

# WebradioManager

---

Travail de diplôme 2014

Simon Menetrey – T.IN E2A – V1.2

28/05/2014



## 1 RÉSUMÉ

### 1.1 FRANÇAIS

Cette documentation décrit mon projet « WebradioManager » qui a pour but l'élaboration d'un logiciel C#/.NET permettant la gestion complète de multiples webradios. Le cahier des charges a été écrit en collaboration avec l'entreprise KTFM.

L'utilisateur peut gérer plusieurs webradios différentes à la fois. L'application contient une bibliothèque de musiques commune à toutes les webradios créées. Cette bibliothèque est remplie par l'utilisateur avec ses fichiers musicaux. Pour chacune des webradios, il est possible de créer des listes de lectures basées sur les musiques de la bibliothèque et gérer une grille horaire sur une semaine. Le but final est de générer un flux audio qui sera envoyé sur un serveur de diffusion (ce dernier distribuera le flux aux différents auditeurs, il peut être local ou distant). Chaque webradio peut créer différents flux pour différent serveur en se basant sur leur propre grille horaire et leurs propres listes de lecture. Enfin, chaque webradio dispose d'un serveur de diffusion local intégré.

Une base de données SQLite est utilisée pour sauvegarder les données du programme. La création de flux (transcodeur) et la création de serveurs de diffusion sont effectuées à l'aide d'outils nommés « ShoutCAST » qui sont développés par Nullsoft. Ces outils nécessitent des fichiers de configuration. Le programme s'occupe donc de générer ces fichiers à partir des données dont il dispose pour, ensuite, lancer les exécutable ShoutCAST et gérer leur fonctionnement.

### 1.2 ENGLISH

This documentation describes my project « WebradioManager » which aims to develop a C#/.NET software which allows a complete management of multiple webradios. The specifications were written in collaboration with the KTFM company.

The user can manage several different radios at once. The application contains a library common to all radios. This library is filled with music files by the user. For each of the webradios, it is possible to create playlists based on music library and manage a schedule over a week. The ultimate goal is to generate an audio stream to be sent to a streaming/broadcast server (the latter distributes the stream to different listeners it can be local or remote). Each webradio can create different streams for different servers based on their own schedule and their own playlists. Finally, each webradio has an integrated local broadcast server.

A SQLite database is used to store program data. The creation of stream (transcoder) and the creation of streaming server are performed using tools called « ShoutCAST » that are developed by Nullsoft. These tools require configuration files. The program generates these files from the data available (playlists, schedule and configurations) then run the ShoutCAST's executables and manage their operations.

## 2 CONTENU

1	Résumé .....	1
1.1	Français.....	1
1.2	English.....	1
2	Contenu .....	2
3	Introduction.....	9
3.1	Mise en situation .....	9
3.2	Sujet.....	9
3.3	Qu'est-ce qu'une webradio ? .....	9
3.4	Pourquoi ce sujet ? .....	10
4	Cahier des charges.....	11
5	Analyse fonctionnelle .....	13
5.1	Principe.....	13
5.2	Schéma de l'application .....	13
5.3	Interface principale .....	14
5.3.1	Fenêtre d'administration.....	14
5.3.2	Menu principal .....	14
5.4	Gestion de plusieurs webradios .....	15
5.4.1	Création d'une webradio.....	16
5.4.2	Sélection d'une webradio.....	16
5.4.3	Fermeture d'une webradio .....	16
5.5	Serveur de diffusion interne et externe .....	17
5.6	Onglet « Status » .....	18
5.7	Gestion des musiques/pubs .....	19
5.7.1	Affichage.....	19
5.7.2	Importer et indexer .....	19
5.7.3	Ajout à une playlist .....	20

5.7.4	Modifier .....	20
5.7.5	Supprimer .....	20
5.8	Gestion des listes de lecture .....	21
5.8.1	Listes de lecture musicales .....	21
5.8.2	Listes de lecture publicitaires .....	21
5.8.3	Création .....	21
5.8.4	Génération automatique.....	21
5.8.5	Affichage du contenu d'une playlist.....	22
5.8.6	Retirer d'une playlist .....	22
5.8.7	Recherche .....	22
5.9	Gestion des horaires.....	23
5.9.1	Événement.....	23
5.9.2	Événement périodique .....	24
5.9.3	Remplissage manuel.....	24
5.9.4	Modification d'un événement.....	24
5.9.5	Suppression d'un événement.....	25
5.10	Gestion des transcodeurs.....	26
5.10.1	Création .....	26
5.10.2	Affichage.....	27
5.10.3	Modification .....	27
5.10.4	Contrôles .....	27
5.10.5	Capture live.....	27
5.10.6	Historique de diffusion .....	28
5.11	Gestion du serveur de diffusion .....	28
5.11.1	Contrôles .....	28
5.11.2	Log .....	29
5.11.3	Configuration.....	29

5.11.4	Interface web.....	29
5.11.5	Administration web .....	29
6	Analyse organique .....	31
6.1	Environnement.....	31
6.2	Diagramme de classes .....	31
6.2.1	<i>Model</i> .....	31
6.2.2	MVC .....	31
6.2.3	Observateurs/Sujet .....	32
6.2.4	AudioType et StreamType .....	34
6.2.5	Stockage des webradios .....	35
6.3	Base de données.....	36
6.3.1	SQLite.....	36
6.3.2	Utilisation .....	36
6.3.3	Suppression en cascade.....	37
6.3.4	Schéma .....	38
6.3.5	Twebradio.....	39
6.3.6	Tserver.....	39
6.3.8	Tcalendar .....	40
6.3.9	Tcalendarevent.....	40
6.3.10	Tplaylist.....	41
6.3.11	Taudiotype.....	41
6.3.12	Tmusic.....	41
6.3.13	Tgender.....	42
6.3.14	Tplaylist_has_music.....	42
6.3.15	Thistory.....	43
6.3.16	Ttranscoder .....	43
6.4	Schéma de diffusion .....	45

6.4.1	Principe de base .....	45
6.4.2	Infomaniak.....	45
6.5	ShoutCast.....	46
6.5.1	Présentation .....	46
6.5.2	Pourquoi cet outil ? .....	46
6.5.3	Serveur .....	47
6.5.4	Transcoder.....	47
6.5.5	Schéma de fonctionnement résumé .....	48
6.6	Structures des dossiers/fichiers .....	49
6.6.1	Schéma .....	49
6.6.2	Exécutables Shoutcast.....	50
6.7	Initialisation de l'application .....	51
6.8	Webradio.....	53
6.8.1	Classes associées .....	53
6.8.2	Affichage des webradios disponibles .....	53
6.8.3	Création .....	53
6.8.4	Chargement.....	55
6.8.5	Duplication .....	56
6.8.6	Suppression .....	57
6.8.7	Changement de nom .....	58
6.8.8	Génération des configurations.....	58
6.9	Bibliothèque .....	60
6.9.1	Classes utilisées .....	60
6.9.2	MP3 .....	60
6.9.3	Importation .....	60
6.9.4	Tags ID3 .....	62
6.9.5	Analyse des tags .....	62

