SERVEUR DE SMS (V1.0)

OBJECTIFS DE CE DOCUMENT

Ce document décrit la mise en place hardware et software du serveur de SMS au FabLab.

On verra en détail :

- La création du hardware,
- Le paramétrage du firmware,
- L'installation du plug-in Domoticz,
- Le paramétrage du plug-in Domoticz,
- Les commandes par SMS disponibles.

TABLE DES MATIÈRES

MATÉRIEL POUR LE SERVEUR SMS

Le hardware suivant est nécessaire pour construire le serveur SMS :

- Un ESP8266 qui gère le tout,
- Un modem GPRS de type A6, GA6 ou équivalent qui permet la connexion au réseau téléphonique,
- Un ATtiny 85 pour implémenter un chien de garde (optionnel, présent au FabLab),
- Un Arduino Nano pour gérer des entrées digitales et analogiques et des sorties digitales (optionnel, le support est soudé sur la platine FabLab, mais le Nano n'est pas présent, car par utilisé).
- Une commande d'extraction des fumées (Tongou 25A avec mesure),
- Une commande de la VMC (Tongou 25A avec mesure),
- Une commande du chauffe eau (Tongou 25A avec mesure),
- Un module de gestion de SMS (DIY à base d'ESP8266, modem GSM GA6 et alim 12V),
- 4 modules lecteurs de badges (DIY à base d'ESP8266 et lecteur de badge RC522)

CONVENTIONS

Les conventions suivantes sont utilisées dans ce document :

```
Les zones à saisir sont indiquées de cette façon.
```

En général, on doit les indiquer dans une fenêtre de type « Terminal ».

Les noms de touches sont spécifiées en majuscules entre crochets : par exemple [ENTER].

Les touches de modification sont spécifiées avec leur abréviation suivi d'un tiret et de la lettre à saisir. Par exemple [CTRL-C] ou[ALT-SHIFT-Z].

Les actions à réaliser et les textes variables à insérer sont indiqués comme <<connecter la clef USB>> ou <<mettre ici l'adresse IP du serveur>>.

FABRICATION DU MATÉRIEL

Toutes les informations relatives au matériel et au (micro) logiciel sont disponibles à l'adresse https://github.com/FlyingDomotic/FF_SmsServer

On l'installe sur la machine où on dispose d'un environnement de programmation de type « PlatformIO », soit native, soit intégrée à VSCodium (qui est la version libre et sans télémétrie de Visual Studio Code).

On peut aussi utiliser l'IDE Arduino, à condition de charger d'abord les librairies suivantes

- https://github.com/me-no-dev/ESPAsyncTCP
- https://github.com/me-no-dev/ESPAsyncWebServer
- https://github.com/gmag11/NtpClient
- https://github.com/bblanchon/ArduinoJson
- https://github.com/arcao/Syslog
- https://github.com/mgaman/PDUlib
- https://github.com/karol-brejna-i/RemoteDebug
- https://github.com/FlyingDomotic/async-mqtt-client#origin/fixPr306
- https://github.com/FlyingDomotic/FF_Trace
- https://github.com/FlyingDomotic/FF_Interval
- https://github.com/FlyingDomotic/FF_WebServer
- https://github.com/FlyingDomotic/FF_A6lib

Noter que ces installations n'ont pas besoin d'être réalisées avec VSCodium (qui le gère automatiquement à partir du fichier platformio.ini)

