



::{ Rapport n°2 }::

Julien-Pierre AVÉROUS
Guillaume CALAS
Patrice DE SAINT STEBAN
Fabien DEBUIRE



Table des matières

Introduction	1
1 Le projet	2
1.1 Le groupe de projet	2
1.2 Avancement du projet	2
2 Plugs-In & SDK-Bridge [Julien-Pierre]	4
2.1 Introduction	4
2.2 Plugs-In	4
2.3 SDK-Bridge	6
2.4 Conclusion sur mon travail	7
3 Configuration [Patrice]	9
4 Interface graphique [Fabien]	11
4.1 Interface utilisateur	11
4.2 La vue spectrale : et oui encore !	11
5 L'équaliseur [Guillaume]	12
5.1 Avant propos	12
5.2 Plugs-in	12
5.2.1 clfft	13
5.2.2 clequa	13
6 Bibliothèque [Patrice]	15
6.1 SQLITE	15
6.2 Language SQL	16
6.3 La base du projet	17
7 Les tags [Guillaume]	18
7.1 Avant propos	18
7.2 ID3v2	18

8	Le site internet [Patrice]	20
9	Conclusion générale	22
9.1	Ce qu'il reste à faire	22
9.2	Conclusion	22

Introduction

Cette année, **CUNIXLINGUS Team** s'est lancé dans la mise au point d'un logiciel multi-plateformes (LINUX et MAC en priorité) de mixage audio (une platine numérique en quelque sorte).

Ce choix a été motivé par deux choses ; d'abord nous sommes tous des passionnés de musique électronique (Fabien est *Disk Jockey*), et ensuite nous sommes également très intéressés par la mise au point numérique de procédés mathématiquement très avancés.

De ce fait, notre projet, baptisé *Cl;MaX : Advanced Audio Shaker*, se veut résolument techniquement poussé sur certains points particuliers. Si nous en avons le temps, nous tenterions de tout recréer de A à Z de façon à interagir à tous les niveaux du traitement audio numérique, mais malheureusement nous devons poursuivre nos études en parallèle et notre niveau actuel en algorithmique avancée n'est pas encore suffisant de même que nos connaissances informatiques. Ainsi, nous avons et seront amenés à réutiliser du code en Open Source pour certains traitements très proches du hardware.

Quoi qu'il en soit, notre but final est de proposer aux utilisateurs un logiciel de mixage minimaliste et résolument évolutif grâce au développement de notre application sous formes de plugs-in. Un SDK (*Standard Development Kit*) sera d'ailleurs mis à disposition dans la version finale pour les développeurs extérieurs à notre équipe. Il est d'ailleurs actuellement utilisé par notre équipe pour la mise au point des plugs-in de base.

Enfin, il va s'en dire que ce projet risque de nous apporter énormément de connaissances à tous dans de nombreux domaines tels que le développement d'applications sous formes de plugs-in et le traitement avancé d'un signal quel qu'il soit.

Chapitre 1

Le projet

1.1 Le groupe de projet

Notre groupe, constitué de vieux briscards de la Sup-B2, fonctionne, à allure réduite certes, mais petit à petit notre projet prend forme.

Comme bien souvent, la plus grande difficulté aura été de rassembler les troupes pour travailler. Ce qui fut difficile pour Fabien à cause d'un déménagement précipité, ne le fut heureusement pas pour les autres membres du groupe.

Nous nous sommes donc réuni chez Guillaume pour travailler ensemble durant une semaine complète.

1.2 Avancement du projet

Cette deuxième étape dans l'avancement de notre projet fut cruciale dans le sens où nous avons dû apprendre à travailler avec le SDK concocté avec amour par JP. Ce dernier s'est donc collé principalement au dépannage de notre team, ce qui n'est pas peu dire.

Heureusement, JP a fait un travail exceptionnel dans le développement du SDK et nous a livré un kit de développement très facile à appréhender et à utiliser. De plus le SDK est assez souple et permet de créer réellement tout ce dont nous avons besoin pour développer nos plugs-in, et ce, de façon totalement indépendante les uns des autres. Hors recherches de développement, le temps de création d'un plug-in est réduit au maximum : moins d'une heure en moyenne ! Grâce à ce SDK nous allons pouvoir développer tous nos futurs plugs-in dans les meilleures conditions qui soient.