6.9.6	Indexation.....	63
6.9.7	Modification .....	64
6.9.8	Suppression .....	64
6.9.9	Vérification des données.....	65
6.9.10	Recherche .....	65
6.10	Listes de lecture.....	67
6.10.1	Classes utilisées .....	67
6.10.2	Génération de configuration .....	67
6.10.3	Création d'une playlist.....	68
6.10.4	Génération automatique.....	69
6.10.5	Ajout à une playlist .....	70
6.10.6	Retirer d'une playlist .....	71
6.10.7	Affichage du contenu .....	71
6.10.8	Suppression d'une playlist.....	72
6.11	Grille horaire.....	73
6.11.1	Classes utilisées .....	73
6.11.2	Outil utilisé .....	74
6.11.3	Gestion du calendrier par ShoutCAST .....	74
6.11.4	Génération de configuration .....	75
6.11.5	Affichage du calendrier .....	76
6.11.6	Sélection depuis le calendrier .....	78
6.11.7	Création d'un événement.....	78
6.11.8	Modification d'un événement.....	79
6.11.9	Suppression d'un événement.....	81
6.12	Transcoders .....	82
6.12.1	Classes utilisées .....	82
6.12.2	Outil utilisé .....	83

6.12.3	Définition des bitrates, taux d'échantillonnage et type d'encoder .....	84
6.12.4	Fichier de configuration et log.....	84
6.12.5	Licence MP3.....	87
6.12.6	Création d'un transcoder .....	87
6.12.7	Résolution de nom .....	88
6.12.8	Affichage d'un transcoder .....	88
6.12.9	Modification d'un transcoder.....	88
6.12.10	Suppression d'un transcoder.....	89
6.12.11	Exécution et fermeture .....	89
6.12.12	Administration web (weblet) .....	90
6.12.13	Capture live.....	90
6.12.14	Historique .....	91
6.13	Serveur de diffusion interne.....	95
6.13.1	Classes utilisées .....	95
6.13.2	Outil utilisé .....	96
6.13.3	Configuration.....	96
6.13.4	Mise à jour de la configuration .....	97
6.13.5	Exécution et fermeture .....	97
6.13.6	Statistiques et auditeurs.....	98
6.13.7	Affichage des interfaces web .....	100
6.14	Gestion des processus.....	101
6.14.1	IsRunning.....	101
6.14.2	Détection des crashes/fermetures externes.....	101
6.14.3	Fermeture automatique .....	102
7	Tests.....	103
7.1	Webradios .....	103
7.2	Status.....	103



7.3	Bibliothèque .....	103
7.4	Listes de lecture.....	104
7.5	Grille horaire.....	104
7.6	Transcoders .....	105
7.7	Serveur de diffusion .....	106
7.8	Processus.....	106
7.9	Autres .....	106
8	Conclusion .....	107
8.1	Bilan personnel.....	107
8.2	Améliorations possibles.....	108
9	Remerciements .....	108
10	Apports personnels .....	108
11	Glossaire .....	110
12	Illustrations.....	110
13	Références.....	112
14	Annexes .....	112

### 3 INTRODUCTION

#### 3.1 MISE EN SITUATION

Je m'appelle Simon Menetrey, j'ai 20 ans et je suis actuellement en dernière année de formation technicien ES en informatique. J'ai effectué un CFC d'informaticien en 4 ans avant de commencer ma formation actuelle.

Cette documentation concerne mon travail de diplôme réalisé pour ma dernière année en tant que technicien ES en informatique. Ce travail a pour sujet la création d'un gestionnaire de webradio.

Mon professeur de diplôme, Monsieur Garcia, m'a mis en contact avec une entreprise (KTFM) qui recherche un programme de gestion pour leur webradio. En effet, actuellement, c'est un tiers qui s'occupe de la diffusion de leur webradio via des émissions préalablement enregistrées dans leurs studios. Dans l'état actuel, l'entreprise n'a pas un contrôle direct sur la diffusion de son contenu et elle désirerait pouvoir gérer elle-même l'intégralité de leur webradio. Ainsi, elle supprimera un intermédiaire et aura pleinement contrôle de la diffusion.

En plus de cela, KTFM a aussi besoin d'avoir une traçabilité des morceaux qu'elle diffuse avec des informations précises afin de faciliter le paiement des droits d'auteur à la Suisa<sup>1</sup>.

J'ai donc réalisé le cahier des charges avec KTFM lors de deux rendez-vous afin de répondre au mieux à leurs besoins.



#### 3.2 SUJET

Gestionnaire de webradio :

- Permettre de diffuser directement depuis le logiciel
- Gérer les morceaux à diffuser/listes de lecture
- Gérer les plages horaires
- Historique de diffusion

#### 3.3 QU'EST-CE QU'UNE WEBRADIO ?

Une webradio est une radio diffusée sur internet via la technologie de lecture en continu. Cette technologie fournit ce que l'on appelle un « flux » que les auditeurs écoutent via leur lecteur multimédia préféré ou via un site web.

---

<sup>1</sup> SUISA est la coopérative des auteurs et éditeurs de musique <http://www.suisa.ch/fr/>

### 3.4 POURQUOI CE SUJET ?

Je suis passionné de musique. J'ai aussi toujours recherché un outil simple et gratuit pour gérer une webradio et sa diffusion. J'ai donc imaginé une application, tout-en-un, remplissant ce besoin. J'ai donc l'espoir que mon projet me sera autant utile à moi qu'à l'entreprise KTFM ainsi que de potentielles futures entreprises intéressées.

#### 4 CAHIER DES CHARGES

Ce projet a pour but la création d'un gestionnaire de webradio (de type shoutCAST<sup>2</sup>) complet. Les principales fonctionnalités sont les suivantes :

- Possibilité de gérer plusieurs webradios indépendamment
- Gestion des playlists, horaires et pubs
  - Génération automatique de playlist
  - Génération au format XML
- Gestion des serveurs de diffusions distants et transcoder<sup>3</sup> interne.
  - Diffusion de la webradio
- Indexation des fichiers musicaux (tags, chemin sur le disque dur)
- Historique des morceaux joués
  - Génération d'un compte-rendu afin de faciliter la gestion des droits d'auteurs
- Serveurs de diffusion interne (local)

Options :