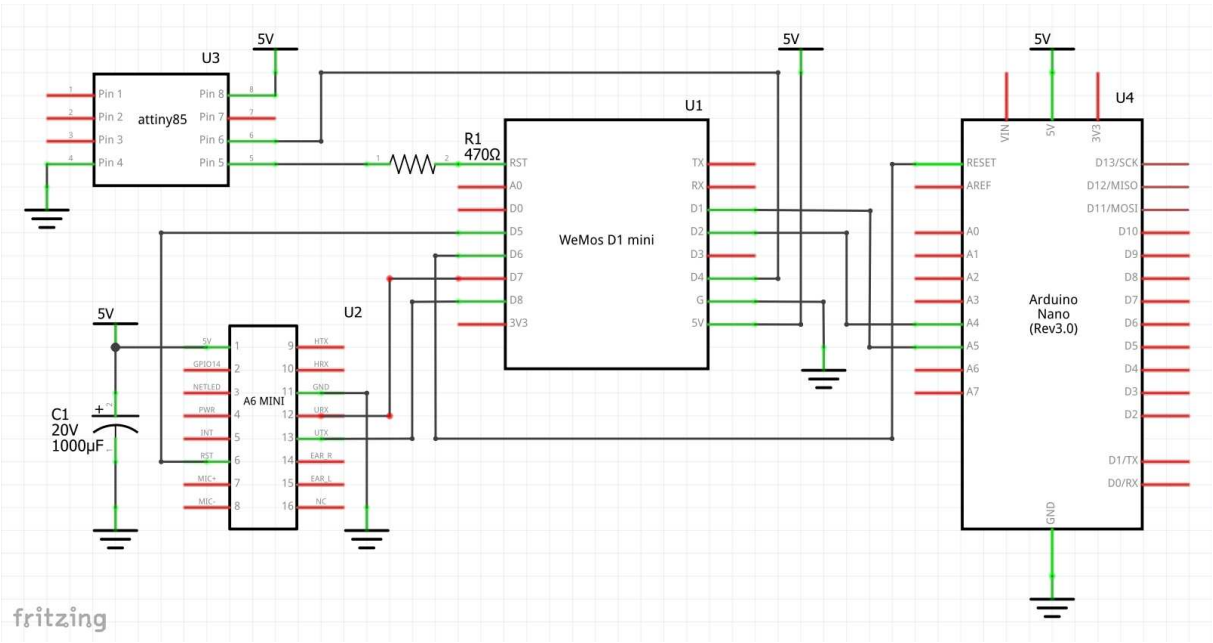
Choisir le répertoire où on souhaite installer le code, s'y positionner et passer la commande :

```
git clone https://github.com/FlyingDomotic/FF_SmsServer.git FF_SmsServer
```

Noter que le nom est FF_SmsServer, ce qui n'est pas visible au premier coup d'œil, les liens Internet étant soulignés par le caractère « underscore ».

On trouve les informations sur le matériel dans le sous-répertoire « examples/hardwareImplementation ».

Voici le schéma de principe :



Comme déjà dit, l'ATtiny 85 et l'Arduino Nano sont optionnels.

Voici également un schéma de connexion :

