield ») faite maison basée sur un petit composant sympa, l'optocoupleur CNY74-4 et sur quelques télécommandes Somfy Telis trouvées de seconde main qui ne demandaient qu'à être désossées…

Par contre côté code Arduino, je n'ai pas trouvé ce que je cherchais, donc j'ai appris un peu sa programmation « sur le tas ». Je voulais une sorte de retour de position et une commande non pas du type « haut », « bas » ou « stop » comme sur les télécommandes Telis, mais bien une commande du type « Volet 1 : 60% ouvert » ou « volet 5 : 100% ouvert ».

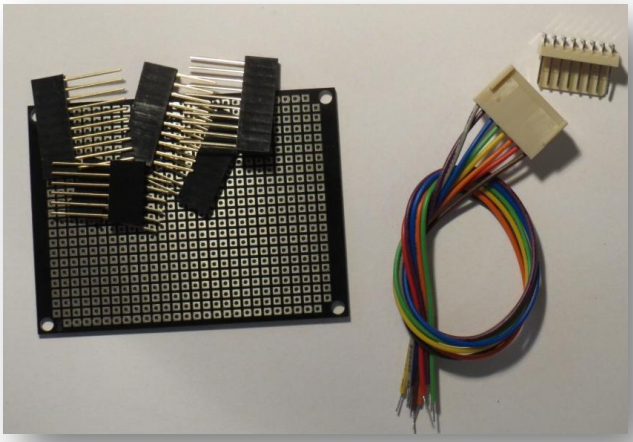
Etape 1 : Création d'un shield Arduino d'interface avec les télécommandes

Etape 2 : Programmation Arduino

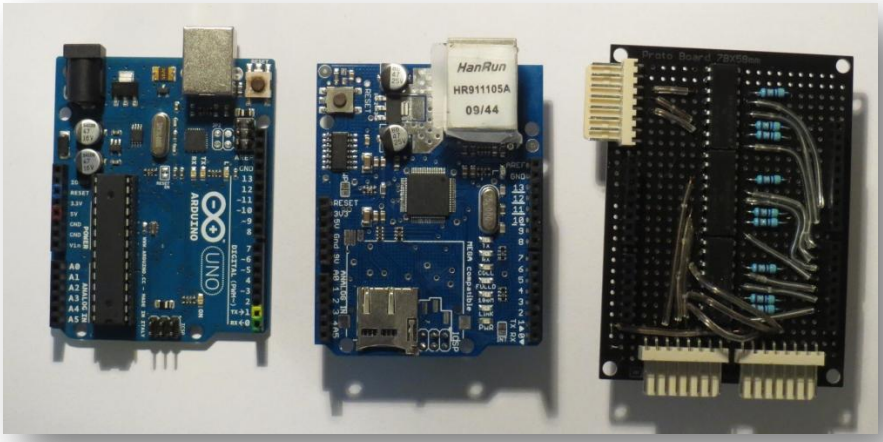
Etape 3 : Installation d'un "slider" qui permet de définir la hauteur des volets roulants

Le matériel utilisé dans ce projet est encore identique à celui utilisé pour le tutoriel précédent, et le « shield » fait maison est composé des éléments suivants :

1 optocoupleur de type CNY74-4 (par volet roulant)
3 résistance 220 Ohms (par volet roulant)
1 connecteur 3 fils (ou 5 si on in inclus l'alimentation de la télécommande) - mâle (par volet roulant)
1 télécommande Somfy Telis 1 canal (par volet roulant)
1 plaque de prototypage muni de connecteurs, le tout compatible Arduino Uno



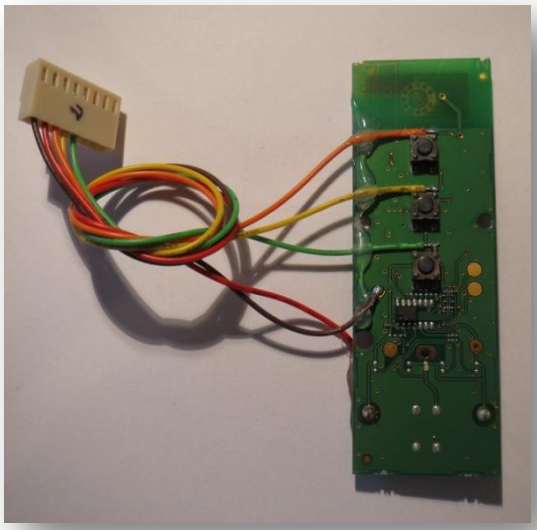
Dans les illustrations qui suivent, je commande 3 volets roulants.

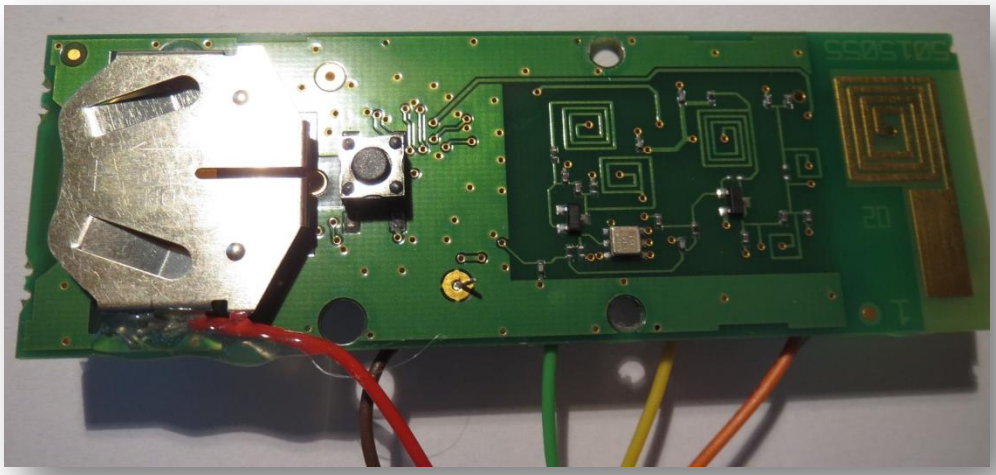


Etape 1 : Création d'un shield Arduino d'interface avec les télécommandes

Les télécommandes Somfy Telis ont 3 boutons poussoirs (haut, bas et MySomfy), et lorsqu'un de ce ceux-ci est poussé, met en contact le circuit à la masse (GND, 0 V). Il suffit donc de souder un fil sur la borne de chaque contacteur qui n'est pas reliée à la masse (borne à droite sur les illustrations qui suivent).