Pour cette deuxième soutenance, réservée à l'appréhension du SDK, nous nous étions fixés comme objectif de traduire notre premier jet de travail sous forme de

plugs-in, ainsi que de réaliser l'équaliseur et les premiers Vu-meters. Malheureusement, les problèmes de Fabien, qui était en charge de l'interface et particulièrement de la classe d'interface, ne nous ont pas permis de réaliser ce que nous voulions. Nous avons fait ce que l'on pouvait sans la classe d'interface, mais le visuel n'est pas au rendez-vous. On espère que la prochaine fois ce petit contre temps sera comblé.

Il nous reste donc un long chemin avant de pouvoir mixer nos premiers morceaux grâce à *CljMaX*, mais nous sommes d'or et déjà en possession de notre propre lecteur audio, et c'est déjà pas mal pour une deuxième soutenance. Le travail à venir sera néanmoins grandement facilité par notre SDK. Il nous reste donc à créer les visuels et les plugs-in d'effets : *reverb*, *echo*, *bpm*, etc.. Mais au final nous devrions être dans les temps.

Chapitre 2

Plugs-In & SDK-Bridge [Julien-Pierre]

2.1 Introduction

Pour cette deuxième soutenance, mon travail a été de finir ma réflexion sur la structure globale d'interaction et de connexion des plugs-in, et de déduire de cette réflexion une amélioration / modification de l'application principale et du SDK pour rendre ce dernier le plus souple, le plus complet et le plus logique possible. Mon travail en dehors de cette finalisation du SDK (qui n'a pas été une mince affaire) a été de rendre le projet parfaitement compatible entre l'environnement X11 de MACOS X et l'environnement d'un Linux "pur" comme une DEBIAN : il fallait qu'en tapant `qmake` et `make` dans la console ça compile de suite, et que ça fonctionne exactement pareil d'un environnement à un autre, pour faciliter le travail en groupe sur nos plateformes hétérogènes. Et de manière tout à fait prévisible les petites différences de fonctionnement sont nombreuses entre deux systèmes pourtant UNIX et POSIX. Finalement, j'ai commencé à faire les premières esquisses des plugs-in de bases primordiales ou non, à savoir la gestion globale de la configuration de l'application (plugs-in à utiliser, mise en relation, etc.) d'un flux audio entrant, d'un flux audio sortant, d'un gestionnaire de son bas niveau, d'un filtre de volume sonore, un "Vu Meter" et un "Spectre".

2.2 Plugs-In

- `clmaster`. C'est le plug-in "clef de voûte" de *Cl_iMaX*. Il est spécial et n'est pas chargé comme les autres plugs-in. Si *Cl_iMaX* ne le détecte pas au lancement, alors *Cl_iMaX* quitte. `Clmaster` est chargé de gérer la diversité des types de plugs-in présents dans le dossier `PlugIn` de *Cl_iMaX*. Effectivement, il peut y avoir plusieurs plugs-in d'entrée audio, plusieurs plugs-in de sortie audio, plusieurs plugs-in de mixage, etc.. Il faut donc gérer celui qui va

être actif (on ne peut pas avoir plusieurs plugs-in de sortie vers les hauts parleurs en même temps !), il faut pouvoir "dire" aux autres plugs-in quel est le plug-in qui est chargé à un moment donné de telle ou telle action, etc.. Pour se repérer, `clmaster` utilise les identifiants textuels de chaque plug-in, et charge par défaut les plugs-in de l'ensemble **Cl_iM_aX** (c'est-à-dire les plugs-in de notre création livrés par défaut avec **Cl_iM_aX**). En cela `clmaster` est primordial.

