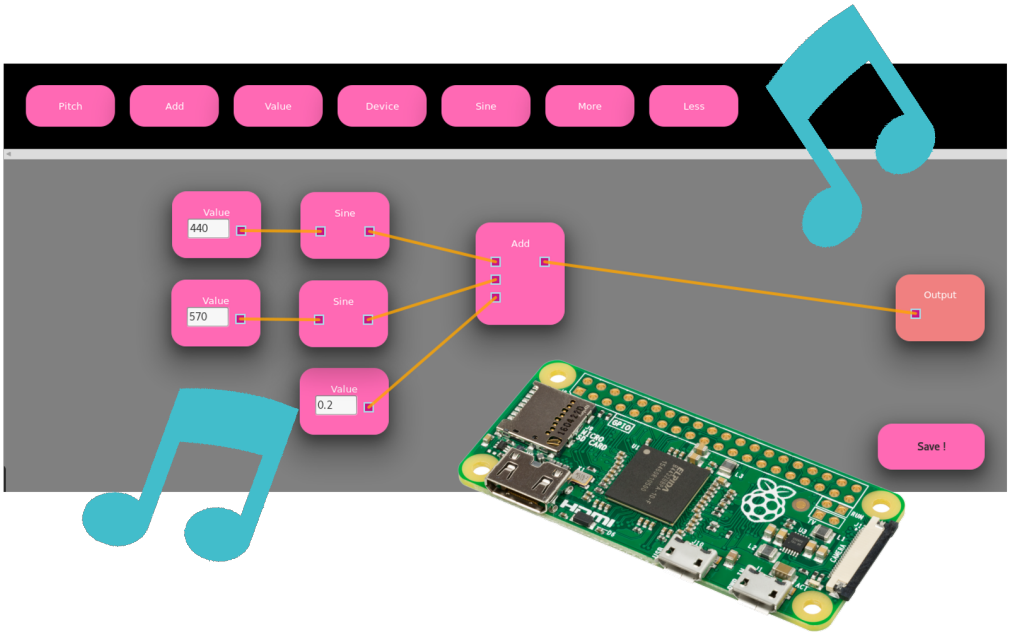
Lycée René Descartes

# *”The Music Swagger”* : De la musique interactive




*Estelle FIKET*
*Thomas POTIER*
*Nour BOULAHCEN*

encadré par
Mr. Bernigole

31 mai 2017

# Table des matières

# I. Présentation

Passionné par l'informatique, je souhaitais m'attaquer à un projet complexe réalisable en équipe. Avec Estelle et Nour, nous avions d'abord pensé à développer un jeu de type "Rogue" ou le joueur explore plusieurs niveaux. Puis, suite à nos discussions, nous avons décidé de créer un système fonctionnant comme un instrument de musique numérique : nous avons imaginé un système qui permettrait de générer du son en fonction de paramètres physiques tels que l'accélération, la distance ou tout ce qui peut être numérisé.

En effet, le traitement du son est un domaine qui nous plaisait et ce projet fait appel aux différentes notions étudiées cette année en cours d'ISN : représentation de l'information, algorithmique, langage et programmation ainsi que les architectures matérielles. De plus, notre idée ne semblait pas être déjà existante ou tout du moins très peu connue.

C'est ainsi que nous nous sommes lancés dans "*The Music Swagger*". L'idée du projet est de produire un système permettant de créer et faire varier des sons afin de générer une musique à partir de capteurs physiques (tels que gyroscope, mètres ultrasons, ou encore photodiodes). Nous avons souhaité privilégier l'ergonomie qui permet à un utilisateur non expérimenté de pourvoir simplement utiliser le projet grâce à un fonctionnement "Plug&Play" et une interface Web simplifiée.

# II. La mise en œuvre

## 1. Les objectifs

Nous nous sommes donné pour objectif de réaliser un système de génération de son à partir de données physiques. Afin d'y parvenir, nous avons avons défini plusieurs modules.