Ensuite, soudez un fil pour la masse (brun) et un pour le 3.0 V (rouge). Notez l'absence de la pile car la télécommande sera simplement alimentée par la carte Arduino (3.3 V).

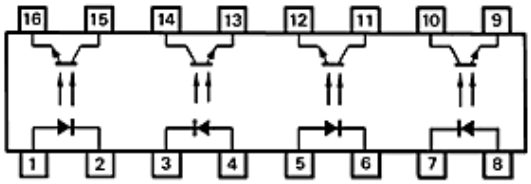




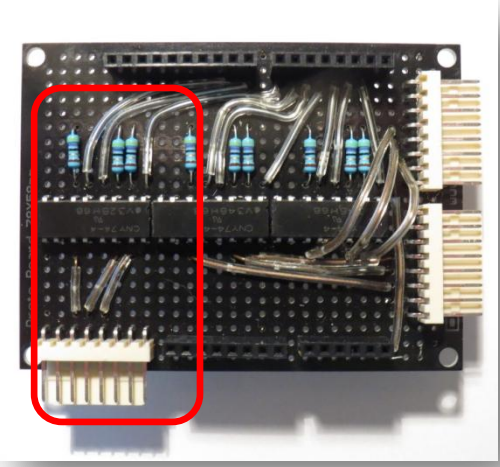
Dans le montage ci-dessus, j'utilise des fiches mâles-femelles bien pratiques, surtout si vous avez quelques volets à contrôler et de la colle chaude pour fixer les fils plus fermement.

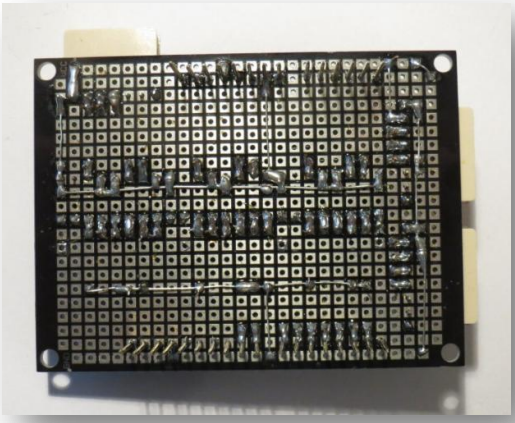
Le « shield » qui doit porter la mise à la masse des contacteurs de la télécommande pourrait être basé sur des relais ou des transistors de type 2n2222, mais dans le cas présent l'utilisation de l'optocoupleur CNY74-4 est assez élégante et ne prends pas beaucoup de place sur la carte.

Le raccordement de l'optocoupleur CNY74-4 est assez facile. La partie « commande », c-à-dire les diodes, sont reliées aux différentes sorties de l'Arduino. Dans mon cas, les pins 7 à 9 de l'Arduino (pour le premier volet roulant) sont reliés respectivement aux bornes 1, 4 et 5 de l'optocoupleur. Les bornes 2, 3, 6 de celui-ci sont quant à elles reliées individuellement à la masse par l'intermédiare d'une résistance 220 Ohms. Comme les télécommandes utilisées ici n'ont que 3 boutons, je n'utilise que 3 des 4 coupleurs (donc pas les dernières bornes 7, 8, 9 et 10).