- `cloau`. Pour *Climax Output Audio*, requis par `clmaster`. Ce plug-in fait parti de l'ensemble **Cl_iM_aX**. Il utilise `FMOD`, et est chargé d'ouvrir un stream audio vers les haut-parleurs (on ne pouvait pas utiliser les flux standards `/dev/xxx` car sous Mac le serveur audio n'étant pas géré "à la Unix" : ça utilise un *Framework CoreAudio*). C'est `clmaster` qui est chargé de demander à `cloau` l'ouverture et la fermeture du stream audio sortant. Ce stream audio sortant est ensuite distribué par `clmaster` à tous les plugs-in susceptibles d'avoir à modifier la sortie sonore.
- `cliau`. Pour *Climax Input Audio*, requis par `clmaster`. Ce plug-in est chargé, à partir d'un nom de fichier, de retourner du contenu audio statique (opposé au *streaming*) en 44100 Hz, 16 bits stéréo. On utilise encore `FMOD`, cette fois pour ne pas à avoir à réécrire de nombreux algorithmes complexes de décompression ou de désencodage audio (AIFF, MP3, etc.). Toutefois, `FMod` ne sait pas faire de *resampling* (c'est-à-dire passer d'une fréquence d'échantillonnage vers une autre). J'ai donc du écrire un algorithme chargé de ça, qui reste à améliorer.
- `clplayer`. Pour *Climax Player*, requis par `clmaster`. Ce plug-in est chargé de stocker un tableau de `clFlow` renvoyés par le plug-in d'entré audio (`cliau` pour nous) selon un nom de fichier, de les placer sur le flux de sortie, tout en gérant un mix de tous ces `flow` (c'est-à-dire les "pistes" sont mixées et non écrasées), puis de libérer ces `clFlow` quand l'application quitte. Il n'utilise pas `FMod`. Il n'est pour l'instant représenté que par un bouton Play/Pause repris de `iTunes`, mais évidemment il évoluera dans le temps, de même que l'algorithme de mixage évoluera. `Clmaster` appelle bien sûr ce plug-in avant tous les autres, car c'est lui qui remplit le buffer audio de sortie, que les filtres vont ensuite modifier.
- `clvolume`. Pour *Climax Volume*. Simple plug-in en *bêta* permettant de régler le niveau sonore du flux de sortie. Il n'utilise pas `FMOD`. L'algorithme d'amplification sonore sera amélioré, ainsi que l'interface graphique. `Clmaster` appelle ce plug-in en dernier, après que les filtres aient été appliqués.

- `clvum`. Pour *Climax Vu Meter*. Simple plug-in en *bêta* chargé d'afficher l'amplitude sonore des deux canaux (gauche/droite) audio sortants. Il n'utilise pas FMOD. L'interface ainsi que l'algorithme doivent être largement améliorés. Toute la difficulté de l'algorithme est que l'on travaille sur le stream de sortie audio qui donne 44100 octets de données à chaque fois, soit 500 ms (il y a deux canaux) **avant** d'être envoyé vers la carte audio, alors que le rafraîchissement graphique se fait toutes les 10 ms. Il faut donc essayer de combiner les deux et faire prendre un léger retard au traitement d'affichage pour être en phase lorsque le flux audio est joué par la carte audio.
- `clspectre`. Pour *Climax Spectre*. Ce plug-in est chargé d'afficher un spectre de la sortie audio sous forme d'une ligne. On rencontre les mêmes problèmes qu'avec `clvum`. L'interface sera elle aussi améliorée.
- `claddsong`. Plug-in temporaire d'exemple, permettant d'avoir deux boutons dans l'interface pour ajouter deux sons d'exemple à `clplayer`.

2.3 SDK-Bridge

Pas mal de petites nouveautés dans le SDK et le Bridge sont palpables, mais pas mal de choses sont surtout de l'évolution de code et de la restructuration pour avoir un SDK toujours plus complet, souple, performant, et stable.

Nota bene : Le SDK est l'élément qui permet de développer un plug-in pour *Cl_iM_aX* : il met à disposition des fonctions, et un protocole. Le Bridge est un code interne à *Cl_iM_aX* qui permet de charger les plugs-in en mémoire, de charger leurs configurations, de gérer leurs fichiers de préférences, de se charger de la communication bas niveau entre les plugs-in et de relier les fonctions du SDK à *Cl_iM_aX*.

Les nouveautés notables :

- `clGlImage`. Le SDK (indirectement le Bridge) met à disposition la fonction `clGlImage` qui permet de charger rapidement des images avec gestion de la transparence. Il suffit de passer en paramètre de la fonction le nom du fichier (et son accès), éventuellement les dimensions que l'on veut en résultat, et le SDK charge tout ça en interne et renvoie un identifiant de liste de construction OpenGL. L'utilisateur n'a alors plus qu'à l'utiliser très simplement pour afficher l'image chargée. QT ne gérant pas (encore) la transparence des images pour OpenGL, j'ai dû chercher sur internet différents

loaders d'image et les réunir sous une même classe (cette classe utilise la signature des images pour déterminer le format et la fonction-décodeur à utiliser. Les formats reconnus sont le JPEG -avec la librairie libjpeg-, le PNG -avec la librairie libpng-, le BMP - décodé en dur- et le TGA -décodé en dur-) pour ensuite fournir facilement des données utilisables par OpenGL en tant que texture.