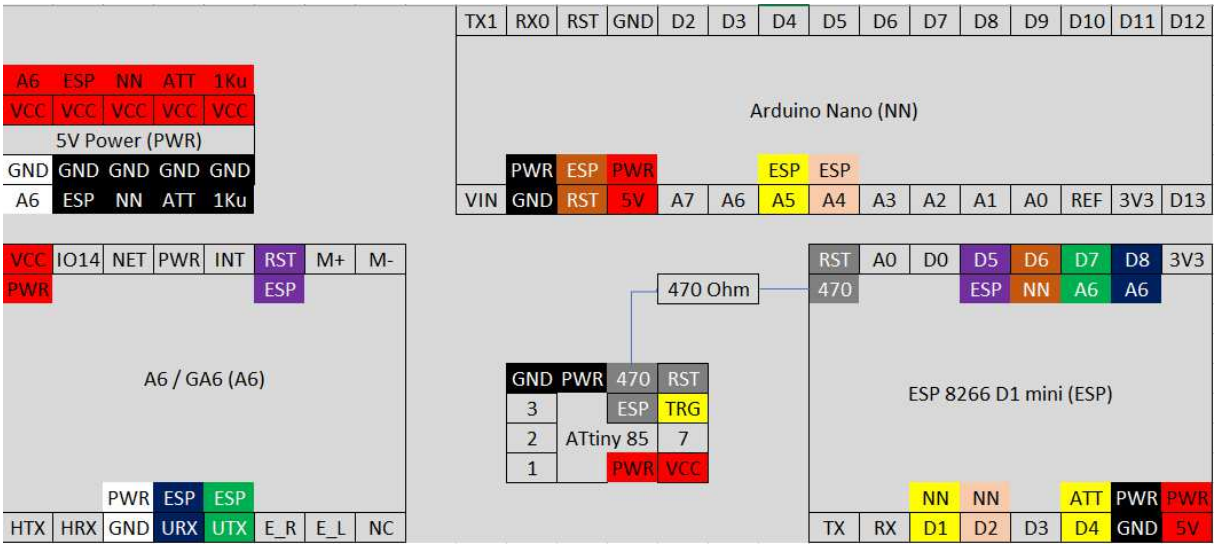
```
ESP8266
                (Antenna side)
                  +------+
(to ATtini 5)RST -+      +- TX  N/C
         N/C  A0 -+      +- RX  N/C
         N/C  D0 -+      +- D1  (to Nano A5)
 (to GA6 RST) D5 -+      +- D2  (to Nano A4)
(to Nano RST) D6 -+      +- D3  N/C
 (to GA6 URX) D7 -+      +- D4  (to ATtiny 6)
 (to GA5 UTX) D8 -+      +- G   GND
         N/C 3V3 -+      +- 5V  5V
                  +------+
                (USB side)
```

```
GA6
                (Antenna side)
                  +------+
        5V   VCC -+      +- HTX N/C
        N/C IO14 -+      +- HRX N/C
        N/C  NET -+      +- G   GND
        N/C  PWR -+      +- URX (to ESP D7)
        N/C  INT -+      +- UTX (to ESP D8)
(to ESP D5)  RST -+      +- E_R N/C
        N/C   M+ -+      +- E_L N/C
        N/C   M- -+      +- NC  N/C
                  +------+
```

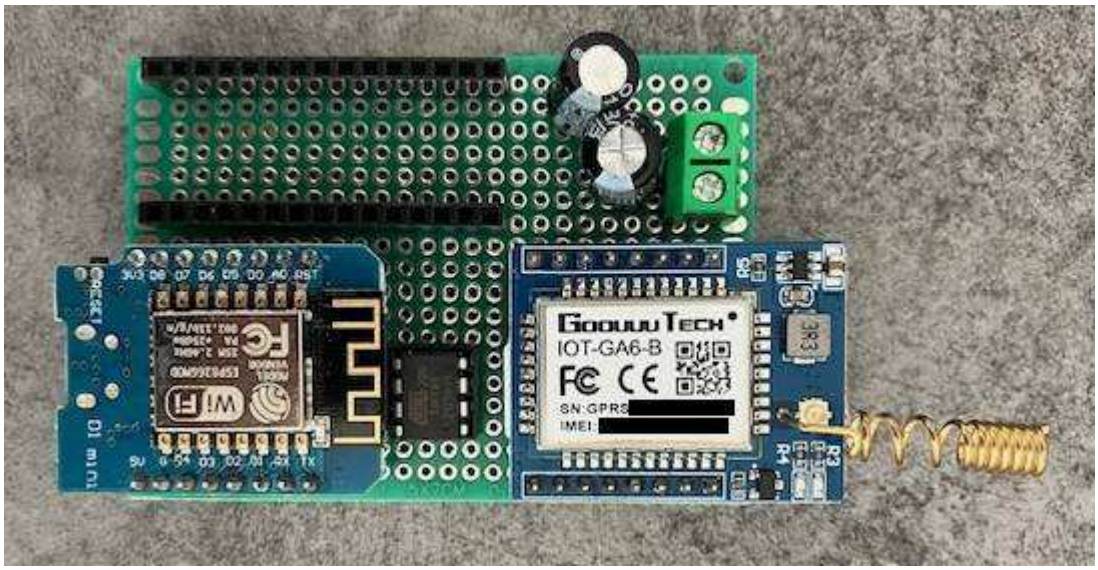
```
ATtiny 85
           +-----+
    N/C 1 -+ O   +- 8 VCC (3V to 5V)
    N/C 2 -+     +- 7 N/C
    N/C 3 -+     +- 6 Trigger (to ESP D4)
    GND 4 -+     +- 5 Reset (to ESP RST through a 470 Ohms resistor)
           +-----+
```

```
Arduino Nano
                  +------+
        N/C  TX1 -+      +- VIN N/C
        N/C  RX0 -+      +- GND GND
 (to ESP D6) RST -+      +- RST (to ESP D6)
        N/C  GND -+      +- 5V  5V
        N/C   D2 -+      +- A7  N/C
        N/C   D3 -+      +- A6  N/C
        N/C   D4 -+      +- A5  (to ESP D1)
        N/C   D5 -+      +- A4  (to ESP D2)
        N/C   D6 -+      +- A3  N/C
        N/C   D7 -+      +- A2  N/C
        N/C   D8 -+      +- A1  N/C
        N/C   D9 -+      +- A0  N/C
        N/C  D10 -+      +- REF N/C
        N/C  D11 -+      +- 3V3 N/C
        N/C  D12 -+      +- D13 N/C
                  +------+
                (USB side)
```