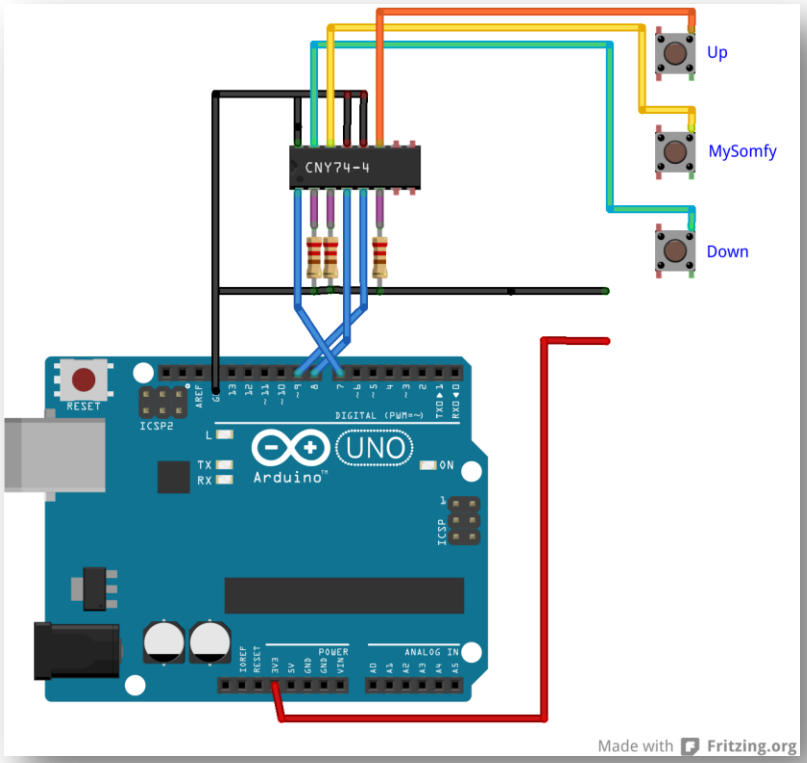


l'optocoupleur CNY74-4

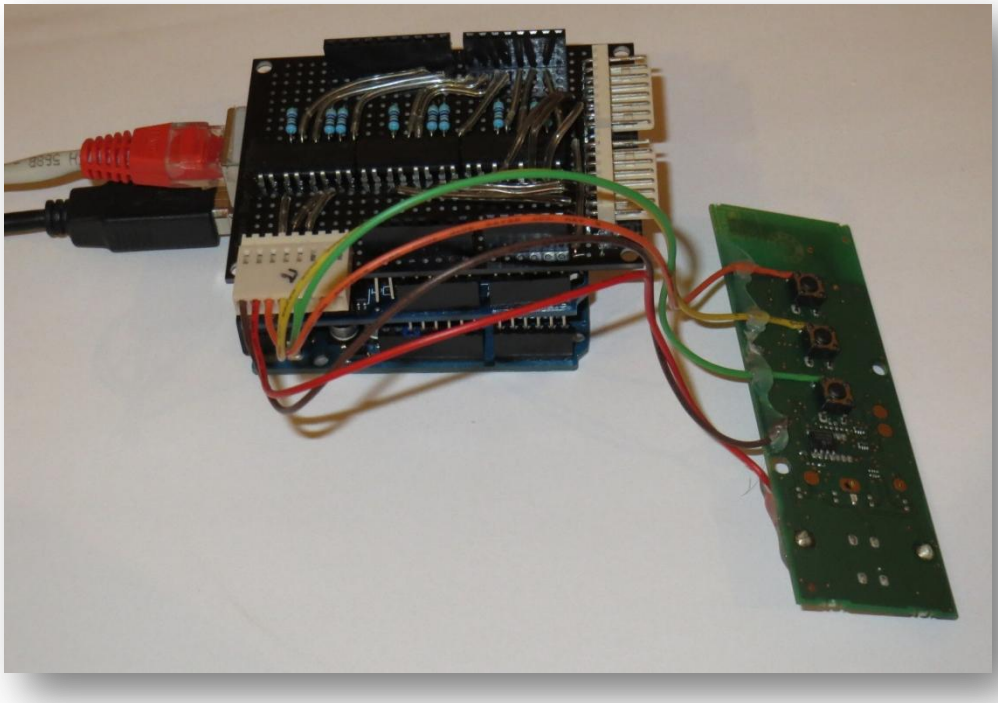




La partie « commandée » de l'optocoupleur est bien entendu reliée à la télécommande. Les bornes 15, 14 et 11 seront respectivement reliées au bouton « bas » (fil vert), bouton « MySomfy » (fil jaune) et bouton « haut » (fil orange). Enfin, les bornes 16, 13 et 12 vont directement à la masse.



Dans mon montage, ce raccordement est répété 3 fois (pour 3 télécommandes).



Etape 2 : Programmation Arduino

Ici encore, le code Arduino pour ce projet concernant la partie Ethernet est amplement inspiré de celui développé par Dominique Meurisse pour l'installation d'un web serveur mesurant la température, la lumière ou la présence (http://Arduino103.blogspot.be/2014_04_01_archive.html).

J'ai tout d'abord adapté le code au contrôle de l'optocoupleur, avec les actions « *press down* », «*press up* » ou « press *stop* » (ou « press MySomfy »).

Dans l'exemple qui suit, le contrôle de la hauteur du volet roulant est effectué par la mesure du temps écoulé entre une action « *press down* » ou «*press up* » et ou « press *stop* ». Les ordres par commande http sont du type http://192.168.1.63/*volet_position*, ou *volet* est un nombre de 1 à 10 pour identifier chaque volet, et *position* un nombre de 0 à 10 (0 correspondant à la *position* complètement fermée et 10 à la position complètement relevée). Chaque valeur entre 0 et 10 correspondra à la position intermédiaire du volet roulant, par pas de 10 %. En d'autres termes, une valeur de *position* de 3 correspondra au volet monté à 30 %, une valeur de *position* de 7 correspondra au volet monté à 70 %, etc.

Pour réaliser cela, j'ai dû chronométrer la fermeture complète du volet. Dans mon cas, 28 secondes sont nécessaires. Donc pour atteindre une ouverture de 40 % à partir d'une position initiale de 10 % ouverte, il faudra laisser 8.4 secondes entre les commandes «*press up* » et « *press stop* » (il faudra monter les volets de (40%-10%) x 28 s, soit 8.4 s). Le programme devra calculer cet intervalle de temps en fonctions de la position initiale et déterminer s'il faut monter ou descendre. Enfin, à chaque montée ou descente totale, nous ne commanderons pas le bouton « press *stop* » pour que la position du volet puisse se « recalibrer » facilement en position 0 ou 10.

Vous l'avez compris, ce montage ne permet pas le retour de la position « physique » du volet, mais retournera un fichier XML contenant la position « théorique » du volet. S'il y a décalage entre les valeurs retournées et réelles, il suffira de monter ou descendre complètement le volet.

La position retournée par le microcontrôleur Arduino par l'intermédiaire du fichier XML sera interprétée par le logiciel OpenRemote et ensuite retournée à l'interface utilisateur pour afficher la position du (des) volet(s) sur tous les smartphones ou tablettes reliées à OpenRemote.

Dans l'exemple suivant, le shield Ethernet est configuré avec une adresse IP 192.168.1.63, les pins 7 à 9 sont ceux qui commandent le premier optocoupleur.