- *Les types de Plugs-in.* Il y en a des nouveaux qui permettent d'affiner la spécificité de chaque plug-in. Les `kClLecturePlug` agissent sur le flux de sortie audio, ils doivent posséder une fonction d'ajout de musique, et sont appelés en premier par `clmaster`. Les `kClAmpliPlug` agissent sur le flux de sortie audio, mais à l'unique différence des filtres, ils sont appelés en dernier par `clmaster`. Enfin, les `kClVisPlug` ne sont plus obligés de traiter le flux de sortie audio.
- `clFlow`, qui n'était pour l'instant qu'un typedef sur `void*`, est à présent une structure qui contient un pointeur `void*` vers des données audio, un entier `int` contenant la taille des données audio en octets, et un pointeur `void*` `userData` qui permet de sauvegarder des données dans un flux statique d'un appel à un autre.
- `clRequestListPlug`. Cette fonction permet d'obtenir une liste des infos de tous les plugs-in (permettant ainsi d'avoir la liste des "id").
- `clIsKeyCode` - `clIsKeySym`. La gestion du clavier arrive avec cette nouvelle version. Chaque plug-in d'interface peut recevoir un événement clavier et savoir si c'est tel `KeyCode` ou tel `KeySym` qui a déclenché l'événement, ainsi que le code ASCII de la touche (la dernière s'il y en a plusieurs) appuyé. Utilise l'API X11.

2.4 Conclusion sur mon travail

Le travail a bien avancé, le SDK est maintenant pleinement opérationnel : les premiers plugs-in utiles ont été écrits. Il a fallu créer du code clair et bien commenté pour que le reste de la team, qui a commencé aussi à utiliser le SDK, puisse le prendre facilement en main. Il m'a fallu aussi pas mal galérer sur la structure finale à adopter, mais aussi sur la correction de bugs qui apparaissaient sur une plateforme et pas sur une autre.

Les travaux de "sous-sol" sont donc presque finis, les bases sont là et elles sont pleinement fonctionnelles, bien conçues et stables. L'évolution visible de l'application peut pleinement commencer avec l'écriture des plugs-in et l'amélioration de chacun d'eux.

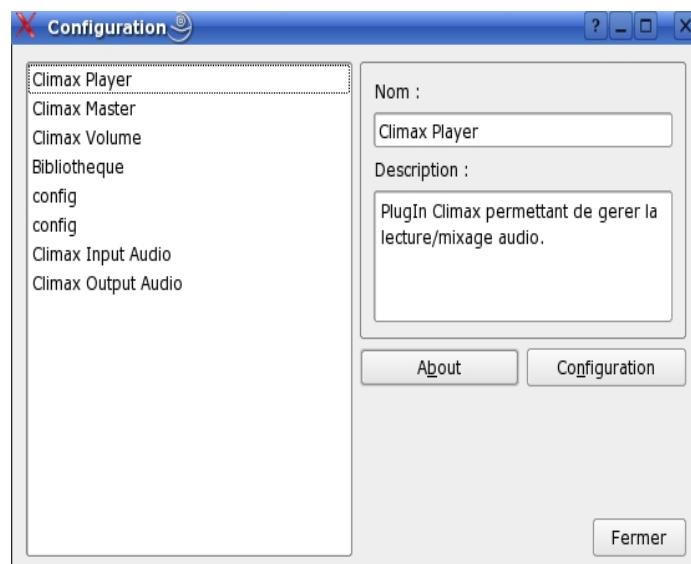
Chapitre 3

Configuration [Patrice]

Chaque plug-in possède une configuration propre ainsi que des informations sur l'auteur du plug-in. Le plug-in *Configuration* liste tous les plugs-in chargés par *Cl_iM_aX* et en affiche la liste. En cliquant sur un des plugs-in de la liste, le Nom et le Commentaire du plug-in apparaît. Deux boutons, About et Config permettent de lancer, si elles sont définies, les fonctions `clAbout` et `clConfig` du plug-in sélectionné.

Chaque plug-in définit donc lui-même la fenêtre de configuration, ce qui permet à chaque programmeur d'être vraiment libre et de construire sa fenêtre comme il le souhaite. C'est la même chose pour la fenêtre About, un programmeur peut décider de la faire en mode console. Mais Qt permet de créer des boîtes d'information : c'est pourquoi, pour aider et simplifier la vie, dans le SDK de *Cl_iM_aX* une fonction permet de créer cette boîte de dialogue en lui passant juste en paramètre le nom du plug-in, l'auteur, la version et un commentaire.

Pour créer ce plug-in, je devais récupérer la liste des plugs-in, or le programme principal ne le faisait pas encore, il a fallu faire la modification pour que cela puisse être possible. Puis savoir comment créer une fenêtre Qt sans Qt designer.



Chapitre 4

Interface graphique [Fabien]

Pour l'interface graphique, nous avons décidé de la faire dans un grand applet OpenGL. Il est à noter que l'avantage de cette décision est qu'en fait on va avoir une fluidité ainsi qu'un certain confort dans la manipulation des plugs-in.