- une partie "server" qui gère la synthèse du son ainsi que l'organisation des données
- une partie "device" qui s'occupe de rapatrier les données d'un capteur physique (accéléromètre par exemple) au serveur
- une partie "client" qui permet la configuration du système

La partie "server" doit permettre le stockage des données reçues des "devices" tout en gardant leur origine. Elle est elle-même composée de plusieurs sous-parties distinctes : une partie Synthèse, une partie Communication et une partie Configuration.

La partie "device" doit permettre un fonctionnement automatique et abstrait de capteurs : cette partie est composée d'un module de communication ainsi que d'un module de gestion du capteur. Chaque "device" doit être fonctionnel après une simple implémentation d'un capteur et une alimentation.

La partie "client" est basée sur une interface Web permettant une portabilité et une facilité d'utilisation. Elle doit permettre de choisir quel son sera généré à partir de quel "device" et comment. Elle doit être intuitive et elle fonctionne en "boite" "drag&drop".

## 2. Les logiciels

### a) Outils

Afin de nous aider durant la phase de programmation et de reflexion, nous avons utilisé plusieurs logiciels. Le logiciel Web *Slack* (slack.com) a été choisi pour discuter et partager des morceaux de code et des idées au sein de notre groupe de travail. C'est une application permettant le "chat" ainsi que le partage de fichier et l'intégration de tierces parties comme *Github*. *Git* par le biais de *Github* est donc le second logiciel dont nous nous sommes servis afin de partager le code, y avoir un accès constant et pouvoir connaître l'évolution du projet. Enfin, je me suis servi du logiciel *PyCharm Community Edition* qui est gratuit pour un usage personnel et non commercial et qui est très puissant quand à la programmation en python notamment grâce à une correction syntaxique, orthographique (peu utile en Python mais pratique) et une correction logique (utilisation de variables qui n'existent pas, conseils d'optimisation ou d'inutilité d'une partie du code). Nous nous sommes servis d'un server *L(W)AMP* (Linux (Windows) - Apache - MySQL - PHP) pour le serveur Web. J'ai travaillé tout au long du projet sur un *Linux* (dérivé de *Debian* plus précisément) qui je pense est le logiciel le plus adapté à la programmation. Les ordinateurs *Raspberry* fonctionnent sous *Raspbian Light*. La totalité des logiciels que nous avons utilisé sont gratuits même s'il ne sont pas tous open-source.

J'ai d'ailleurs proposé à Estelle et Nour de leur installer un système *Linux* sur une clé USB afin qu'ils puissent utiliser leur ordinateur personnel et pour leur expliquer tout ce dont ils avaient besoin sur *Git* et autres. C'était peut-être un peu compliqué pour eux, car après plusieurs propositions, je n'ai pas eu de clé USB de leur part. J'ai donc été le seul à publier mon code sur *Github*.

### b) Langages

Notre projet possède une complexité qui réside dans le nombre important de langages que nous utilisons. En effet, nous avons eu besoin des languages suivant : Python3, PHP (PHP : Hypertext Preprocessor), HTML5 (HyperText Markup Language), JS (JavaScript), CSS3 (Cascading Style Sheets) et le SQL (Structured Query Language).

**Python3** Le Python3 a été utilisé pour une grande partie du projet notamment dans la communication entre le ”server” et les ”devices” ainsi que dans la génération du son.

**PHP** Le PHP nous à servi au niveau du ”server” pour permettre une communication avec la base de données.

**HTML5, JS, CSS3** Ces languages nous ont servi dans la partie configuration du ”server” pour l'interface Web. Le JS a permis l'interaction avec l'utilisateur notamment pour le ”drag&drop”. Le CSS3 a servi pour l'esthétique des pages et l'HTML5 à la forme et au contenu de ces dernières.

**SQL** Le SQL nous a permis d'interagir entre le PHP et la base de donnée MySQL.

**LaTeX** J'ai utilisé ce langage pour écrire mon dossier.

### c) Communication

Afin de faire communiquer les parties entre elles, nous avons utiliser plusieurs moyens.

**Wi-Fi/UDP/IP** Ce sont les protocoles utilisés pour la communication entre les ”devices” et le ”server”.

**JSON** C'est la façon dont nous envoyons la configuration à la page permettant de l'enregistrer via une requête ”POST”.