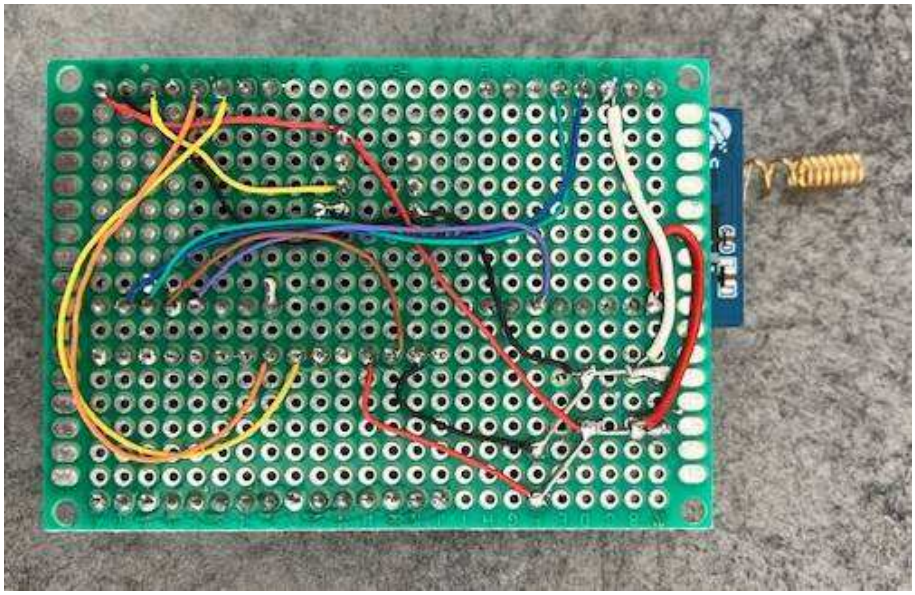
Voici une implémentation possible sur une platine à trous 7x9 :



… et sa réalisation physique d'un côté …



… et de l'autre :



PARAMÉTRAGE DU MICRO-LOGICIEL

Une fois la partie hardware réalisée, il faut maintenant créer et charger les différents micro-codes.

CARTE SIM

Noter qu'il faut insérer une carte SIM dans le montage pour que le serveur puisse fonctionner. Le modèle Free à 2 € par mois convient très bien, et offre un nombre illimité de SMS.

Pour éviter des blocages intermittents de la SIM, il est conseillé de la déverrouiller. On peut le faire soit en l'insérant dans un téléphone, soit en passant la commande :

```
AT+CLCK="SC",0,"<<code pin>>
```

Remplacer <<code pin>> par le code PIN actuel de la SIM.