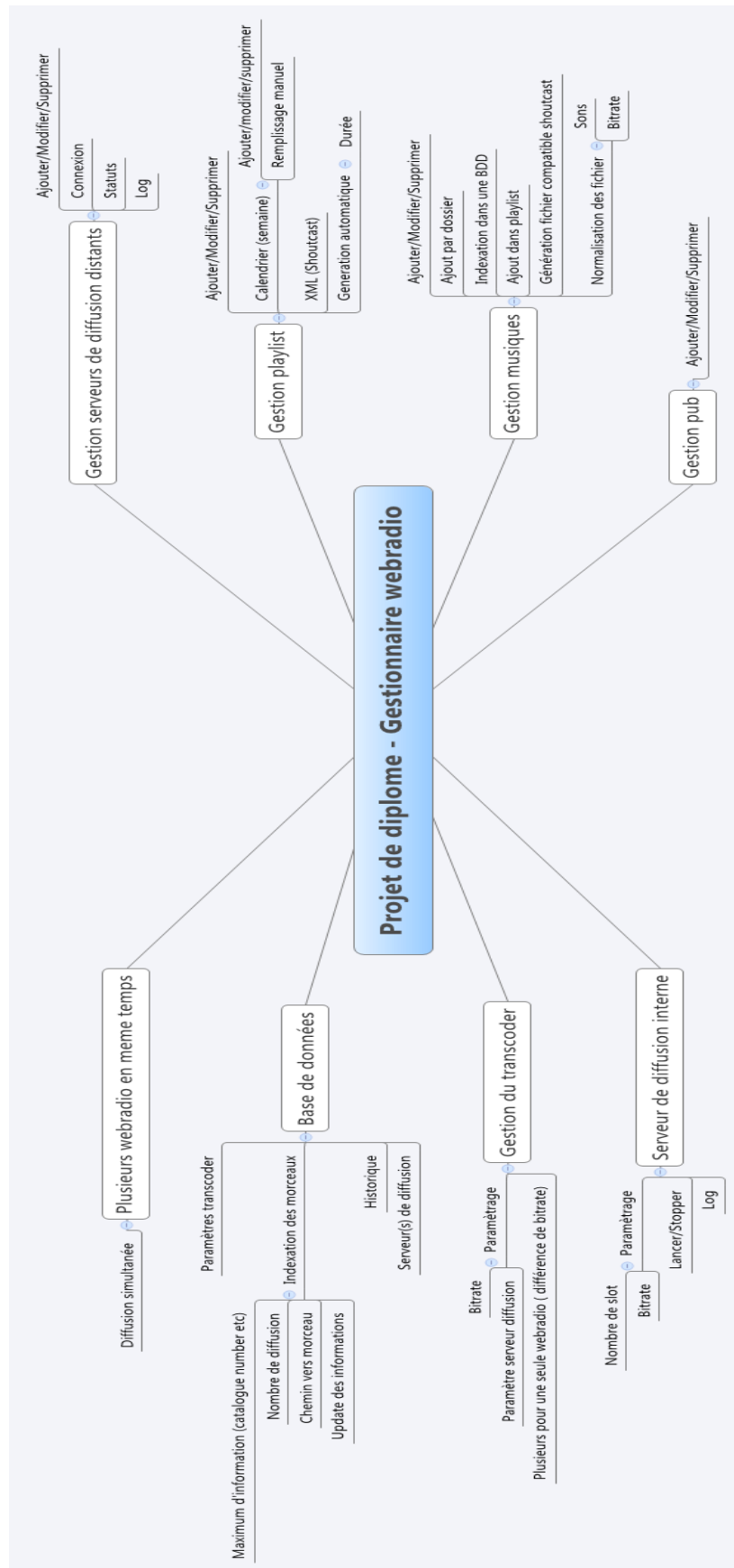
- Si serveur de diffusion interne activé : Serveur WEB interne contenant un mini-site
- modifier les informations des musiques indexées.

Plus de détails dans le résumé (1).

---

<sup>2</sup> Voir le chapitre concernant [ShoutCAST](#) dans l'analyse organique.

<sup>3</sup> Permet de créer un flux audio vers un serveur de diffusions



**Figure 1 - Mindmap (discussion avec KTMAAnalyses**

## 5 ANALYSE FONCTIONNELLE

### 5.1 PRINCIPE

Pour le principe de base d'une webradio, rendez-vous au chapitre concernant [ce sujet](#) dans l'analyse organique.

L'application a pour but de créer un flux audio qui sera récupéré par un serveur de diffusion et ce dernier s'occupera de diffuser le flux aux différents auditeurs. Un schéma sous forme d'une affiche est disponible en [annexe](#).

### 5.2 SCHÉMA DE L'APPLICATION

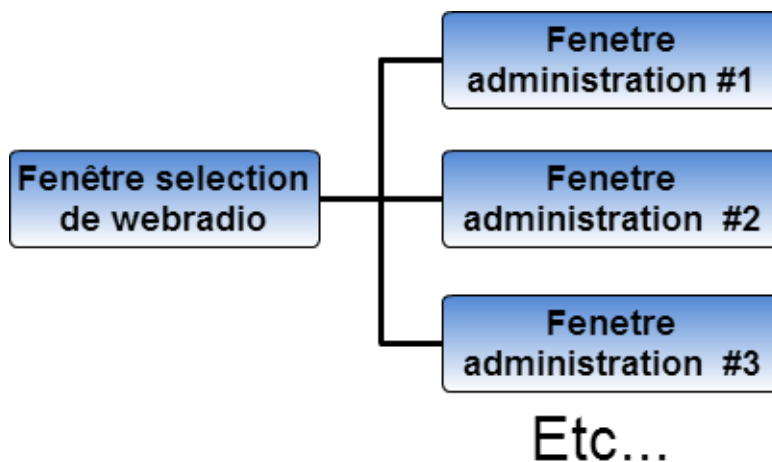


Figure 2 - Schéma application

Lors du lancement de l'application, la fenêtre de sélection de webradios s'affiche. L'utilisateur choisit la webradio qu'il veut gérer, puis une fenêtre d'administration s'ouvre avec les informations de la webradio sélectionnée. Il peut à tout moment réafficher la fenêtre de sélection et ouvrir une nouvelle fenêtre d'administration avec une autre webradio à gérer.

La diffusion de webradio est possible via des serveurs distants ou alors via le serveur interne (local) à l'application. Il y en a 1 par webradio.

## 5.3 INTERFACE PRINCIPALE

### 5.3.1 FENÊTRE D'ADMINISTRATION



Figure 3 - Interface principale AdminView

Cette fenêtre est l'interface principale de l'application. Elle est associée à une webradio. Comme montré dans le schéma de l'application (Figure 2), il est possible d'avoir plusieurs fenêtres d'administration ouvertes simultanément, une pour chaque webradio lancée. Les différents onglets sont décrits dans les chapitres suivants.

### 5.3.2 MENU PRINCIPAL

- File
  - Generate all configs : Génère la configuration de chaque élément de la webradio
  - Check library : Vérifie chaque fichier de la bibliothèque (si le fichier n'existe plus à l'emplacement spécifié, son enregistrement est supprimé).
- About : Affiche les informations sur le logiciel et son auteur

## 5.4 GESTION DE PLUSIEURS WEBRADIOS

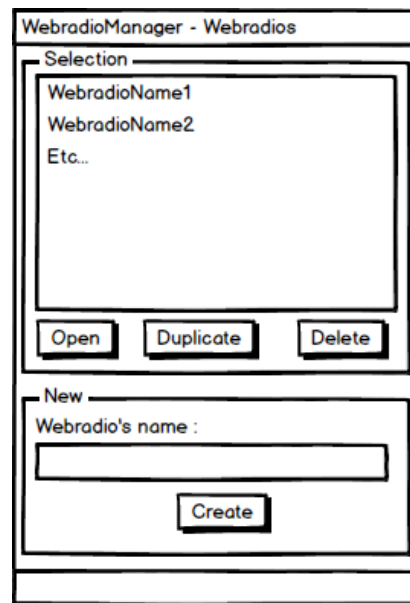


Figure 4 - Interface gestion des webradios SelectionView

La fenêtre de gestion des webradios s'affiche au démarrage de l'application pour sélectionner ou créer une webradio. Lorsque l'utilisateur clique sur « Open », la fenêtre principale s'ouvre avec les données de la webradio sélectionnée. Il peut aussi utiliser un double clic pour ouvrir une webradio.