**MusicSwaggerProtocol** C'est le protocole que nous avons inventer afin de pouvoir faire communiquer les ”devices” et le ”server” basé sur UDP/IP.

### d) Source d'information

Nous avons souvent utilisé des sites internet suivants :

`http://stackoverflow.com` pour des erreurs connues et des astuces

`http://developer.mozilla.org` pour tout ce qui touche au développement Web

Nous utiliserons ensuite les termes suivants : "device" pour qualifier l'ensemble Raspberry Pi et capteur, "server" pour l'ensemble serveur Web et serveur Python3. Pour permettre la communication entre les "devices" et le "server", nous avons opter pour un système de liaison par Wi-Fi, et des "devices" à base de Raspberry Pi Zero W pour la portabilité et le prix. Une interface Web sera mise en place afin de pouvoir configurer le comportement de façon simple et sans besoin d'application particulière. Nous avons choisis d'utiliser Python3 pour sa simplicité, sa portabilité ainsi que la richesse de ses modules. L'interface Web est composée de PHP pour l'interaction avec la base de donnée et l'HTML, JS et CSS pour la mise en forme et les interactions avec l'utilisateur (notamment avec le système de "Nodes" en "drag&drop").

## 3. Le rôle de chacun

| Taches | Planning | |
|---|---|---|
| Interface Web | ≈20h | Nour, Estelle, Thomas |
| Communication | ≈10h | Thomas |
| Device | ≈10h | Thomas |
| Server | ≈10h | Thomas, Nour, Estelle |
| Intégration | ≈2h | Thomas |

## 4. Les différentes parties

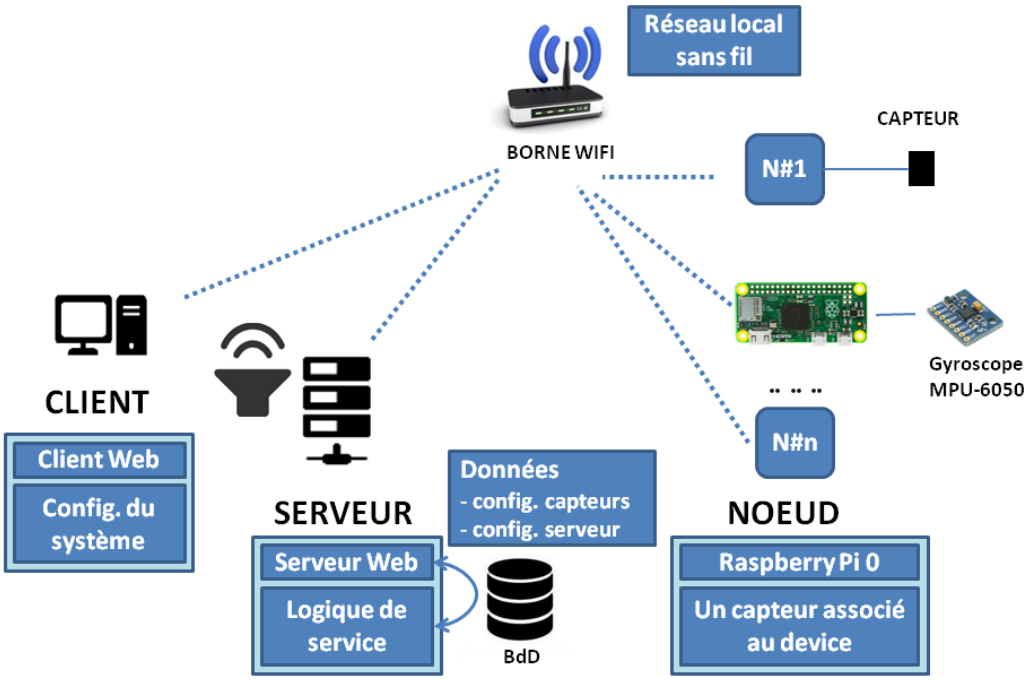
## 5. Autre

Nous avons choisis un intervalle de rafraîchissement du serveur à 50ms car il faudrait pour ne pas altérer le son avoir au moins une période de chaque fréquence audible tout en gardant un intervalle faible pour ne pas distinguer à l'oreille les changements de valeur. 50ms semble être un bon compromis car il permet d'aller jusqu'à une période de 50ms (soit une fréquence de 20Hz, minimum de l'oreille humaine) et donc peut jouer sans problème toutes les fréquences audibles.