Voici un exemple de programme à charger dans l'ESP pour passer des commandes « AT » sur le modem :

```
#include <SoftwareSerial.h>
SoftwareSerial BTSerial(0,0); // RX (INTERRUPT)| TX

// Use AT+CLCK="SC",0,"<<your pin>>" to disable PIN on a SIM card

char s = ' ';
char b = ' ';

boolean NL = true;
boolean NL2 = true;
char CR = 13;
char LF = 10;
char PROMPT = '>';

void setup()
{
  Serial.begin(74880);
  while (!Serial);
  Serial.println("");
  Serial.println("Serial ok");
  BTSerial.begin(9600, SWSERIAL_8N1, D7, D8, false, 250); // Connect on ESP:D7 on A6:U_TX and
ESP:D8 on A6:U_RX
  BTSerial.enableIntTx(true);
  while (!BTSerial);
  Serial.println("BT ok");
  Serial.println("Go!");
}

void loop()
{
  // Read from the module and send to the Arduino Serial Monitor
  while (BTSerial.available()) {
    b = BTSerial.read();
    if (NL2) {Serial.write(CR); Serial.write(LF); NL2 = false; NL = true;}
    Serial.write(b);
  }

  // Read from the Serial Monitor and send to the Bluetooth module
  while (Serial.available()) {
    s = Serial.read();
    BTSerial.write(s);

    // Echo the user input to the main window. The ">" character indicates the user entered
text.
    if (NL) {Serial.write(CR); Serial.write(LF); Serial.write(PROMPT);  NL = false; NL2 =
true;}
    if ((s == 10) || (s==13)) {NL = true;}
```

```
    Serial.write(s);
  }
}
```

MICRO-LOGICIEL POUR L'ESP8266

Le code comporte plusieurs options, qui sont activables dans le fichier « platformio.ini ». Voilà la version qui a été utilisée pour compiler la version FabLab :

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[platformio]
src_dir = ./
default_envs = smsServer

[env]
framework = arduino
platform = espressif8266
board = d1_mini
board_build.f_cpu = 160000000L
board_build.ldscript = eagle.flash.4m1m.ld
board_build.filesystem = littlefs
upload_speed = 460800
monitor_speed = 74880
build_flags =
        -D MQTT_MAX_PACKET_SIZE=512
        -D WEBSOCKET_DISABLED
        #-D USE_LIB_WEBSOCKET
        -D REMOTE_DEBUG
        #-D ATTACHED_NANO_RESET_TIME=1000
        -D ISOLATION_TIME=1000
        -D WIFI_MAX_WAIT_SECS=25
        -D HARDWARE_WATCHDOG_PIN=D4
        -D HARDWARE_WATCHDOG_ON_DELAY=5000
        -D HARDWARE_WATCHDOG_OFF_DELAY=1
        -D HARDWARE_WATCHDOG_INITIAL_STATE=0
        -D FF_TRACE_KEEP_ALIVE=5*60*1000
        -D FF_TRACE_USE_SYSLOG
        #-D FF_TRACE_USE_SERIAL
        #-D FF_TRACE_ENABLE_SERIAL_FLUSH
        #-D FF_A6LIB_DUMP_MESSAGE_ON_SERIAL
        #-D DISABLE_MDNS
        -D CONNECTION_LED=-1
        -D AP_ENABLE_BUTTON=-1
        -D AP_ENABLE_TIMEOUT=240
        -D ARDUINOJSON_DECODE_UNICODE=0
        -D DEBUG_FF_WEBSERVER
        -D SERIAL_COMMAND_PREFIX="command:"
        -D PRINT_RECEIVED_SMS_ON_SERIAL
        #-D SEND_SMS_FROM_SERIAL
        #-D ATTACHED_NANO_SLAVE_ID=56
lib_deps =
        https://github.com/me-no-dev/ESPAsyncTCP
        https://github.com/me-no-dev/ESPAsyncWebServer
        https://github.com/gmag11/NtpClient
        https://github.com/bblanchon/ArduinoJson
        https://github.com/arcao/Syslog
```