Le code Arduino Uno est le suivant :

```
//
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//                                                       DIY Somfy Control
//                                                    D. Swinnen - July 2014
//
```

```
// This example illustrates how to control a simple relay by http command
// The relay is connected on PIN 2 of the Arduino Uno board
// The http command from a web browser as follow: "192.168.1.63/1_2" to command screen
1 at position 20% opened
//
// This code and application to control a relay are inspired and adapted from the ones
described bx D. Meurisse
// http://Arduino103.blogspot.be/2014_04_01_archive.html
// http://mchobby.be/data-files/pi-vigilance/WebServer_BasicNode-v01b.ino
//
//
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#include <SPI.h>
#include <Ethernet.h>

//--- Outils de dÃ©bogage ---------------------------------------
#define GG_DEBUG

#ifdef GG_DEBUG
  #define DEBUG_PRINTLN(x) Serial.println(x)
  #define DEBUG_PRINTLN2(x,y) Serial.println(x,y)
  #define DEBUG_PRINT(x) Serial.print(x)
  #define DEBUG_PRINT2(x,y) Serial.print(x,y)
#else
  #define DEBUG_PRINTLN(x)
  #define DEBUG_PRINTLN2(x,y)
  #define DEBUG_PRINT(x)
  #define DEBUG_PRINT2(x,y)
#endif

//-- Identification du Noeud ----------------------------------
// Cette information est retournee dans les differentes reponses
//    WEB. Elles peuvent etre utile pour identifier la fonctionalitÃ©
//    du Noeud
#define NODE_ID "DIY_Somfy_Control"

//--- Definition des erreurs HTTP --------------------------------
// 400 Bad Request  -
 The request could not be understood by the server due to malformed syntax
// 401 Unauthorized - The request requires user authentication.
// 403 Forbidden    -
 The server understood the request, but is refusing to fulfill it.
//                    Authorization will not help and the request SHOULD NOT be repeat
ed.
// 404 Not Found    - The server has not found anything matching the Request-URI.
// 405 Method Not Allowed - The method specified in the Request-
Line is not allowed for the resource identified by the Request-URI.
const int HTTP_ERR_BadRequest       = 400;
const int HTTP_ERR_Unauthorized     = 401;
const int HTTP_ERR_Forbidden        = 403;
const int HTTP_ERR_NotFound         = 404;
const int HTTP_ERR_MethodNotAllowed = 405;
const int HTTP_ERR_ServerError      = 500;
```

```
//"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"""""""""""""""""""""
//"""""""""""""""""""""""""""""""""""""""" Here put the number of screen """""""""""""
"""""""""""""""""""""
//"""""""""""""""""""""""""""""""""""""""" Here you map the pin layout, """"""""""""""
"""""""""""""""""""""
//""""""""""""""" set the duration for each screen to close from the fully open positi
on (in ms) """"""""""
//"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"""""""""""""""""""""

const int NumberOfScreens = 1;
// const int NumberOfScreens = 3;

const int PinDown[NumberOfScreens] = {7};
const int PinHold[NumberOfScreens] = {8};
const int PinUp[NumberOfScreens]   = {9};
//const int PinUp[NumberOfScreens] = {29, 33, 26};
//const int PinHold[NumberOfScreens] = {28, 32, 27};
//const int PinDown[NumberOfScreens] = {30, 31, 24};

// 10 as the position for fully open
int TargetPosition[NumberOfScreens]={10};
int InitialPosition[NumberOfScreens]={10};
int CurrentPosition[NumberOfScreens]={10};
//int TargetPosition[NumberOfScreens]={10, 10, 10};
//int InitialPosition[NumberOfScreens]={10, 10, 10};
//int CurrentPosition[NumberOfScreens]={10, 10, 10};

const long TimeToClose[NumberOfScreens] = {28000};
//const long TimeToClose[NumberOfScreens] = {28000, 28000, 38000};

int CurrentPositionNEW[NumberOfScreens]={10};
int OldPositionNEW[NumberOfScreens]={10};
int NumOrderNEW[NumberOfScreens]={10};
int CommandTerminated[NumberOfScreens]={1};
//int CurrentPositionNEW[NumberOfScreens]={10, 10, 10};
//int OldPositionNEW[NumberOfScreens]={10, 10, 10};
//int NumOrderNEW[NumberOfScreens]={10, 10, 10};
//int CommandTerminated[NumberOfScreens]={1, 1, 1};

//"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"""""""""""""""""""""
//"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"""""""""""""""""""""
//"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"""""""""""""""""""""
//"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"""""""""""""""""""""

String requestAction;
int ScreenID;
```

```
String Order;
int DeltaPosition[NumberOfScreens];
int Status[NumberOfScreens];
long DeltaTime[NumberOfScreens];
String Direction[NumberOfScreens];
int CommandsPressed;

unsigned long TimeWhenAnyCommandsPressed;
unsigned long TimeNow;
unsigned long TimeCommandsPressed[NumberOfScreens];


//--- Request Methods ------------------------------------------
const byte REQUEST_METHOD_NONE = 0;
const byte REQUEST_METHOD_GET = 1; // Post & Head are ignored

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1,63);
EthernetServer server(80);

// Buffer pour la parsing de la Ligne courante
//   de la requete HTTP (on traite une ligne a la fois)
String currentLine;
#define CURRENT_LINE_BUFF_SIZE 128

// Nom de l'action dans l'URL http://192.168.1.177/monAction?param1=1&param2=x


#define REQUEST_ACTION_BUFF_SIZE 20



//--- Setup() ----------------------------------------------------
//
//----------------------------------------------------------------
void setup() {
////  pinMode(1, OUTPUT); //pin selected to control
//  pinMode(2, OUTPUT); //pin selected to control
//  pinMode(3, OUTPUT); //pin selected to control
//  pinMode(4, OUTPUT); //pin selected to control
//  pinMode(5, OUTPUT); //pin selected to control
//  pinMode(6, OUTPUT); //pin selected to control
  pinMode(7, OUTPUT); //pin selected to control
  pinMode(8, OUTPUT); //pin selected to control
  pinMode(9, OUTPUT); //pin selected to control

//  pinMode(22, OUTPUT); //pin selected to control
//  pinMode(23, OUTPUT); //pin selected to control
//  pinMode(24, OUTPUT); //pin selected to control
//  pinMode(25, OUTPUT); //pin selected to control
//  pinMode(26, OUTPUT); //pin selected to control
```

```
//  pinMode(27, OUTPUT); //pin selected to control
//  pinMode(28, OUTPUT); //pin selected to control
//  pinMode(29, OUTPUT); //pin selected to control
//  pinMode(30, OUTPUT); //pin selected to control
//  pinMode(31, OUTPUT); //pin selected to control
//  pinMode(32, OUTPUT); //pin selected to control
//  pinMode(33, OUTPUT); //pin selected to control

  Order="";


  Serial.begin(9600);
  //*DEBUG_PRINTLN( "Setuping...." );

  // RÃ©server la taille du buffer
  currentLine.reserve( CURRENT_LINE_BUFF_SIZE+1 ); // 128 caracteres + NULL
  requestAction.reserve( REQUEST_ACTION_BUFF_SIZE+1 ); // 6 caractere max + NULL

  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();

  //*DEBUG_PRINT("server is at ");
  //*DEBUG_PRINTLN(Ethernet.localIP());

}

void loop() {
  //Serial.print("Lecture Analogique = ");
  //Serial.println(photocellReading);     // La valeur analogique brute
  //-- Gestion des connexion entrantes --
  manageRequests();
  manageActions();
}

void manageRequests() {
  byte charCount;
  byte requestMethod = REQUEST_METHOD_NONE;


  // Ecouter les connexions entrantes
  EthernetClient client = server.available();
  if (client) {
    //*DEBUG_PRINTLN("new client");
    // Une requete HTTP fini avec une ligne blanche... il faut donc la detecter
    boolean currentLineIsBlank = true;
    // SI la rÃ©ponse est une erreur (qui peut etre detectee dÃ¨s le parsing du
Header)
    // ALORS il ne faut pas essayer de traiter la requete HTTP
    boolean skipProcessing = false;

    currentLine = "";
    charCount   = 0;
    requestMethod = REQUEST_METHOD_NONE;
```

```
    requestAction = "";

    while (client.connected()  && !(skipProcessing) ){
      if (client.available()) {
        char c = client.read();

        // Pour les curieux
        //Serial.print( '.' );
        //Serial.print( c );

        // Recomposer une String (jusqu'a  la taille du buffer)
        if( charCount < CURRENT_LINE_BUFF_SIZE ) {
          currentLine += c;
          charCount++;
        }

        // SI nous arrivons Ã  la fin d'une ligne (en recevant un caractere de fin
        // de ligne) et que la ligne est est vide (blank), la reception de requete
http
        // est terminee
        // ALORS nous pouvons traiter la demande (sauf erreur deja  detectee) :-)
        if (c == '\n' && currentLineIsBlank && !skipProcessing) {

          // Traitement de la requete client (Client Request)
          processRequest( client, requestMethod, requestAction );

          break;
        }

        //--- INSPECTION LIGNE DE LA REQUETE ------------------------------
        // SI on reÃ§oit un caractÃ¨re de fin de ligne dans le flux de la requete
        // ALORS on inspecte le contenu de la ligne
        //    But: Extraire les infos utiles au traitement de la requete
        if (c == '\n') {
          // DEBUG
          //*DEBUG_PRINT( "Inspect: " );
          //*DEBUG_PRINT( currentLine ); //la ligne contient deja un CR/LF
          //*DEBUG_PRINTLN(" ");
          // Detection de la mÃ©thode GET/POST/... sur la ligne
          //    GET /setparam?id=12 HTTP/1.1

          //   Seule la methode GET est detectee (les autres sont ignorees pour le
moment)
          if( currentLine.indexOf( " HTTP/1.") >= 0 ){