Le bouton « Duplicate » crée une copie exacte de la webradio sélectionnée avec un préfix « copy of » à son nom. Enfin, le bouton « Delete » supprime la webradio sélectionnée.

La fenêtre n'est pas redimensionnable et ne peut être maximisée.

Concernant la création, la partie inférieure de la fenêtre propose la création d'une nouvelle webradio. L'utilisateur lui donne un nom, puis clique sur le bouton « Create ». Attention : Les webradios doivent avoir des noms différents. Un message d'erreur notifie l'utilisateur si une webradio a déjà le même nom.

L'utilisateur peut gérer autant de webradios en même temps qu'il le souhaite, en effet, à tout moment, il peut ouvrir la fenêtre de gestion des webradios et en sélectionner une autre. Cela ouvrira une nouvelle fenêtre principale, sans pour autant fermer les autres déjà ouvertes, avec les informations de la webradio fraîchement sélectionnée.

Chaque webradio possède ses propres listes de lecture, sa propre grille horaire ainsi que ses propres transcodeurs. Un transcodeur est un outil qui permet d'envoyer un flux musical à un serveur de diffusion. Pour plus d'informations, rendez-vous au chapitre concernant la [diffusion](#).



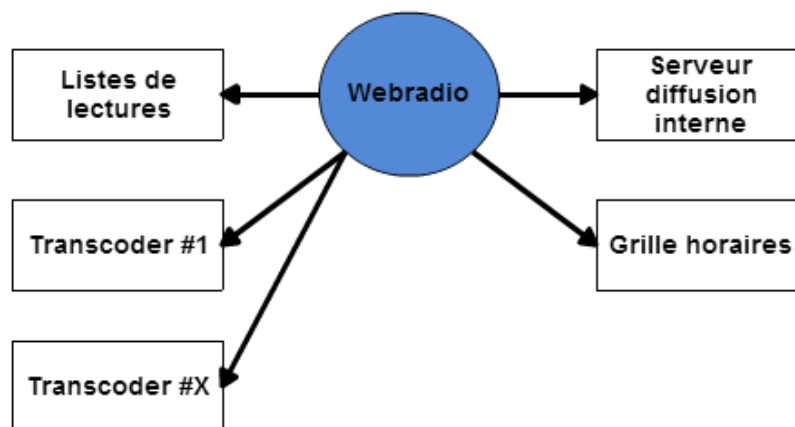
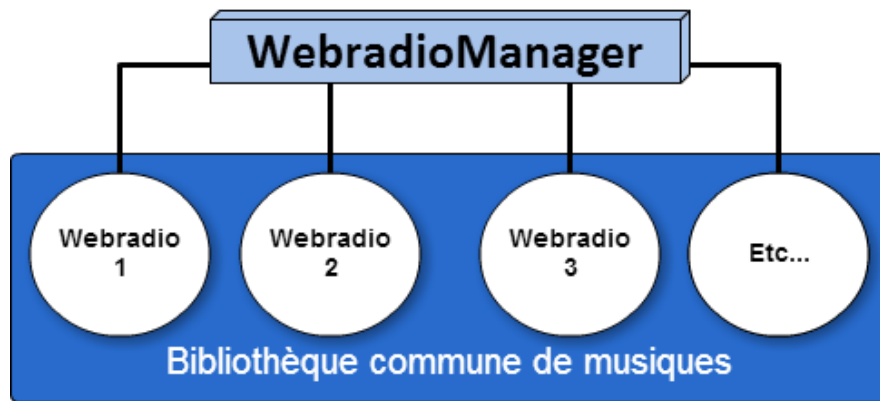


Figure 5 - Schéma webradios

#### 5.4.1 CRÉATION D'UNE WEBRADIO

L'utilisateur entre le nom de la future webradio dans le champ correspondant, puis clique sur « create » pour créer la webradio et ouvrir la fenêtre d'administration liée à la nouvelle webradio fraîchement créée. Le nom d'une webradio ne peut pas dépasser 255 caractères.

#### 5.4.2 SÉLECTION D'UNE WEBRADIO

L'utilisateur sélectionne une webradio parmi la liste proposée, puis clique sur « open » pour ouvrir une fenêtre d'administration liée la webradio sélectionnée.

#### 5.4.3 FERMETURE D'UNE WEBRADIO

Lors de la fermeture d'une fenêtre d'administration de webradio, l'utilisateur doit confirmer s'il est sûr de vouloir fermer la fenêtre en sachant que cela arrêtera le serveur de diffusion et les transcodeurs. Seule exception, si une autre *vue* qui pointe sur la même webradio est aussi affichée, l'utilisateur n'a pas besoin de confirmer, car les transcodeurs ne seront pas arrêtés dans ce cas. Ils sont seulement interrompus si la fenêtre fermée est la seule pointant sur la webradio en question.

## 5.5 SERVEUR DE DIFFUSION INTERNE ET EXTERNE

Il faut bien différencier les serveurs de diffusion qui sont externes et ceux internes. Les externes sont des serveurs hébergés par un provider<sup>4</sup> (exemple : infomaniak<sup>5</sup> pour KTFM). L'application va donc se servir de ces serveurs pour diffuser la webradio.

Les serveurs internes sont des serveurs de diffusion hébergés dans l'application elle-même (en local sur l'ordinateur).

---

<sup>4</sup> Fournisseur de services internet

<sup>5</sup> <http://www.infomaniak.com/>

## 5.6 ONGLET « STATUS »

Navigation tabs: Status, Library, Playlists, Timetable, Transcoders, Server

**WebradioName**

Name:

**Status**

Transcoder	Status
Transcoder1	Off
Transcoder2	On

**Server listeners**

Infos

Informations about server

Hostname	User agent	Connection time (s)	UID
12.0.0.1	VLC	50	1261456
...	...	...	...

**Current tracks**

Transcoder	Titre	Artist	Album
KTFM	Test	Test	Test
...	...	...	...

Figure 6 - Onglet "status"

L'onglet de statut est la page d'accueil de la webradio. L'utilisateur peut y modifier le nom de la webradio via le champ en dessous du nom. Si l'utilisateur change le nom, tous les transcodeurs et le serveur seront éteints.

Un tableau affiche l'état (allumé ou éteint) des différents transcodeurs de la webradio actuelle.

Dans la partie de droite, 2 tableaux :

- « Server listeners » : Affiche la liste des auditeurs/clients connectés au serveur local. Au-dessus, les informations concernant le nombre de connectés, le nombre maximum de connectés et le temps moyen de connexion sont affichés.
- « Current tracks » : Affiche le morceau actuellement joué par chaque transcodeur en fonctionnement.