4.1 Interface utilisateur

L'utilisation de la bibliothèque GLUI a grandement simplifiée mes démarches. Tout d'abord j'avoue avoir été quelques peu désorienté par la création d'une bibliothèque de contrôle. Le problème s'est vu réglé à la découverte de GLUI, car grâce à cette dernière l'interface graphique est quasi-terminé. A savoir que cette bibliothèque répertorie déjà tout ce qui est *sliders*, boutons, cases à cocher et *labels* par exemple. Le principe de fonctionnement est assez simple, à chaque événement on associe un entier à une variable qui est ensuite "matchée" à un motif afin de savoir quelle sera l'action associée.

4.2 La vue spectrale : et oui encore !

La vue spectrale est pour l'heure mon plus gros soucis car cette dernière est très gourmande en ressource entraînant des erreurs assez fréquemment. Cependant d'ici à la soutenance je pense être capable d'en montrer une sans trop d'erreurs.

Le principe de cette vue est de récupérer grâce à une fonction F_{Mod} des points de volume à intervalle régulier afin d'en tracer le graphique dans une fenêtre animée.

Chapitre 5

L'équaliseur [Guillaume]

5.1 Avant propos

Un égaliseur est un dispositif électronique, ou numérique, qui permet de modifier un signal audio en fréquence. C'est-à-dire que l'on peut modifier l'amplitude de certaines bandes de fréquences ce qui permet de créer toutes sortes d'effets. Pour modifier les amplitudes en fréquence, on a deux possibilités : l'égaliseur logarithmique, qui agit sur des bandes de fréquences qui doubles d'une bande sur l'autre, et l'égaliseur linéaire, qui agit sur des bandes de largeur identiques.

La principale difficulté est donc de convertir un signal audio numérisé du domaine temporel au domaine fréquentiel. Pour cela, on utilise un formidable outil mathématique : la *transformée de FOURIER*, et plus précisément la *transformée rapide de FOURIER*¹ (TFR ou FFT en anglais). Cet algorithme permet également de réaliser de gros calculs très rapidement tout en majorant l'erreur.

5.2 Plugs-in

Pour réaliser l'égaliseur de *ClM_aX*, j'ai découpé mon travail en deux plugs-in : un plug-in outil (`clfft`) et un plug-in filtre (`clequa`). De cette manière, les autres plugs-in pourront, s'ils en ont besoin, utiliser la FFT. Par exemple, les *Vu-meter* pourront utiliser la FFT pour récupérer la trace spectrale (bien qu'une option soit directement incluse dans la fonction qui gère l'égaliseur pour récupérer la trace en un seul passage).

¹Algorithme de COOLEY & TUKEY

5.2.1 `clfft`

Le plug-in outil gérant la transformée de FOURIER est celui qui m'a donné le plus de mal. En effet, après mes premières recherches sur le sujet, j'avais opté pour un algorithme itératif, plus rapide et moins gourmand en mémoire vive. J'avais récupéré un algorithme Open Source et je l'avais adapté pour l'occasion. Mais après maintes efforts, je me suis aperçu que cet algorithme était non bijectif, i.e. que l'on pouvait récupérer la trace spectrale, mais que la transformée de FOURIER inverse n'était tout simplement pas possible sans grosses pertes de qualité. Mais le mal était fait, le temps perdu fut énorme. Par chance, j'ai rapidement trouvé un autre algorithme, que j'ai un petit peu optimisé, et adapté pour mes besoins. Cet algorithme s'est révélé suffisamment vélocité pour ce que nous lui demandons. Le temps de traitement étant très largement inférieur au temps de lecture d'un *sample* de 500ms (44100 échantillons, stéréo, en 16bits signés).

L'utilisation de la FFT se fait très facilement, voici le prototype de la fonction :

```
void fft(char inv, short buffer_size, double *RealIn, double
         *ImagIn, double *RealOut, double *ImagOut)
```

Avec :

- `inv` : 0 pour FFT et 1 pour FFT Inverse
- `buffer_size` : nombre d'échantillons à traiter
- `RealIn` : valeurs réelles en entrée
- `ImagIn` : valeurs imaginaires en entrée
- `RealOut` : valeur réelles en sortie
- `ImagOut` : valeur imaginaire en sortie

5.2.2 `clequa`

Le plug-in qui gère l'équaliseur à proprement parlé est `clequa`. D'abord, ce plug-in s'occupe de découper le flux audio passé par le programme principal, en paquet de 16384 échantillons (il rajoute des 0 dans le cas où le nombre d'échantillons est insuffisant²). Ces échantillons sont ensuite passés à la fonction `PerformFFT` qui va s'occuper de séparer la voie gauche et la voie droite. Le traitement via le plug-in `clfft` est alors effectué sur chacune des voies avec une FFT. On récupère ainsi la trace spectrale qu'il ne reste plus qu'à modifier.