             //*DEBUG_PRINTLN( " +-> Method Detection" );
             if( currentLine.indexOf( "GET " )==0 ){
               requestMethod = REQUEST_METHOD_GET;
             }

             //*DEBUG_PRINTLN( " +-> Query Parsing" );
             int iSlashPos = currentLine.indexOf( "/" );
             int iEndOfQuery = currentLine.indexOf( " ", iSlashPos+1 );
             int iQuestionMark = currentLine.indexOf( "?", iSlashPos+1 );
             // DEBUG_PRINTLN( iSlashPos );
```

```
            // DEBUG_PRINTLN( iEndOfQuery );
            // DEBUG_PRINTLN( iQuestionMark );
            if( iQuestionMark >= 0 ){
              // Requete du type
              //   http://192.168.1.177/read?ID=16
              //   http://192.168.1.177/?ID=16
              // Verif longueur maximal de requestAction
              if( (iQuestionMark-(iSlashPos+1)) > REQUEST_ACTION_BUFF_SIZE ){
                sendError( client, HTTP_ERR_ServerError, "Action name exceed max
length" );
                //*DEBUG_PRINTLN( " +-> skip processing");
                skipProcessing = true;
                break;
              }
              // Copier ne nom de l'action
              requestAction = currentLine.substring( iSlashPos+1, iQuestionMark );
            }
            else {
              // Requete du type
              //   http://192.168.1.177/read
              //   http://192.168.1.177
              // Verif longueur maximal de requestAction
              if( (iEndOfQuery-(iSlashPos+1)) > REQUEST_ACTION_BUFF_SIZE ){
                sendError( client, HTTP_ERR_ServerError, "Action name exceed max
length" );
                //*DEBUG_PRINTLN( " +-> skip processing");
                skipProcessing = true;
                break;
              }
              // Copier ne nom de l'action
              requestAction = currentLine.substring( iSlashPos+1, iEndOfQuery );

            }

            //&&&&&&&&&&&&&&&&&&&
            //&&&&&&&&&&&&&&&&&&&
            ScreenID=(requestAction.substring(0,1)).toInt();
            Order=requestAction.substring(2);
            //&&&&&&&&&&&&&&&&&&&
            //&&&&&&&&&&&&&&&&&&&


            //*DEBUG_PRINTLN( " +-> Query checks" );
            // Il doit y avoir une action
            if( requestAction.length() == 0 ){
              sendError( client, HTTP_ERR_ServerError, "Action missing!" );
              //*DEBUG_PRINTLN( " +-> skip processing");
              skipProcessing = true;
              break;
            }

          }

          // Reinit du buffer pour la ligne suivante
```

```
          currentLine = "";
          charCount = 0;
          // Jusqu'a preuve du contraire, la prochaine ligne est vide
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          // SI on reÃ§oit un caractÃ¨re...
          // ALORS la ligne n'est pas vide.
          //
          // NB: Ignorer le caractÃ¨re de Line Feed (\r qui est souvent juste devant
le retour chariot \n)
          currentLineIsBlank = false;
        }

      }

    }

    // Laisser le temps au web browser pour recevoir les donnees
    delay(1);
    // Fermer la connexion:
    client.stop();
    //*DEBUG_PRINTLN("client disonnected");
  }
}

// Description:
//   Renvoi un header HTML avec le code d'erreur PUIS clos la
//   connexion ethernet
// Parameter:
//   HtmlErrorCode - Code d'erreur HTML
//   sInfo - information complÃ©mentaire pour la page d'erreur
//
void sendError( EthernetClient client, int HtmlErrorCode, String sInfo){
    char c;
    //*DEBUG_PRINTLN( "sendError: flush input stream" );

    // Jeter le reste de la Requete HTTP Ã  la poubelle
    while( client.available() ){
      c = client.read();
      // Pour les curieux...
      //DEBUG_PRINT( '.' );
      //DEBUG_PRINT( c );
    }
    // Donner du temps au browser pour recevoir la reponse
    delay( 1 );

    DEBUG_PRINTLN( "sendError: sending error response" );
    // Envoyer un HEADER http
    client.println("HTTP/1.1 ");
    client.print( HtmlErrorCode );
    client.println(" ERROR");
    client.println("Content-Type: text/html");
    client.println("Connection: close");
```

```
    client.println();
    // Envoyer le contenu HTML
    client.println("<!DOCTYPE HTML>");
    client.println("<html>");
    client.print( "NODE_ID (" );
    client.print( NODE_ID );
    client.println( ")<br />" );
    client.print("HTML ERROR CODE (");
    client.print( HtmlErrorCode );
    client.println(")<br />");
    client.print("Info: ");
    client.println( sInfo );
    client.println("</html>");

    DEBUG_PRINTLN("sendError: sent!");
}


//------------------------------------------------------------------------
// processRequest
//------------------------------------------------------------------------
// Description:
//   Fait un traitement de la requete du client.
//   C'est ici qu'il faut ajouter les nouvelles entree.
//
// Note:
//   Si on arrive dans ce code, l'URL est verifiee et correctement formatee
//
void processRequest( EthernetClient client, byte RequestMethod, String RequestAction
){
    if( RequestMethod != REQUEST_METHOD_GET ){
      sendError( client, HTTP_ERR_MethodNotAllowed, "Method Not Allowed" );
      return;
    }
    // dÃ©tection de l'action et du traitement Ã  faire.

    //%%%%%%%%%%%%%%%%%%%%%%%%% STATUS %%%%%%%%%%%%%%%%%%%%%%%%%

     if(requestAction=="status"){

      client.println("HTTP/1.1 200 OK");
      client.println("Content-Type: text/xml"); //retourne du XML
      client.println("Connection: close");
      client.println();
      client.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");

      client.print("<node id=\""); client.print("DIY_Somfy_Box"); client.println("\">"
);

      for (int c=1; c < NumberOfScreens+1; c++){
        client.print ("<CurrentPosition"); client.print (c);client.print (">");
client.print(CurrentPosition[c-1]);
        client.print ("</CurrentPosition"); client.print (c);client.print (">");
      }
```

```
      client.print("<TimeSinceRollover>"); client.print (int(millis()/86400000));
client.print("</TimeSinceRollover>");

      client.println("</node>");
    }
    //%%%%%%%%%%%%%%%%%%%%%%%%% STATUS %%%%%%%%%%%%%%%%%%%%%%%%%

    //%%%%%%%%%%%%%%%%%%%%%%%%% RESET %%%%%%%%%%%%%%%%%%%%%%%%%

     if(requestAction=="reset"){

      client.println("HTTP/1.1 200 OK");
      client.println("Content-Type: text/xml"); //retourne du XML
      client.println("Connection: close");
      client.println();
      client.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");

      client.print("<node id=\""); client.print("DIY_Somfy_Box"); client.println("\">"
);

      client.print("<Reset>"); client.print ("Reset"); client.print("</Reset>");

      client.println("</node>");

      for (int c=0; c < NumberOfScreens; c++){
       digitalWrite(PinUp[c], HIGH); delay(200); digitalWrite(PinUp[c], LOW);
delay(200);
      }

    }
    //%%%%%%%%%%%%%%%%%%%%%%%%% RESET %%%%%%%%%%%%%%%%%%%%%%%%%



    if(Order=="$%7Bparam%7D"){
     //digitalWrite(2, HIGH);    // set pin 4 high
     //Serial.println("stop");
     //delay(CurrentPosition * 100);
     //digitalWrite(2, LOW);    // set pin 4 high

      client.println("HTTP/1.1 200 OK");
      client.println("Content-Type: text/xml"); //retourne du XML
      client.println("Connection: close");
      client.println();
      client.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");

      client.print("<node id=\""); client.print("DIY_Somfy_Box"); client.println("\">"
);

      for (int c=1; c < NumberOfScreens+1; c++){
        client.print ("<CurrentPosition"); client.print (c);client.print (">");
        client.print(CurrentPosition[c-1]);
        client.print ("</CurrentPosition"); client.print (c);client.print (">");
```

```
      }

      client.println("</node>");
    }


    else {
    for (int i=0; i < NumberOfScreens; i++){

        if(ScreenID==i+1 && Status[i]==0){
              TargetPosition[i]=Order.toInt();
              DeltaPosition[i]=InitialPosition[i] - TargetPosition[i];
              DeltaTime[i]=DeltaPosition[i]*TimeToClose[i]/10;
              TimeWhenAnyCommandsPressed=millis();
              CommandsPressed=1;
              Status[i]=1;
              if (DeltaPosition[i]>0){Direction[i]="Down";}
              else if (DeltaPosition[i]<0){Direction[i]="Up";}
            }
    }

    }
}


void manageActions() {

  TimeNow=millis();
  if(TimeNow-TimeWhenAnyCommandsPressed>2000UL){

    if (CommandsPressed==1){
      DEBUG_PRINT ( "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%" );DEBUG_PRINTLN(" ");
      for (int i=0; i < NumberOfScreens; i++){
        DEBUG_PRINT ("i: "); DEBUG_PRINT (i); DEBUG_PRINT ("...ScreenID");DEBUG_PRINT
(i+1);
        DEBUG_PRINT ("..."); DEBUG_PRINT ("MOVE: ");DEBUG_PRINT
(InitialPosition[i]);DEBUG_PRINT ("-->"); DEBUG_PRINT (TargetPosition[i]);
        DEBUG_PRINT ("...DeltaPosition: ");DEBUG_PRINT(DeltaPosition[i]); DEBUG_PRINT
("...DetaTime: ");DEBUG_PRINT (DeltaTime[i]);DEBUG_PRINT (" ms...");
        DEBUG_PRINT ("Status: "); DEBUG_PRINT (Status[i]);DEBUG_PRINTLN ("");
        }
     DEBUG_PRINT ( "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%" );DEBUG_PRINTLN ("");

    CommandsPressed=0;
    }


    for (int i=0; i < NumberOfScreens; i++){
      if (Status[i]==1){
        if (Direction[i]=="Down"){
          TimeCommandsPressed[i]=millis();
```