## 5.7 GESTION DES MUSIQUES/PUBS

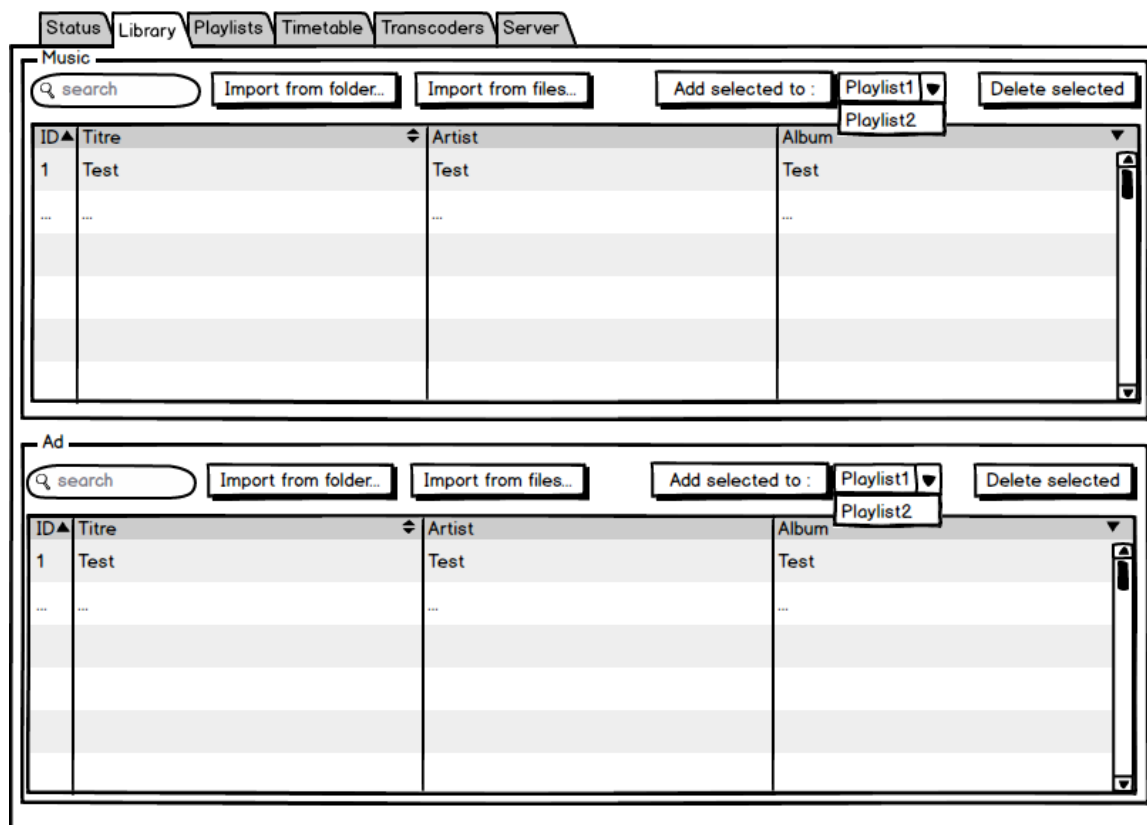


Figure 7 - Bibliothèque de musiques

Les musiques sont indexées pour former une bibliothèque commune dans l'application. Elle est commune, car elle est accessible depuis n'importe quelle webradio créée. L'interface est doublée : une partie pour les musiques et une autre pour les pubs. Elles sont identiques.

### 5.7.1 AFFICHAGE

La partie principale de cette interface est l'affichage du contenu de la bibliothèque dans la partie inférieure. Il est affiché sous la forme d'un tableau. Il est possible de sélectionner un ou plusieurs éléments.

### 5.7.2 IMPORTER ET INDEXER

L'utilisateur peut importer des fichiers musicaux dans sa bibliothèque (à l'heure actuelle, seuls les fichiers MP3 sont importables). Le bouton « Import from folder... » ouvre une boîte de dialogue où l'utilisateur peut sélectionner le dossier à analyser afin d'importer les fichiers musicaux qui y sont présents. C'est la partie dite « d'indexation ». Cette indexation peut être récursive, c'est-à-dire que les sous-dossiers du dossier sélectionné vont être analysés aussi. Une boîte de dialogue demande donc à l'utilisateur s'il veut effectuer une analyse récursive.

Le bouton « Import from files... » permet d'importer un ou plusieurs fichiers sélectionnés manuellement. La sélection s'effectue via une boîte de dialogue Windows standard.

À gauche de ces 2 boutons, une barre de recherche permet d'effectuer une recherche dans la bibliothèque de l'application.

Le bouton de droite « Delete selected » va supprimer les éléments sélectionnés dans la liste de morceaux affichés en dessous.

---

#### 5.7.3 AJOUT À UNE PLAYLIST

Pour ajouter des morceaux à une playlist, l'utilisateur peut en sélectionner autant qu'il le souhaite dans la liste d'affichage puis sélectionner une playlist via le menu déroulant situé à droite du bouton « Add selected to ». Ce dernier permet donc de confirmer l'ajout à une playlist.

---

#### 5.7.4 MODIFIER

L'utilisateur peut modifier les informations enregistrées dans la bibliothèque et, automatiquement, ces informations seront écrites dans le fichier MP3 lié. Les champs « id », « duration » et « path » ne sont pas modifiables. Pour modifier un champ (entrer en mode modification), l'utilisateur a plusieurs possibilités :

- Cliquer sur la cellule du champ qu'il désire modifier, puis commencer à écrire les modifications.
- Sélectionner une ligne, puis double cliquer sur la cellule à modifier.
- Sélectionner une cellule et appuyer sur la touche F2.

Dans tous les cas, l'utilisateur doit appuyer sur la touche « enter » pour valider.

Si, par exemple, le fichier musical n'existe plus à l'endroit indiqué, l'utilisateur en est averti.

---

#### 5.7.5 SUPPRIMER

L'utilisateur a la possibilité de supprimer une ou plusieurs musiques/pubs d'un seul coup grâce à la sélection multiple.

## 5.8 GESTION DES LISTES DE LECTURE

La gestion des différentes listes de lecture se fait dans l'onglet « playlists ».

Figure 8 - Onglet "playlists"

### 5.8.1 LISTES DE LECTURE MUSICALES

Une liste de lecture musicale contient exclusivement des musiques. Son type est « music ».

### 5.8.2 LISTES DE LECTURE PUBLICITAIRES

Une liste de lecture publicitaire contient des pubs, des annonces ou des jingles. Son type est « ad ».

### 5.8.3 CRÉATION

La création d'une playlist se fait via le cadre situé en haut à gauche. Il lui faut un nom et un type. La playlist fraîchement créée est ensuite ajoutée à l'une des 2 listes situées à gauche en fonction de son type. L'utilisateur ne peut pas créer de playlist avec le même nom par type de playlist (« Music » ou « Ad ») et par webradio.

### 5.8.4 GÉNÉRATION AUTOMATIQUE

L'utilisateur a la possibilité de générer automatiquement une playlist en définissant une durée (minimum 1 minute et maximum 500 minutes), un nom, un genre musical et un type. Si le type « Ad » est choisi, le menu déroulant « Gender » est désactivé (une pub ne peut pas avoir de genre musical).

La liste de genre affiche les genres disponibles dans la bibliothèque de l'application. Le logiciel va prendre différentes musiques ou publicité afin de remplir le temps voulu. L'algorithme fait en sorte de s'approcher le plus possible de la durée demandée, mais plus celle-ci est longue, plus cela sera possible d'être précis.

---

#### 5.8.5 AFFICHAGE DU CONTENU D'UNE PLAYLIST

Quand une playlist est sélectionnée (musique ou pub), son contenu est affiché dans la partie centrale de la *vue*. Les informations des morceaux sont affichées de même manière que pour l'onglet « [Library](#) ». Au-dessus de cette partie, des informations concernant la playlist sont affichées telles que le temps total de lecture, le nombre de morceaux présents, etc.