Pour modifier le spectre en fréquence il suffit de multiplier les bandes de fréquence par un certain coefficient α défini par la formule suivante :

²Zero padding

$$\alpha = 10^{\frac{G}{20}}$$

où G est le gain à ajouter à la bande de fréquence.

Par soucis de souplesse pour l'utilisateur, nous allons laisser une assez grande marge de manoeuvre allant de -50dB à +50dB. En fait, on veut laisser l'utilisateur libre de pousser les fréquences jusqu'au rupteur s'il le souhaite.

Comme Fabien ne nous a pas fourni d'interface, et encore moins de classe d'interface, je n'ai pas pu m'occuper de créer l'interface de configuration et d'utilisation de l'équaliseur. La seule chose que je sais, c'est que ça fonctionne, mes nombreux tests m'ont rassuré sur ce point. L'interface reste donc à mettre au point pour la prochaine soutenance.

Chapitre 6

Bibliothèque [Patrice]

Pour le projet nous avons eu besoin d'une base de donnée qui contiennent la bibliothèque des musiques pour pouvoir classer toute la musique qu'on a et pouvoir ensuite la trier et rechercher la musique que l'on veut lire. Il fallait que toutes ces actions se fassent facilement et qu'on puisse ajouter des nouvelles musiques aussi facilement.

Au début, on avait pensé à faire la base dans un fichier texte, mais cela n'est pas très pratique pour faire des tris, et des recherches. C'est pour cela que notre choix s'est porté sur une base SQL. Car le langage SQL est universel et simple à réaliser. Mais une base de donnée est stockée sur un serveur ou alors il faut l'installer sur l'ordinateur. C'est pourquoi nous avons choisi la base de donnée SQLITE, une simple librairie qui enregistre la base de donnée dans un seul fichier.

6.1 SQLITE

La base de donnée SQLITE est une petite librairie écrite en C qu'il suffit de lier avec le projet pour qu'elle fonctionne. Il n'y a pas de paramètres de base de donnée à donner (serveur, port, user ...) car cette librairie stocke toutes les données dans un fichier binaire. Elle est bien adaptée aux petits programmes qui ont besoin en interne d'une base de donnée, exactement comme *ClMaX*. Cette base de donnée utilise une API en C mais est aussi utilisée dans d'autres langages comme PHP ou DELPHI.

Ensuite, il y a trois commandes à connaître pour utiliser la base de donnée SQLITE : `sqlite3_open`, `sqlite3_close` et `sqlite3_exec`. Car pour utiliser la base de donnée, il faut d'abord l'ouvrir grâce à la fonction `int sqlite3_open(const char*, sqlite3**)` en lui passant en paramètre le nom de la base de donnée et

l'adresse d'un pointeur vers une structure de type `sqlite3`, renvoie si la connexion c'est bien passée ou pas. Ensuite la fonction la plus importante est la fonction :

```
typedef int (*sqlite_callback) (void*, int, char**, char**).
```

`int sqlite3_exec(sqlite3*, const char *sql, sqlite_callback, void*, char**)` qui permet d'exécuter une commande SQL. Cette fonction prend en paramètre la structure `sqlite3`, la requête SQL, puis une fonction de *callback* qui est appelée après la réception de la requête, pour chaque ligne de la requête. Le quatrième paramètre est un paramètre qui sera envoyé en premier paramètre de la fonction de *callback*. Enfin le dernier paramètre est un pointeur vers une chaîne de caractère qui contiendra le message d'erreur si il y en a un. Et enfin, on doit utiliser la fonction `int sqlite3_close(sqlite3*)` pour fermer la connexion à la base.

6.2 Language SQL

Le langage SQL (*Structured Query Language*, traduisez Langage de Requêtes Structurées) est un langage de manipulation de données qui permet de créer, modifier et sélectionner des données d'une base de donnée relationnelle.