# III. Réalisation

## 1. Mon travail

Voici une représentation du projet dans sa globalité.




**”device”** C’est la partie englobant le capteur, le ”DeviceBrain” et une partie de communication. Elle communique avec le ”server” grâce à un ”communicator” (interface de transfert de donnée par réseau (Wi-Fi) sur UDP/IP).

**”server”** C’est la partie qui permet de générer le son, la réelle partie qui met en forme et concrétise le projet. Elle contient une partie de génération de son, une partie de structuration des modifications du son et d’une partie de communication avec les ”devices”. Elle a un accès à la base de donnée du ”configurator” pour générer du son en fonction de ce que l’utilisateur désire.

**”configurator”** C’est l’interface Web basé sur un serveur *LAMP*, indépendante du reste du projet par ses langages (diffé-

rent de Python3). Elle communique ses données à travers une base de donnée. Elle est intuitive et fonctionne par ”drag&drop” de boites et liens.

Afin d’utiliser au mieux les compétences de chacun, il a fallu discuter de ce que chacun souhaitait faire et pouvait faire. Estelle débutait en informatique, Nour avait quelques compétences en programmation et moi qui est passionné par l’informatique je connaissais l’ensemble des langages et outils à utiliser pour réaliser notre projet.

J’ai pris à ma charge les développements les plus complexes (et les plus longs à développer) du projet, notamment une partie du ”server” qui permet de générer du son ainsi que la partie ”device” et la communication entre les deux. En effet, j’avais déjà quelques notions de communication réseau.

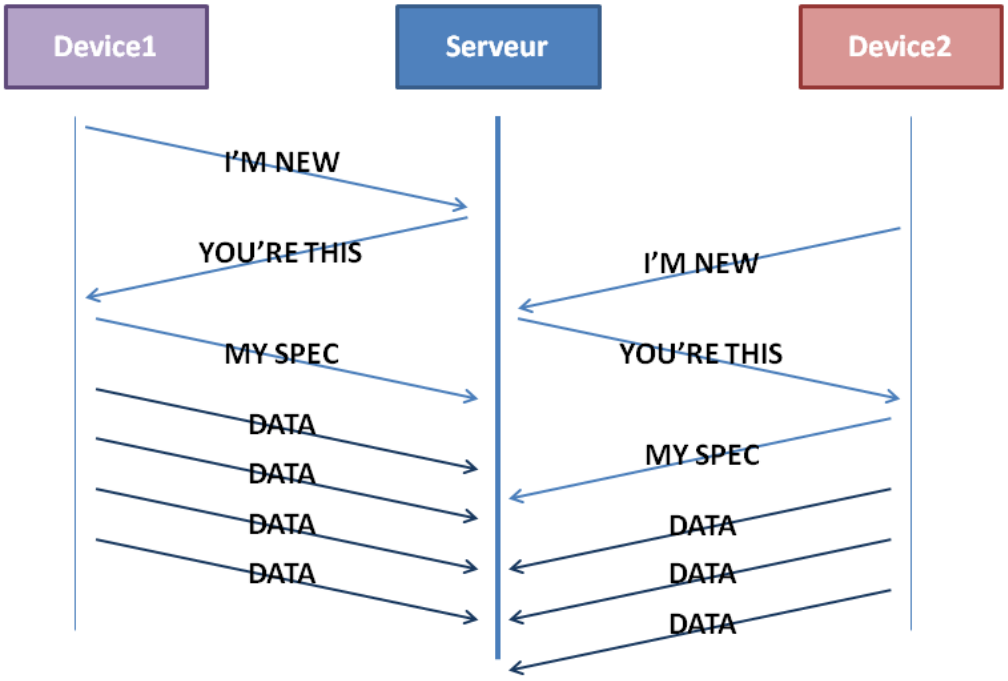
Je me suis dès le début chargé de structurer proprement le projet et de définir des interfaces simples et abstraites afin de faciliter le travail de chacun qui n’aurait alors qu’à s’occuper de sa partie en utilisant des fonctions simples et explicites. Exemple :

```
WaveGenerator().sinusoid(time=1000, sample_rate=22000, freq=440)
```

Chaque partie est délimitée par une classe Python.