---

#### 5.8.6 RETIRER D'UNE PLAYLIST

L'utilisateur peut sélectionner un ou plusieurs morceaux dans la partie affichant le contenu de la playlist afin de les retirer de cette dernière. Une fois la sélection faite, le bouton « Remove selected » supprime le/les morceau(x) et le nouveau contenu de la playlist est affiché.

---

#### 5.8.7 RECHERCHE

L'utilisateur a la possibilité de recherche parmi les morceaux d'une playlist sélectionnée via le champ de recherche au-dessus de l'affichage du contenu de la playlist.

## 5.9 GESTION DES HORAIRES

Figure 9 - Onglet "timetable"

Figure 9 - Onglet "timetable"

	2 mer.	3 jeu.	4 ven.	5 sam.	6 dim.	7 lun.	8 mar.
07 00							
08 00							
09 00							
10 00	Playlist1						
11 00							
12 00	Playlist2						
13 00							
14 00							

Figure 10 - Calendrier

### 5.9.1 ÉVÉNEMENT

Un événement est un élément du calendrier. Il est composé des informations suivantes :

- Un nom
- Une playlist (nom de la playlist qui sera jouée à l'événement)
- Un ou plusieurs jours où la playlist sera jouée



- Une heure de début (identique pour chaque jour sélectionné)
- Une durée de lecture (la playlist est jouée en boucle pendant le temps défini)
- Une option pour jouer la playlist en mode aléatoire ou non
- Une priorité (dans le cas où plusieurs événements se superposent, celui avec la plus grande priorité sera joué). De 0 à 100.

---

### 5.9.2 ÉVÉNEMENT PÉRIODIQUE

Un événement périodique se produit à intervalles réguliers.

Cette fonctionnalité n'est à l'heure actuellement pas encore implémentée.

[http://wiki.winamp.com/wiki/SHOUTcast\\_Calendar\\_Event\\_XML\\_File\\_Specification#Time\\_Periodic\\_Events](http://wiki.winamp.com/wiki/SHOUTcast_Calendar_Event_XML_File_Specification#Time_Periodic_Events)

---

### 5.9.3 REMPLISSAGE MANUEL

L'utilisateur peut utiliser le formulaire de création d'événements en le remplissant à la main ou alors il a la possibilité de sélectionner sur le calendrier, la plage horaire où il désire créer un événement.

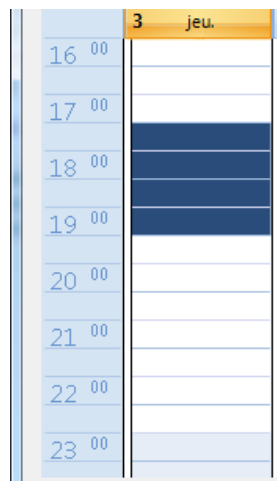


Figure 11 - Sélection multiple calendrier

La partie bleu foncé est la sélection faite par l'utilisateur. Dans ce cas, l'heure de début sera 17h30 et sa durée sera 2h, car la sélection finit à 19h30. Ces informations sont automatiquement affichées dans le formulaire de création. Évidemment, le reste des informations demandées sera à remplir à la main par l'utilisateur.

L'utilisateur peut aussi remplir manuellement les informations concernant le début et la durée d'un événement. La case à cocher « All » permet de cocher tous les jours de la semaine ou au contraire de les décocher. La case « shuffle » permet de choisir si l'événement lit la liste de lecture de façon aléatoire. « Priority » définit le niveau de priorité de l'événement par rapport à un autre qui serait prévu au même moment.

---

### 5.9.4 MODIFICATION D'UN ÉVÉNEMENT

	1	2	3	4	5	6	7
	lundi	mardi	mercredi	jeudi	vendredi	samedi	dimanche
06 00							
07 00			Test(0) Euphoric Shuffle : False				
08 00	Événement				Élément		
09 00	EuphoricTime(0) Euphoric Shuffle : False	EuphoricTime(0) Euphoric Shuffle : False	EuphoricTime(0) Euphoric Shuffle : False	EuphoricTime(0) Euphoric Shuffle : False	EuphoricTime(0) Euphoric Shuffle : False		
10 00							
11 00							
12 00							
13 00							

Figure 12 - Définition d'un événement avec des éléments

L'utilisateur peut modifier le début, la durée et les jours d'un événement. Pour se faire, il peut déplacer les « zones » (éléments) d'un événement pour le placer où bon lui semble. Il existe malgré cela des règles :

- Il ne peut pas y avoir plus d'un élément d'un événement dans le même jour.
- Tous les éléments commencent et finissent à la même heure. Si un élément est changé dans sa durée, cela sera appliqué à tous les autres éléments de cet événement.

L'utilisateur doit redémarrer ses transcodeurs pour que la nouvelle configuration soit prise en compte. Le programme en cours est donc interrompu et reprendra selon le nouveau calendrier après redémarrage.

### 5.9.5 SUPPRESSION D'UN ÉVÉNEMENT

Pour supprimer un événement, l'utilisateur doit faire clic droit sur l'événement en question. Une boîte de dialogue lui demande s'il est sûr de vouloir supprimer cet événement. La suppression d'un élément de l'événement entraîne la suppression complète de ce dernier.

## 5.10 GESTION DES TRANSCODEURS

Figure 13 - Onglet "Transcoders"

Cet onglet permet de gérer les différents transcodeurs d'une webradio. Ces derniers servent à diffuser le flux audio vers un serveur de diffusion. Ils utilisent les playlists et le calendrier configuré par l'utilisateur.

### 5.10.1 CRÉATION

La partie de gauche offre la possibilité de créer un nouveau transcoder avec ses différents réglages. Pour des mesures de simplicité, le minimum requis est demandé à l'utilisateur. Les informations suivantes sont demandées :

- Le type de flux (mp3 ou ACC+)
- Le bitrate<sup>6</sup> du flux
- Le sample rate<sup>7</sup>
- Le nom du flux (qui sert de nom pour le transcoder)
- L'URL du flux (par exemple : site web de la webradio) : pas obligatoire
- Un port d'administration

<sup>6</sup> Débit binaire : une mesure de la quantité de données numériques transmises par unité de temps.

<sup>7</sup> Taux d'échantillonnage

- Adresse IP du serveur (ou le nom dns via bouton « resolve »)
- Port du serveur
- Mot de passe du serveur

Le bouton « resolve » permet à l'utilisateur d'entrer le nom de domaine du serveur puis de cliquer sur ce bouton pour trouver l'adresse IP liée. Si elle est trouvée, elle sera affichée dans le champ à la place du nom DNS.

---

#### 5.10.2 AFFICHAGE

Dans la partie droite, la liste des transcodeurs de la webradio est affichée. L'utilisateur peut cliquer et sélectionner un des transcoder afin de la supprimer à l'aide du bouton « Delete ». Les informations du transcodeur sélectionné sont affichées à droite de la liste, dans les champs prévus à cet effet. Le statut (on ou off) est affiché en dessous de ces dernières. Tout en bas, le log du transcoder est affiché et peut être effacé à l'aide du bouton « Clear ».