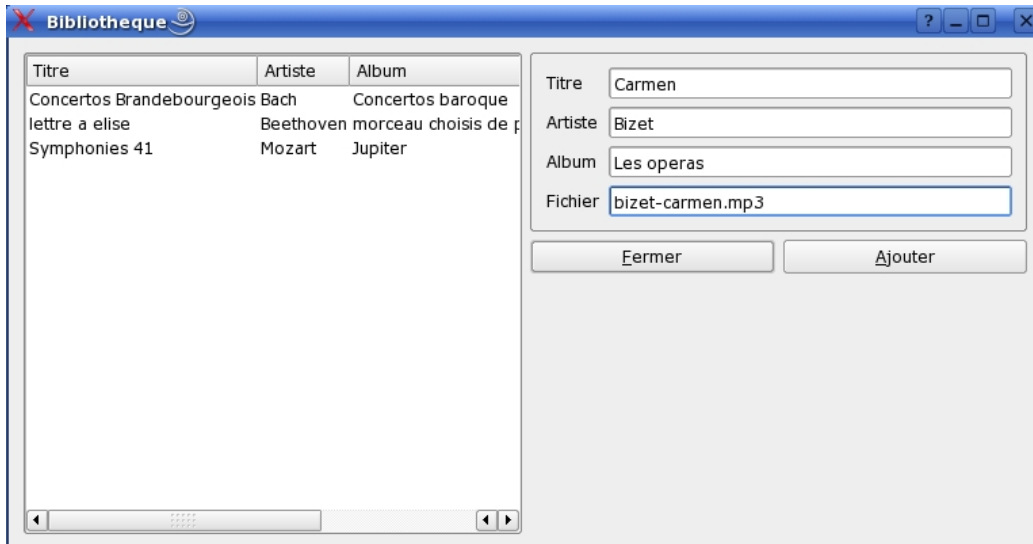
La principale commande de SQL est la commande `SELECT` qui permet de sélectionner des données dans une table d'une base de donnée. Voici la syntaxe de la requête :

```
SELECT [ALL] | [DISTINCT] <liste des noms de colonnes> | *
FROM <Liste des tables>
[WHERE <condition logique>]
[ORDER BY <colonne de trie> ASC|DESC]
```

On ne peut ainsi sélectionner que quelque champs d'une table, ou même de plusieurs tables, puis mettre des conditions grâce à la clause `WHERE`. La clause `ORDER BY` permet de trier les données avant de les renvoyer au programme.

Mais le langage SQL permet d'insérer et de modifier ou même supprimer des données dans une table grâce aux instructions suivantes :

```
INSERT INTO <Nom de la table> (colonne1,colonne2,colonne3,...)
VALUES (Valeur1,Valeur2,Valeur3,...)\
UPDATE <Nom de la table> SET Colonne = <Valeur Ou Expression>
WHERE qualification \
DELETE FROM <Nom de la table> WHERE qualification
```



6.3 La base du projet

Dans *Cl_iM_aX*, la base de donnée sert à avoir accès à notre liste de musique pour pouvoir ensuite la charger dans une des pistes du programme. Cette bibliothèque permettra de classer et de rechercher la musique que l'on voudra lire ensuite. Pour l'instant ce plug-in ne fait que lister les entrées qui sont déjà dans la base de donnée, mais on pourra ajouter de nouvelles musiques et aussi scanner un répertoire pour pouvoir ajouter toutes les musiques du répertoire, puis de les ajouter dans la base de donnée. On pourra ensuite modifier ou supprimer les informations comme le titre d'une chanson, l'artiste ou l'album. Voici la structure pour l'instant de la base.

```
CREATE TABLE `file` (
  `id` INT NOT NULL ,
  `titre` VARCHAR( 50 ) ,
  `artiste` VARCHAR( 50 ) ,
  `album` VARCHAR( 50 ) ,
  `file` VARCHAR( 80 ) NOT NULL ,
  PRIMARY KEY ( `id` )
);
```

Pour créer ce plug-in, il a fallu que je comprenne le fonctionnement de la librairie `SQLITE`, puis de réaliser à la main la fenêtre `Qt`. Il m'a aussi permis à me familiariser avec la création de plug-ins pour *Cl_iM_aX*.

Chapitre 7

Les tags [Guillaume]

7.1 Avant propos

Les tags sont des informations qui sont collées directement à la suite, ou à la tête, de certains fichiers, dont les fichiers musicaux font parties. Ces tags contiennent tout un tas d'informations supplémentaires mais pas forcément indispensable. Par exemple, un tag pourra contenir le nom de la chanson, le titre de l'album, l'année de parution, le nom de l'artiste, etc..

Toutes ces informations sont très utiles pour remplir la base de donnée simplement et sans demander le moindre effort de la part de l'utilisateur. Le but est donc d'arriver à extraire les informations contenues dans les tags des fichiers audio afin de les classer dans la base de donnée.