Plus précisément, j’ai développé la partie de communication qui est présente du coté ”device” et ”server”. Celle-ci s’occupe de toute la communication, le ”server” et le ”device” ne s’occupent de rien au niveau du réseau. Le ”server” reçoit simplement une notification et les données envoyées par un ”device” par le biais d’un ”callback”, c’est-à-dire une fonction qui est appelée lorsqu’un événement se passe. Le ”communicator” possède

son propre ”Thread” ce qui lui permet de ne pas interférer avec la génération du son par exemple.

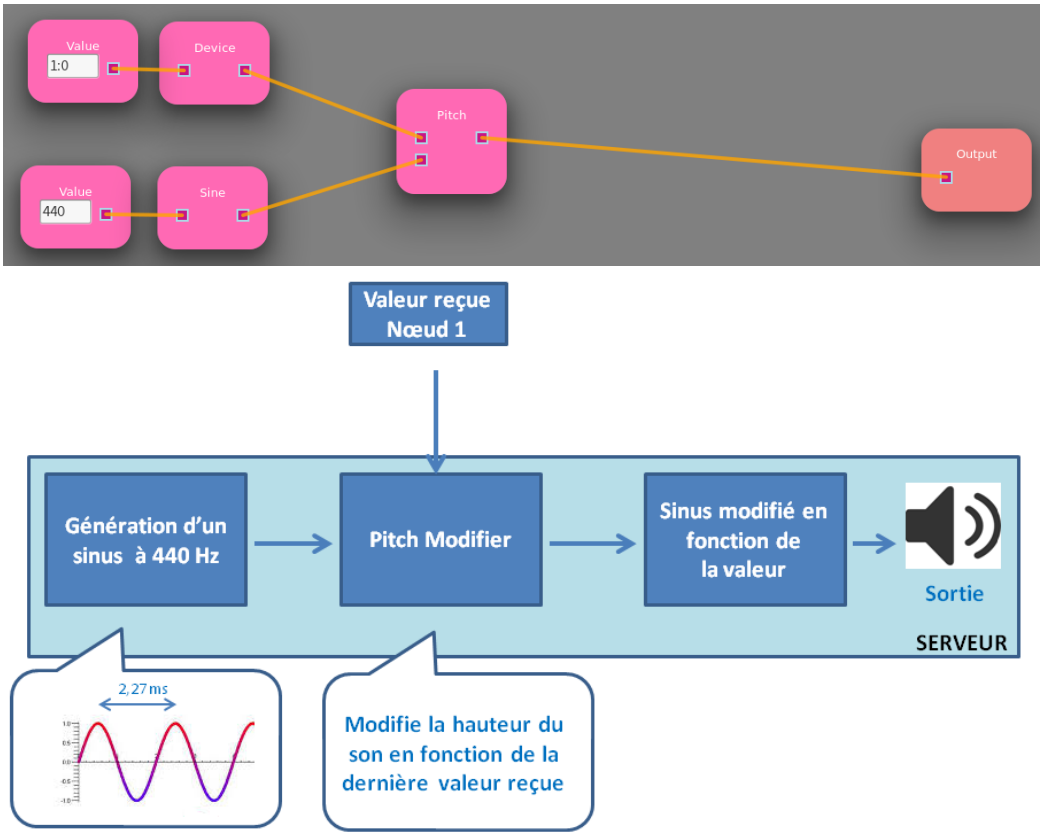



J’ai aussi développé la partie ”device” qui fait le lien entre la partie ”sensor” et ”communicator”. Cette partie est lancée au démarrage des RPi et permet au ”device” de commencer son fonctionnement.