```
        https://github.com/mgaman/PDUlib
        https://github.com/karol-brejna-i/RemoteDebug
        https://github.com/FlyingDomotic/async-mqtt-client#origin/fixPr306
        https://github.com/FlyingDomotic/FF_Trace
        https://github.com/FlyingDomotic/FF_Interval
        https://github.com/FlyingDomotic/FF_WebServer
        https://github.com/FlyingDomotic/FF_A6lib
extra_scripts =
  pre:deleteEspAsyncWebServerSpiffs.py
  pre:setFirmwareName.py

[env:smsServer]
build_flags = ${env.build_flags}
monitor_filters = default, esp8266_exception_decoder
```

Ensuite, on peut générer le micro-code, et le charger dans l'ESP.

CHARGEMENT DES DONNÉES ET DE LA CONFIGURATION

Le serveur SMS se connecte au réseau par le WiFi. Du coup, SSID et mot de passe sont à indiquer dans le fichier de configuration du serveur (/data/config.json). Dans la ligne :

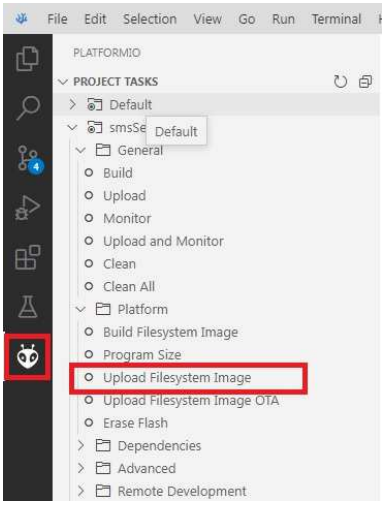
```
"ssid":"mySSID","pass":"myKey"
```

… remplacer « mySSID » par le SSID auquel se connecter (FabLab19), et « myKey » par le mot de passe du WiFi.

Charger ensuite le répertoire /data dans le système de fichier de l'ESP8266 (l'outil dépend de l'IDE utilisé).

Avec VSCodium, on active le menu « PlatformIO » en cliquant sur l'icône 

Charger le contenu du répertoire « data » en cliquant sur la ligne « Upload Filesystem Image»



MICRO-LOGICIEL POUR L'ATTINY 85

Il est possible installer un chien de garde sur le montage. Le code chargé dans l' ATtiny gère la pinoche « Reset » de l'ESP, et l'active s'il n'a pas reçu de changement de signal sur une de ses pinoches pendant une durée donnée. De son côté, l'ESP envoie des impulsions de temps en temps dans sa boucle loop. Si le code de l'ESP part en vrille, ces impulsions ne sont plus envoyées, et l'ATtimy va le réinitialiser.

Le code (super simple) se trouve à https://github.com/FlyingDomotic/FF_Watchdog

On doit l'installer localement, par :