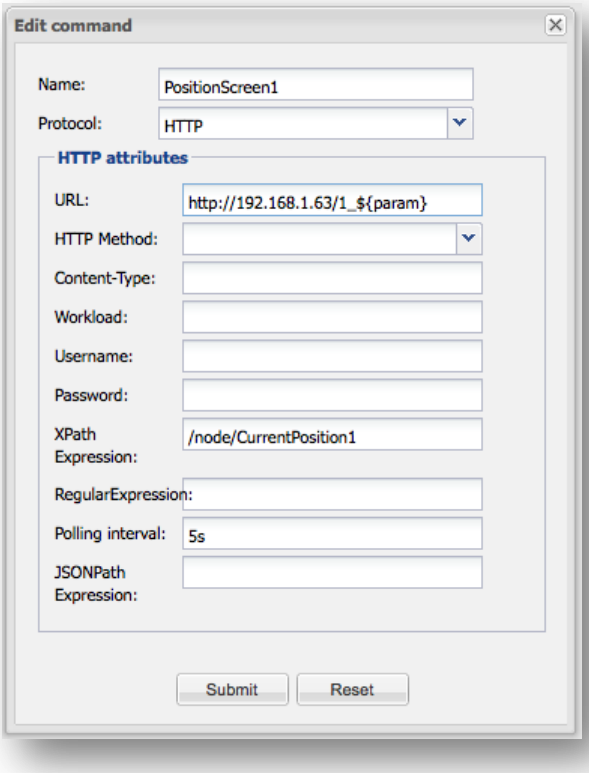
```
          digitalWrite(PinDown[i], HIGH); delay(200); digitalWrite(PinDown[i], LOW);
delay(200);
          Status[i]=2;

          DEBUG_PRINT ("DDOOWWNN---TimeCode: "); DEBUG_PRINT (TimeCommandsPressed[i]);
DEBUG_PRINT ("...ScreenID"); DEBUG_PRINT (i+1); DEBUG_PRINT ("...DOWN for ");
DEBUG_PRINT (DeltaTime[i]); DEBUG_PRINT (" ms...");
          DEBUG_PRINT (" Pin "); DEBUG_PRINT (PinDown[i]);DEBUG_PRINT("...Status:
");DEBUG_PRINT (Status[i]);DEBUG_PRINTLN ("");
        }

        else
          if (Direction[i]=="Up"){
          TimeCommandsPressed[i]=millis();
          digitalWrite(PinUp[i], HIGH); delay(200); digitalWrite(PinUp[i], LOW);
delay(200);
          Status[i]=2;

          DEBUG_PRINT ("UUPP-------TimeCode: "); DEBUG_PRINT (TimeCommandsPressed[i]);
DEBUG_PRINT ("...ScreenID"); DEBUG_PRINT (i+1); DEBUG_PRINT ("...Up for ");
DEBUG_PRINT (DeltaTime[i]); DEBUG_PRINT (" ms...");
          DEBUG_PRINT (" Pin "); DEBUG_PRINT (PinUp[i]);DEBUG_PRINT("...Status:
");DEBUG_PRINT (Status[i]);DEBUG_PRINTLN ("");
        }

      }
      }
  }

   for (int i=0; i < NumberOfScreens; i++){
      if (Status[i]==2 && millis()> TimeCommandsPressed[i]+abs(DeltaTime[i])-200){

        if (TargetPosition[i]==0 || TargetPosition[i]==10){
          DEBUG_PRINT ("HHOOLLDD---NO HOLD PIN PRESSED for ScreenID "); DEBUG_PRINT
(i+1); DEBUG_PRINTLN("");
        }
        else {
         digitalWrite(PinHold[i], HIGH); delay(200); digitalWrite(PinHold[i], LOW);
delay(200);
         DEBUG_PRINT ("HHOOLLDD---TimeCode: "); DEBUG_PRINT (millis()); DEBUG_PRINT
("...ScreenID"); DEBUG_PRINT (i+1);DEBUG_PRINT ("...HOLD...");
         DEBUG_PRINT (" PinHold "); DEBUG_PRINT (PinHold[i]);DEBUG_PRINT(" HIGH /
LOW...["); DEBUG_PRINT (millis()-TimeCommandsPressed[i]-200);DEBUG_PRINT ("
ms]");DEBUG_PRINTLN ("");
        }
        CurrentPosition[i]=TargetPosition[i];
        InitialPosition[i]=TargetPosition[i];
        DeltaPosition[i]=0;
        DeltaTime[i]=0;
        Status[i]=0;
        }
    }

}
```