---

#### 5.10.3 MODIFICATION

Les informations affichées dans les champs à droite peuvent être modifiées puis enregistrées (bouton « Update ») afin de mettre à jour les paramètres du transcoder sélectionné.

---

#### 5.10.4 CONTRÔLES

La partie « status » permet de contrôler le transcodeur sélectionné. La case à cocher « Debug » permet de lancer le serveur avec son log dans une fenêtre console.

Si une erreur est détectée lors du lancement du transcoder, l'utilisateur en est informé. Il est possible que le transcoder en question soit déjà lancé dans un autre processus ou alors que le fichier exécutable ne soit plus présent.

Le bouton « next » (qui est noté « next track » sur l'interface finale) permet à l'utilisateur de dire au transcoder de passer à la musique suivante dans la playlist qui est actuellement jouée par le transcoder sélectionné.

Attention, il est possible que la configuration du transcoder demande à être régénérée pour bien fonctionner. L'utilisateur ne doit pas hésiter à utiliser le bouton « update » pour régénérer la configuration du transcodeur sélectionné.

---

#### 5.10.5 CAPTURE LIVE

L'utilisateur peut choisir de couper la lecture du calendrier (et donc des playlists) pour capturer le son, en provenance de son micro ou de sa carte son, et le diffuser via le transcoder en question.

Dans la partie « Live capture », la liste des différents périphériques audio de l'ordinateur est affichée. Le bouton à sa droite permet de rafraichir la liste. Une fois un périphérique sélectionné et le transcoder lancé, l'utilisateur peut démarrer la capture avec le bouton « Start » et la stopper avec « Stop ».

À savoir que cette fonctionnalité n'est pas encore tout à fait au point et peut présenter des bugs.  
Vérifier le log lors de l'activation.

#### 5.10.6 HISTORIQUE DE DIFFUSION

L'historique de diffusion est enregistré (quelle musique est passée à quelle heure) et l'utilisateur peut générer un fichier de type PDF avec le bouton « generate ». Il lui sera demandé l'emplacement désiré pour l'enregistrement du fichier. Il est aussi possible de vider l'historique avec le bouton « clear ».

#### 5.11 GESTION DU SERVEUR DE DIFFUSION

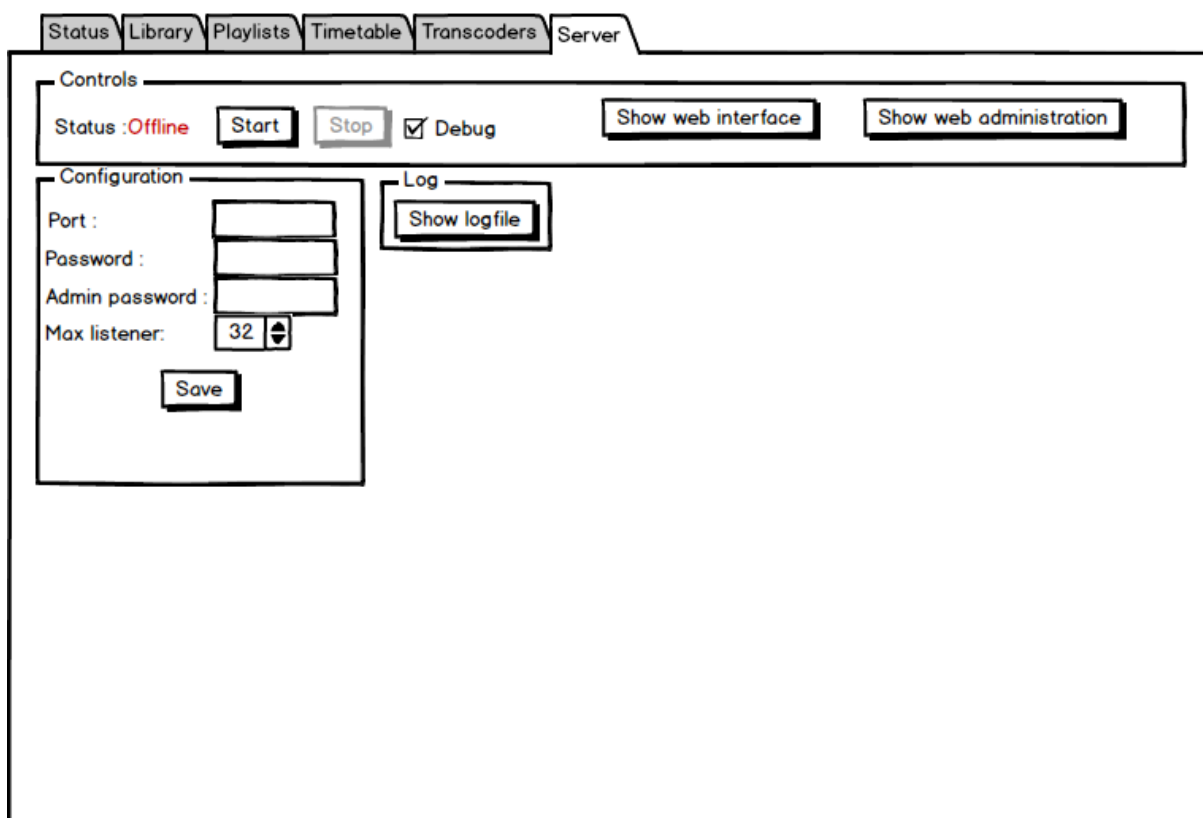


Figure 14 - Onglet "server"

L'onglet serveur propose un serveur de diffusion interne (local) pour la webradio. Bien entendu, l'utilisateur doit créer un transcoder qui s'y connectera afin de diffuser le flux audio.

Le serveur de diffusion permet de distribuer un flux audio à des auditeurs.

##### 5.11.1 CONTRÔLES

La partie « controls » permet de démarrer ou arrêter le serveur de la webradio via les boutons « start » et « stop ». La case à cocher « Debug » permet de lancer le serveur avec son log dans une fenêtre console. À sa droite, un bouton permet de lancer l'interface web et un autre l'administration web via le navigateur par défaut de l'utilisateur.

Si une erreur est détectée lors du lancement du serveur, l'utilisateur en est informé. Il est possible que le serveur soit déjà lancé dans un autre processus ou alors que le fichier exécutable ne soit plus présent.

#### 5.11.2 LOG

Si l'utilisateur lance le serveur en mode « debug », il peut avoir un log sous forme de programme console qui apparaît dans une fenêtre minimisée. Le bouton « Show logfile » permet d'afficher le fichier de log du serveur dans le bloc note.

#### 5.11.3 CONFIGURATION

Différents éléments sont configurables pour le serveur de diffusion :

- Port : le port de connexion au serveur pour le transcoder (Défaut : 8000)
- Password : Le mot de passe de connexion au serveur pour le transcoder
- Admin password : Le mot de passe pour l'accès à l'administration web du serveur
- Max listener : Le nombre maximum de connexions au serveur (connexion client/auditeur).  
Par défaut : 1. Maximum : 2000.

Attention : Les 2 mots de passe doivent être différents !

#### 5.11.4 INTERFACE WEB

Une interface web est fournie avec le serveur de diffusion. L'outil utilisé pour créer le serveur génère tout seul cette interface ainsi que les informations qui y figurent. Cette interface permet de voir différentes informations sur le serveur et le stream. Il est aussi possible de télécharger le fichier permettant d'écouter le flux directement sur un lecteur multimédia.