```
cd <<là ou vous voulez l'installer>>
```

```
git clone https://github.com/FlyingDomotic/FF_Watchdog.git FF_Watchdog
```

On peut le charger avec l'IDE Arduino directement dans l'ATtiny.

MICRO-LOGICIEL POUR L'ARDUINO NANO

De même, il est possible d'ajouter un petit paquet d'entrée/sorties digitales et analogiques en intégrant un Arduino Nano. Comme pour le chien de garde, c'est un élément facultatif, qu'on peut utiliser pour commander quelques éléments hors d'un outil domotique (par exemple pour recycler une alimentation).

Pour le moment, on n'a pas trouvé de besoin au FabLab. Cependant, le module a été câblé avec le support pour l'Arduino, au cas où on lui trouverait quelques chose à faire ;-)

Dans l'implémentation actuelle, le Nano est utilisé en slave I2C de l'ESP. Le code doit être adapté au besoin, et basé sur une communication I2C entre l'ESP et le Nano, faite sur mesure.

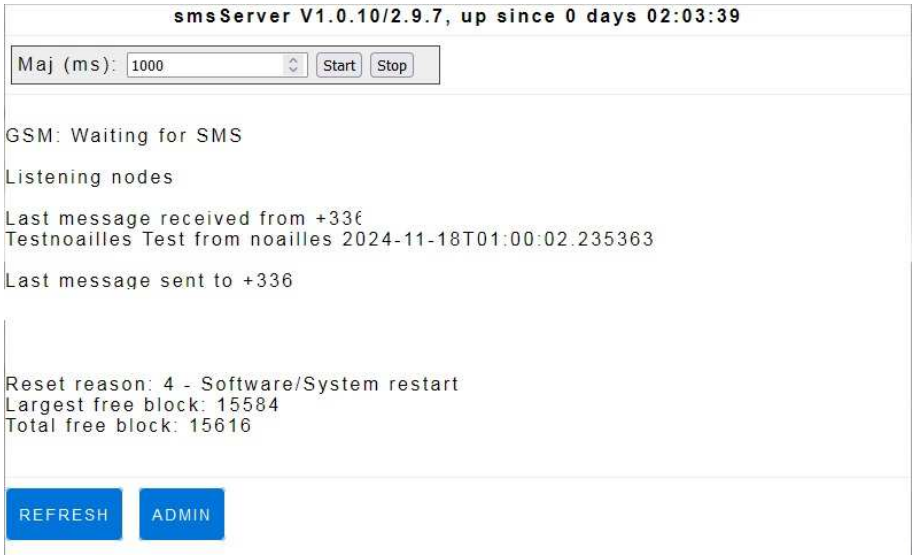
Voici un exemple de code pour lire des commande venant de l'ESP et lui répondre :

```
/*
 * This code is to be used with a Nano (or similar) controller to connect lot of (up to 18) contacts to another controller (ie ESP8266)
 *
 * Communications between Nano and ESP is done through SPI (ESP is master, Nano slave)
 *
 */

#include <Wire.h>

#define VERSION "2.5"
#define VERSION_HEXA 0x25
#define SLAVE_ID 59                                 // Slave #59

// Master request data
void requestEvent() {
  Serial.println("Received request from master");
  int message = <<What you want to send back to ESP>>
  Wire.write(message, 2);
}

// Master sent message
void receiveEvent(int nbChar) {
  // Get one command uint8_t
  Serial.print("Len received ");
  Serial.println(nbChar);
  uint8_t c = Wire.read();
  if (c) {
    Serial.print("Received ");
       Serial.print(c);
       << Do here what you want with received command>>
  }
}