Etape 3 : Installation d'un curseur de position ("slider") qui permet de définir la hauteur des volets roulants

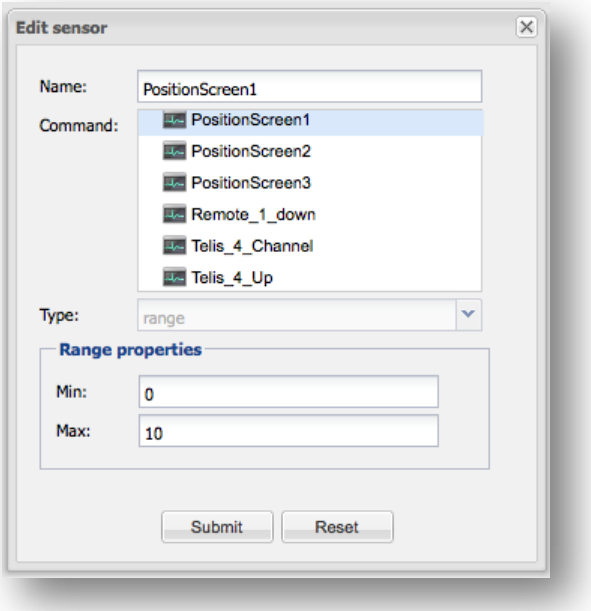
Dans le logiciel OpenRemote, commencez par créer une commande http http://192.168.1.63/1_${param}. OpenRemote émettra une commande de position du screen 1 par la variable {param} qui sera en fait définie par la position du slider du panneau de commande (voir plus loin).

Dans XPath Expression, entrez « /node/CurrentPosition1 » qui permet l'extraction (le « parsing ») de l'information de la position du volet roulant depuis le fichier XML que le serveur http de l'Arduino génère à chaque fois que OpenRemote le lui demande (toute les 5 secondes – « Polling interval »).

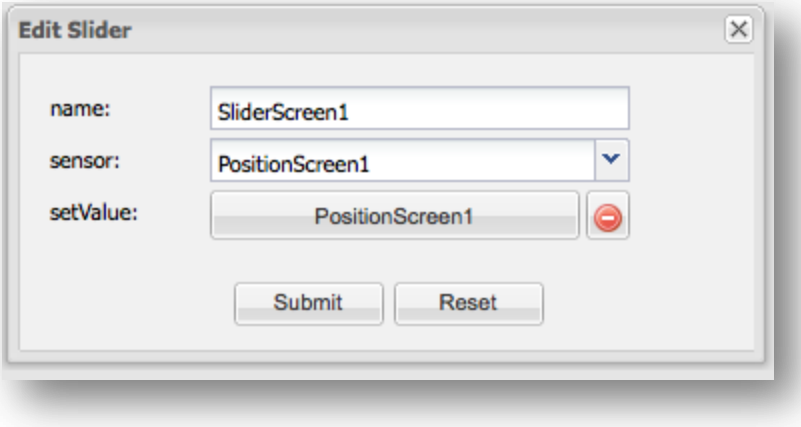
Notez que dans le code Arduino la section qui commence par « `if(Order=="$%7Bparam%7D"){` ». Elle a justement pour but de répondre à la requête du logiciel OpenRemote et de renvoyer la position « théorique » du volet roulant par le fichier XML.



Créez ensuite un «senseur », dont les valeurs vont de 0 à 10 (0 pour position complètement fermée, et 10 complètement ouverte).