Aussi, j’ai réalisé la partie de génération du son dans le ”server”. Ainsi, ce dernier peut aller chercher la dernière configuration disponible dans la base de données, retracer la structure en arbre ayant pour racine la sortie du son et générer à partir de là des portions de son de 50ms (intervalle minimal pour pouvoir avoir une période complète de la plus basse fréquence audible). Il est donc doté d’une partie permettant la lecture de son et d’une partie permettant la génération de son. De plus, une interface simple est disponible pour créer des ”boxes” qui permettent de prendre

un(des) son(s) ou une(des) valeur(s) et de les traiter. Chaque ”box” représente un nœud de l’arbre.

Voici un exemple de configuration ainsi que son schéma fonctionnel :




## 2. Rendu final

Afin de ne pas trop entrer dans le détail, voici une explication simplifiée du fonctionnement du logiciel. Tout d’abord, le serveur Web *LAMP* et le ”server” doivent être mis en marche.

Ensuite, l’utilisateur peut allumer les différentes ”devices”. Il va maintenant pouvoir configurer le *server* via l’interface Web.

Dès que le ”server” détecte une configuration fonctionnelle,

il va l'appliquer et générer le son. Le "device" dès son allumage envoie des données au "server" qui toute les 50ms va générer du son en fonction de ces dernières et de la configuration.

Enfin, le son est émis et peut être entendu.

## 3. Bilan

Les premiers enseignements ont été riches. Nous nous sommes lancés dans une projet intéressant mais ambitieux. Nous avons beaucoup discuté du projet en cours d'ISN, mais aussi au CDI, et de nos discussions, je pensais que nous avions tous la même vision de ce qu'il fallait faire même si nous n'avancions pas au même rythme. Je ne me suis rendu compte que tardivement qu'Estelle et Nour avaient du mal à avancer et je leur ai apporté mon support pour les aider à programmer en leur expliquant comment il fallait faire. De plus, l'arrivée tardive du code de Nour et Estelle n'a permis de faire l'intégration de l'ensemble que tardivement, ce qui a nécessité de reprendre des parties de code déjà rédigées. C'est une partie du projet que je n'avais pas envisagé de cette façon : cela m'a pris beaucoup de temps en plus de ce qui était à faire et je pense que les outils collaboratifs auraient du être utilisés plus tôt pour échanger plus vite sur les problématiques rencontrées. Afin d'améliorer l'avancement du projet, il aurait fallu rédiger un cahier des charge détaillé dès le début du projet et peut-être travailler sur un projet moins complexe : le projet nous a séduit tous les trois, mais je crois que j'étais le seul en début de projet à bien appréhender la complexité des développements. Il aurait été souhaitable aussi de faire régulièrement des points de synchronisation avec nos développements respectifs : nos réunions étaient très utiles, mais peut-être pas assez précises et détaillées sur le contenu de nos travaux, les échanges étant principalement oraux. Le bilan reste très positif, le projet fonctionne bien conformément aux objectifs que nous nous étions fixés et j'ai beaucoup appris sur le travail collaboratif. Plus globa-

lement, j’ai discuté et travaillé ponctuellement avec plusieurs personnes dans différents projets et j’ai tiré une grande satisfaction de pouvoir donner des idées, apporter un support aux différentes équipes qui pour certaines ont quelques lignes de code que j’ai pu rédiger. Le projet pourrait bien-sûr être amélioré, notamment en XXXX

et si nous avions un aperçu global du projet et de son découpage, je suis le seul que ce soit réellement investi dans le projet dès le début. En effet, le principal problème que j’ai relevé durant ce projet est que mes coéquipiers se sont mis a travailler beaucoup trop tard et n’ont pas rempli les taches qu’il devaient. Ainsi, je me suis retrouver dans l’obligation de faire une grande partie de leur travail pour faire avancer le projet et j’ai été bloqué durant une grande période car certaines portions dépendaient de leur travail.