void setup() {
  Serial.begin(115200);
  Serial.print("Slave starting, version ");
  Serial.println(VERSION);
  Wire.begin(SLAVE_ID);         // This is slave #59
  Wire.onRequest(requestEvent); // Called when requested to send data to master
  Wire.onReceive(receiveEvent); // Called when receiving data from master
}
```

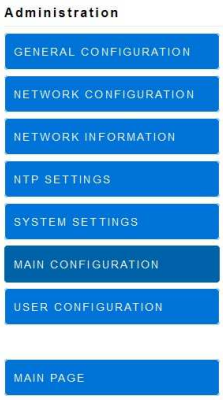
```
void loop() {
}
```

PARAMÉTRAGE DU SERVEUR DE SMS

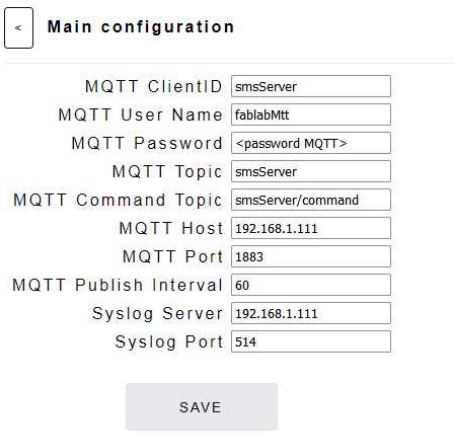
Maintenant qu'on a chargé l'ESP, et connecté l'ESP au WiFi, on peut attaquer le paramétrage du serveur SMS au travers de l'interface Web du serveur. On s'y connecte par http://smsServer/. On arrive sur une page telle que :



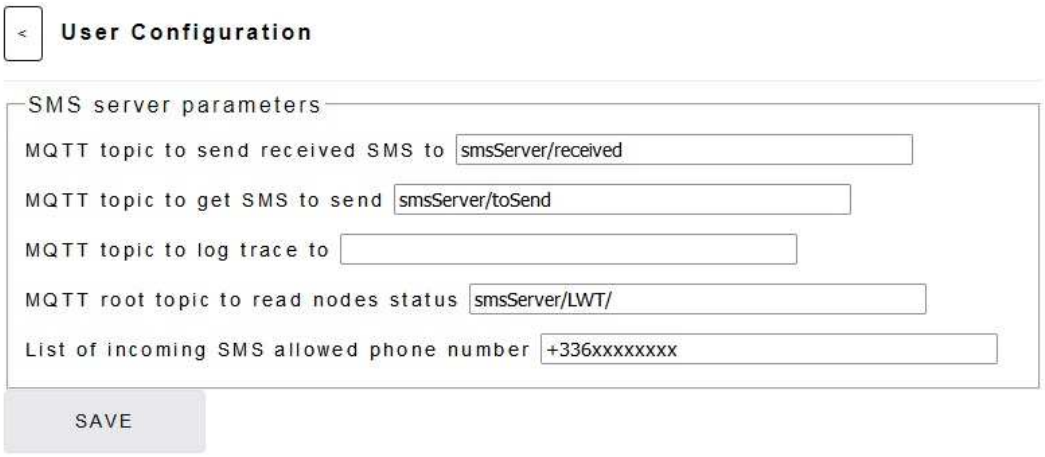
Cliquer sur « Admin »



Cliquer sur « Main configuration »



Saisir les valeurs indiquées ci-dessus, puis cliquer sur « Save ». On revient à la page précédente. Cliquer sur « User configuration ».



Saisir les numéros de téléphones autorisés à appeler le serveur, puis cliquer sur « Save »

Le serveur est paramétré. Simple, isn't it ?

INSTALLATION DU PLUG-IN DOMOTICZ

Le plugin Domoticz se trouve à https://github.com/FlyingDomotic/domoticz-ff_smsserver-plugin.

On l'installe par :

```
cd domoticz/plugins
git clone https://github.com/FlyingDomotic/domoticz-ff_smsserver-plugin.git FF_SmsServer
cd FF_SmsServer
```

PARAMÉTRAGE DU PLUG-IN DOMOTICZ

La génération automatique des commandes se fait en lançant l'outil suivant :

```
./FF_SmsServerConfig.py
```

Il est possible de relancer cette commande après chaque ajout/retrait de dispositifs.

Penser à relancer le plugin Domoticz après un changement.

Si les valeurs par défaut ne vous conviennent pas, éditer le fichier FF_SmsServerConfig.json pour l'adapter à vos besoins.

Lancer le script FF_SmsServerConfig.py qui va générer le fichier smsTables.json. L'éditer si vous voulez supprimer la visibilité de certains dispositifs, changer leur nom externe, ajouter des commandes ...

Relancer Domoticz.

S'assurer que Domoticz accepte les nouveaux dispositifs (voir « Astuces diverses » dans le document d'installation du serveur Domoticz).

Aller dans la page « Configuration » / « Matériel » et ajouter un matériel « FF_SmsServer with network interface ».

Donner le nom « smsTables.json » pour le fichier de configuration JSON (situé dans le répertoire du plugin FF_SmsServer).

LISTES DES COMMANDES PAR SMS DISPONIBLES

Lancer le script makeDoc.py qui va générer le fichier config.txt, qui contient, trié par nom de dispositif Domoticz, les commandes disponibles sur votre installation. Ne pas oublier d'ajouter le préfixe choisi (par défaut domoticz) devant toutes les commandes.

```
./makeDoc.py
```

Ne reste plus qu'à envoyer le fichier config.txt par mail, ou à l'imprimer, éventuellement en l'ayant fait passer dans un tableur pour lui donner un peu de couleurs ;-)