7.2 ID3v2

En ce qui concerne le MP3, l'ID3 est le tag qui fait référence. Actuellement en version 2, ce tag est très complet et évolutif. Chaque tag se décompose en frames pouvant contenir différents types d'informations (informations, images, données numériques, etc.). La taille maximale d'un tag peut atteindre 256Mo avec des frames d'une taille maximale de 16Mo, autant dire qu'on arrivera ,pour ainsi dire, jamais à les remplir. L'id3¹ est l'organisme en charge de la normalisation et du développement de ce format. On trouve sur leur site toutes les informations nécessaires sur l'ID3v2. Ils mettent également à disposition différentes bibliothèques pour les développeurs, ce qui est une aubaine en ce qui nous concerne.

Cette bibliothèque permet d'effectuer les opérations de base, à savoir la lecture

¹<http://www.id3.org>



et l'écriture des tags. Nous n'avons donc pas à coder ces fonctions ce qui nous fait gagner un temps précieux. Il ne reste plus qu'à l'intégrer à la base de donnée de notre projet, ou de créer un plug-in outil, ce que nous ferons sans doutes très bientôt.

Chapitre 8

Le site internet [Patrice]

Un bon projet se doit d'avoir un bon site internet. C'est pourquoi j'ai recréé un site internet au visuel adapté à notre projet. C'est-à-dire que le site possède plusieurs rubriques, comme la présentation du projet, du groupe, mais aussi le téléchargement des rapports, du projet, des sources.

Comme le projet n'est composé que de plugs-in, le site possède une page qui rassemble tous les plugs-in disponibles, leurs fonctions, ainsi que le téléchargement de chacun d'eux pour les différentes plateformes ainsi que leurs codes sources. Le site doit aussi insister d'autres programmeurs à écrire des plugs-in pour notre programme, c'est pourquoi une partie du site leur est réservé qui contiendra la documentation du SDK mais aussi des tutoriaux pour les aider à développer pour *ClMaX*.

Pour simplifier la création, et la mise à jour du site internet, j'ai utilisé un système de CMS (*Content Management System*) que je développe maintenant depuis un an pour de nombreux sites dont je m'occupe. Ce système permet de tout modifier du site directement en ligne, de créer des pages, de modifier le menu, d'ajouter des photos...



FIG. 8.1 – Le site Internet

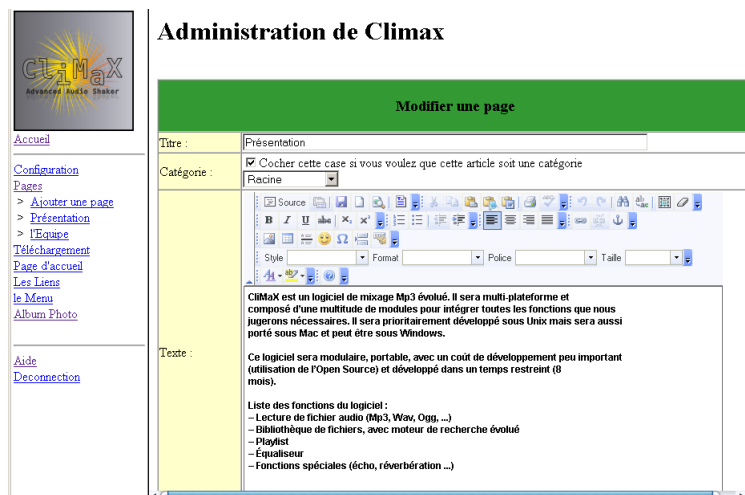


FIG. 8.2 – Administration du Site Web

Chapitre 9

Conclusion générale

9.1 Ce qu'il reste à faire

Malgré ce bon début, la route est longue avant de pouvoir commencer à mixer nos propres morceaux. . .

Il nous reste à réaliser, le "scratch", élément indispensable à toute bonne platine de mixage, le plug-in de calcul de BPM, ainsi qu'une bonne floppée de filtres en tous genres. Et bien sûr, la base de donnée pour gérer nos pistes audio, les documentations et l'installateur.

Il nous reste donc largement de quoi faire. . .

9.2 Conclusion

Pour ce deuxième projet au sein de l'EPITA, on peut dire que ça part plutôt bien. Nos expériences acquises l'année dernière nous sont bien utiles et nous permettent de progresser à la vitesse "grand V".

Pour le moment nous n'avons pas eu de gros problèmes, et nous espérons que cela va durer ce qui nous permettrait de rester serein pour la poursuite en parallèle de nos études.

On se retrouve donc à la prochain soutenance pour de nouvelles, et trépidantes, aventures numériques. . .