# Annexes

## Music Swagger Protocol

```
MusicSwaggerProtocol v1.0 definition :
    - based on UDP/IP
    - simplest possible (shortest)

    - broadcast every packets

    - Server Port : 55666
    - Device Port : 55665

Data representation :
    | DEST CUID | SRC CUID | OP NUM | DATA LEN | DATA | ( CRC
    ↪ |)
    | (8b)      | (8b)     | (8b)   | (8b)     | (8b) | (
    ↪ (8b)|)

    - DESTination Connection Unique Identifier : the CUID of
    ↪ the destination of the packet (0x00 is the server, 0xff
    ↪ is no one particularly)
    - SouRCe CUID : the CUID of the source of the packet
    - OPeration NUMber : number describing which operation is
    ↪ given
    - DATA LENgth : the size of the data following in bytes
    - DATA : the actual data of the packet, formatted as the OP
    ↪ requires to
    (- CRC)

Constants :
    - DATA_VALUE_SIZE = 0x20
    -
```

```
Operations list :
    - OP NUM = OP NAME :
        - Description...
        | DATA DESC    |
        | (SIZE in bit)|
        - PARAMETER :
            - OPTION : DESC

    - 0x00 = INFO :
        - Generally for server response, give some info about
        ↪  the situation
        | ID  | INFO |
        | (8b)| (...)|
        - ID :
            - 0x00 : Nothing to say
            - 0x01 : Packet error
            - 0x02 : Invalid CUID
            - 0x03 : Not supposed to receive that
            # - 0x04 : I'm still alive ! (if not received for
            ↪  1min : forgetting the device)
            # TODO
        - INFO :
            Some text about the situation (ASCII).

    - 0x01 = IAMNEW :
        - When you start a device that needs a CUID to
        ↪  communicate over the network
        | GUID   |
        | (128b) |
        - GUID :
            The GUID of the device


    - 0x02 = YOURETHIS :
```

```
    - Server response to a 0x01 (give CUID)
    | GUID   | CUID |
    | (128b) | (8b) |
    - GUID :
        The GUID of the device
    - CUID :
        The new CUID of the device

-0x03 = MYSPEC :
    - give the device specs to server
    | NCHAN | NLEN | DLEN | NAME   | DESC   |
    | (8b)  | (8b) | (8b) | (NLENb)| (DLENb)|
    - NCHAN :
        The number of channel available
    - NLEN :
        The length of the device name
    - DLEN :
        The length of the description
    - NAME :
        The name of the device (utf8)
    - DESC :
        The description of the device (utf8)


- 0x10 = GIVE DATA
    - When device need to give data to server (can't work
    ↪  before a MYSPEC packet)
    | CHAN1            | ... |
    | (DATA_VALUE_SIZE)| ... |
    - CHAN1, ... :
        The values of each chanel

- 0x20 = GOODBYE
    - Before the device is turned off
```

## Base de donnée

(Certaines bases sont vides car se remplissent durant le fonctionnement)

Table 1 : Content of table available_tools

| ID | inputs | need_input | name | type |
|---|---|---|---|---|
| 1 | 2 | 0 | Pitch | PITCH |
| 2 | 3 | 0 | Add | ADD |
| 3 | 0 | 1 | Value | VALUE |
| 4 | 1 | 0 | Device | DEVICE |
| 5 | 1 | 0 | Sine | SINE |
| 6 | 4 | 0 | More | MORE |
| 7 | 4 | 0 | Less | LESS |
| 9 | 0 | 0 | Random | RAND |
| 10 | 2 | 0 | Multiply | MULTI |
| 11 | 2 | 0 | Sum | SUM |
| 12 | 2 | 0 | Doppler | DOP |
| 13 | 2 | 0 | Ampli | AMP |
| 14 | 2 | 0 | DIST | Distortion |

Table 2 : Content of table boxes

| ID | TYPE | BOX_ID | SPEC_PARAM |
|---|---|---|---|

Table 3 : Content of table connections

| ID | GUID | CUID | inited |
|---|---|---|---|

Table 4 : Content of table links

| ID | FROM_B | TO_B | WHERE_L |
|---|---|---|---|

Table 5 : Content of table specifications

| ID | numchan | name | description | CUID |
|---|---|---|---|---|

Table 6 : Content of table update_number

| ID |
|---|

## Github

http://github.com/TheMusicSwagger/Device La partie ”device” du projet

http://github.com/TheMusicSwagger/Server La partie ”server” du projet

http://github.com/TheMusicSwagger/Communicator La partie ”communicator” du projet

http://github.com/TheMusicSwagger/Documentation La documentation du projet