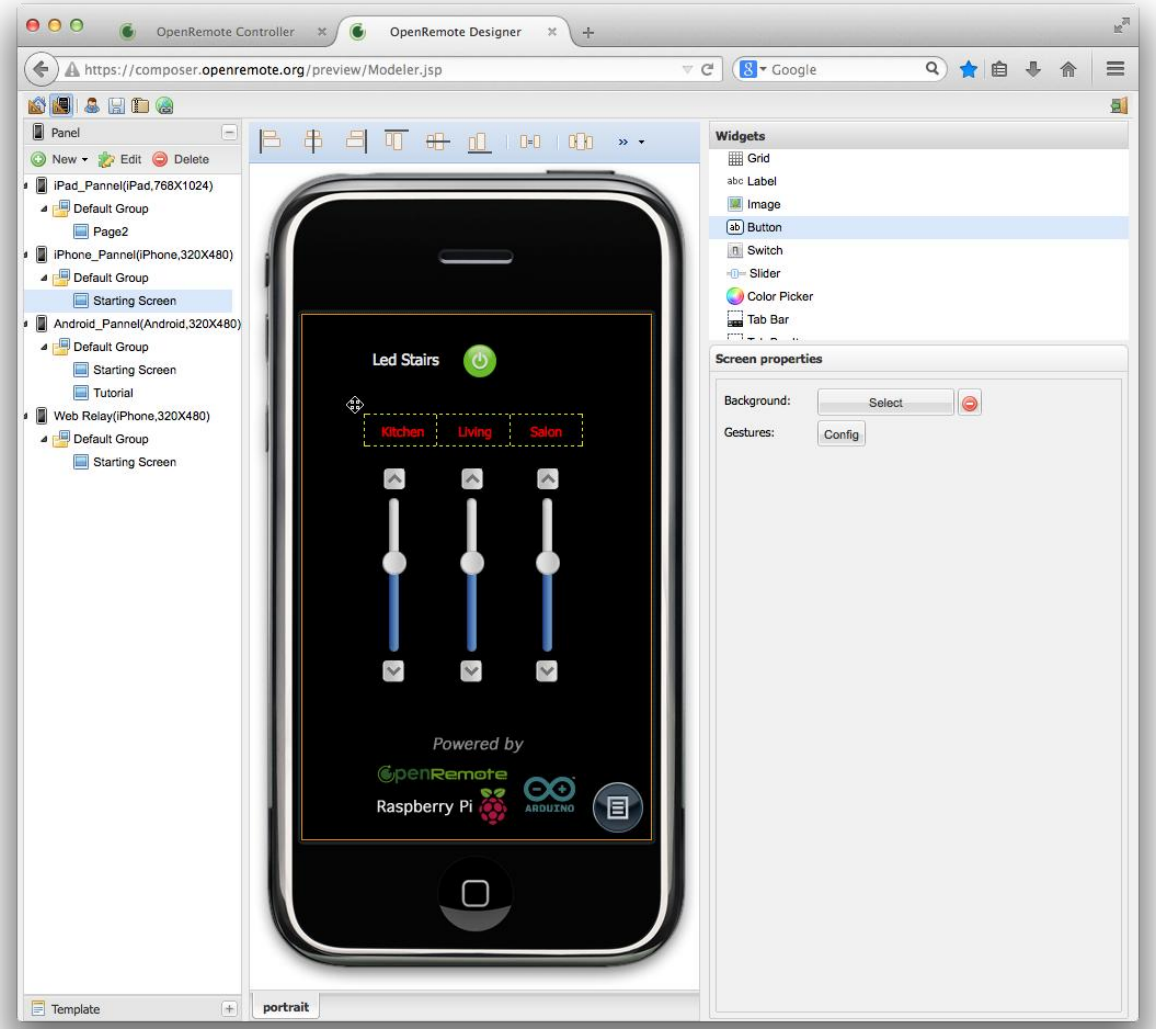


Enfin, il faut créer un d'un curseur de position (« slider ») dont la position sur le panneau de contrôle dépendra de la valeur reçue par OpenRemote à travers le senseur « PositionScreen1 » et inversement, c-à-d. que la position du curseur déterminera la position du volet roulant.



Pour terminer, c'est à vous de générer un panneau sympa, des boutons dont la couleur change en fonction de l'état et quelques images et légendes, tous cela selon votre goût. Ci-dessous, j'ai un panneau de commande sur iOS qui commande 3 volets roulants et des lumières leds le long de mon escalier. N'oubliez pas d'y inclure un bouton « settings ».

Pour information, je n'ai pas construit un système de relais comme décrit précédemment pour la commande des leds 220 V car j'ai préféré installer un relais Ethernet commercial (IPX800) au format DIN qui s'intègre dans mon coffret électrique. Ce dernier peut être commandé par une commande http, exactement comme nous l'avons décrit pour le trio relais-Arduino-Shield « maison ».


OpenRemote Controller
OpenRemote Designer
https://composer.openremote.org/preview/Modeler.jsp
Google
Panel
New
Edit
Delete
iPad_Pannel(iPad,768X1024)
Default Group
Page2
iPhone_Pannel(iPhone,320X480)
Default Group
Starting Screen
Android_Pannel(Android,320X480)
Default Group
Starting Screen
Tutorial
Web Relay(iPhone,320X480)
Default Group
Starting Screen
Template
Led Stairs
Kitchen
Living
Salon
Powered by
OpenRemote
Raspberry Pi
ARDUINO
portrait
Widgets
Grid
Label
Image
Button
Switch
Slider
Color Picker
Tab Bar
Screen properties
Background:
Select
Gestures:
Config

5. Perspectives

D'autres projets marrants existent sur le net. Dans des moments perdus, j'essaierai d'en évaluer et d'en implémenter certains, comme par exemple :

- Mesure de la consommation électrique avec data login sur le Web avec un senseur de courant non invasif (pince ampérométrique)
- Commande du Raspberry par email (comme par exemple pour arrêter / démarrer le server VPN)
- Commande du relais (ou volets roulants) basée sur des informations météo qui se trouvent sur le web (avec XML ou JSON parsing)
- Estimation du volume d'eau de pluie dans une cuve par communication sans fil
- Détecteur de courrier dans la boite aux lettres et envoi d'un email à l'aide d'un Arduino construit soi-même avec optimisation de la consommation électrique pour épargner les batteries, et par communication sans fil
- Caméra Raspberry commandée depuis l'extérieur.
- Essais d'utilisation d'une sonde « reed » pour la mesure de consommation de gaz ou d'eau de ville à mettre en contact avec les compteurs « officiels »

Certains demanderont des codes de programmation. Je vais donc me mettre au Python pour commencer…

Notes

Si ces notes sont utiles et intéressent des amateurs débutants de DIY dans le domaine de la domotique ou autre, j'envisagerai de faire un blog mais il y en a déjà beaucoup. C'est pourquoi dans un premier temps je préfère partager ce document sur quelques sites de discussion.

Enfin, soyez indulgent car ce document nécessite idéalement au moins une bonne relecture !