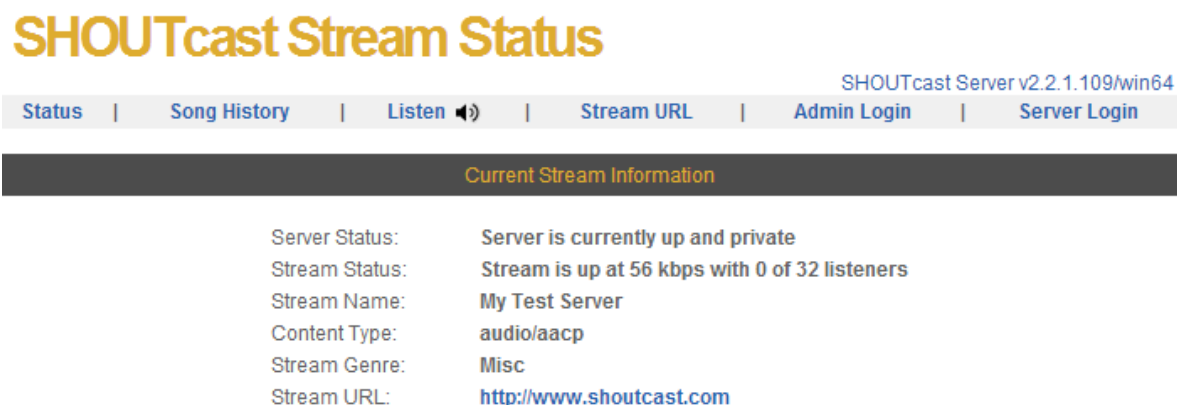


Figure 15 - Interface web ShoutCAST serveur

Un historique des morceaux joué par le serveur est aussi disponible.

#### 5.11.5 ADMINISTRATION WEB

Via l'interface web décrite précédemment, il est possible d'accéder à une section d'administration. Pour se faire, il faut cliquer sur le lien « Admin login » puis entrer le nom d'utilisateur « admin » et le mot de passe configuré pour le serveur.

**SHOUTcast Listeners and Status**

Uptime: 14 minutes 29 seconds SHOUTcast Server v2.2.1.109/win64

[Listeners](#) | [Log \(Tailing | Save\)](#) | [Song History](#) | [Ban List](#) | [Reserved List](#) | [Admin Logout](#) | [Server Login](#)

---

**Current Stream Information**

Stream Details	Server Status:
Log file: <code>sc_serv.log</code>	Server is currently up and private 🎧
Configuration file: <code>examples/sc_serv_basic.conf</code>	Stream Status: Stream is up at 56 kbps with 0 of 32 listeners
Intro file is empty	Stream Name: My Test Server
Backup file is empty	Content Type: audio/aacp
Idle timeouts are 30s	Stream Genre: Misc
Source connection type: v2	Stream URL: <a href="http://www.shoutcast.com">http://www.shoutcast.com</a>
Stream artwork not available	Stream Source: 127.0.0.1 [ kick ]
Playing artwork not available	Stream Uptime: 22 seconds

Figure 16 - Administration web ShoutCAST serveur

Pour l'administrateur, il est possible de :

- Voir la liste des « listeners » (clients qui écoutent la webradio depuis ce serveur)
- Voir le log du serveur
- Gérer une liste d'adresses IP bannies
- Gérer une liste d'adresses IP qui disposent d'un accès réservé (si une adresse réservée désire écouter le webradio, mais que le serveur est plein, un client sera éjecté du serveur pour laisser une place au client disposant d'une adresse réservée)

Tous ces services sont fournis par l'interface d'administration web.

## 6 ANALYSE ORGANIQUE

### 6.1 ENVIRONNEMENT

- Langage : C#/.NET
- IDE : Visual Studio 2013
- OS : Windows 7 64 bits

### 6.2 DIAGRAMME DE CLASSES

#### 6.2.1 MODEL

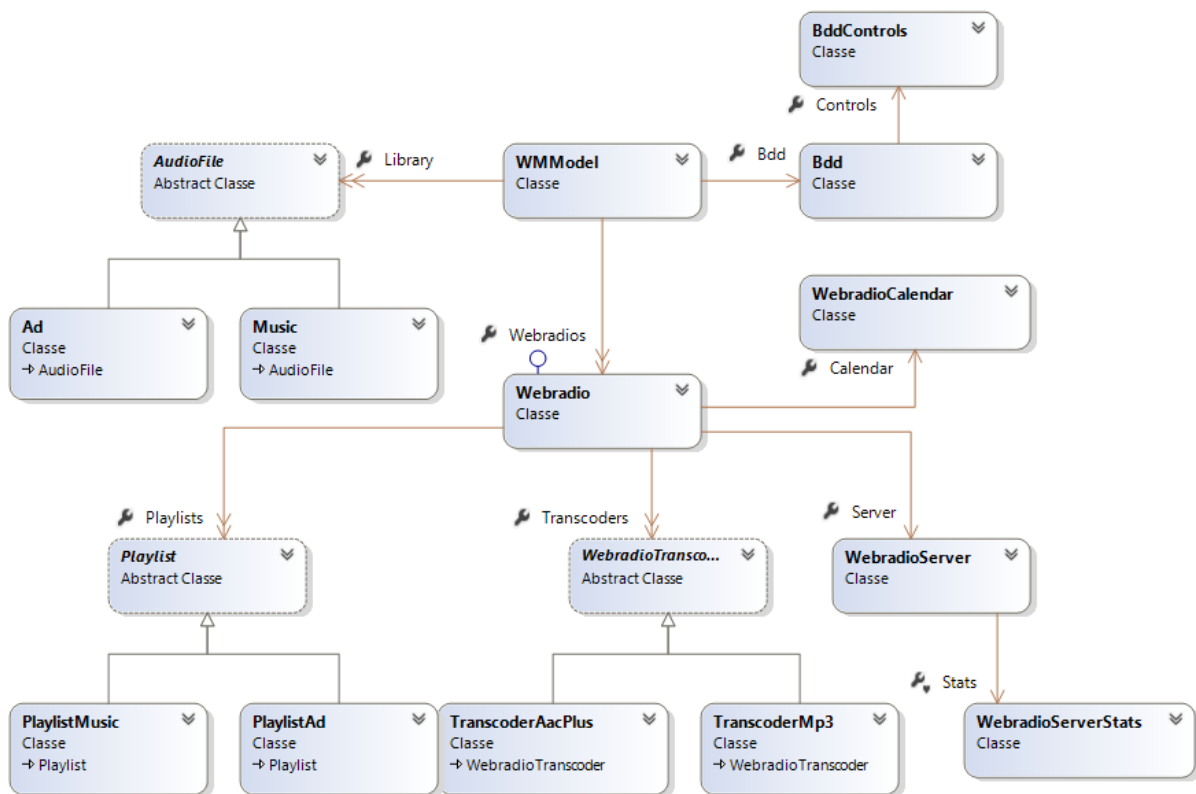


Figure 17 - Diagramme model

#### 6.2.2 MVC

L'application est codée en suivant le *modèle* (patron) MVC

(<http://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur>).



## MVC Architecture (basic)

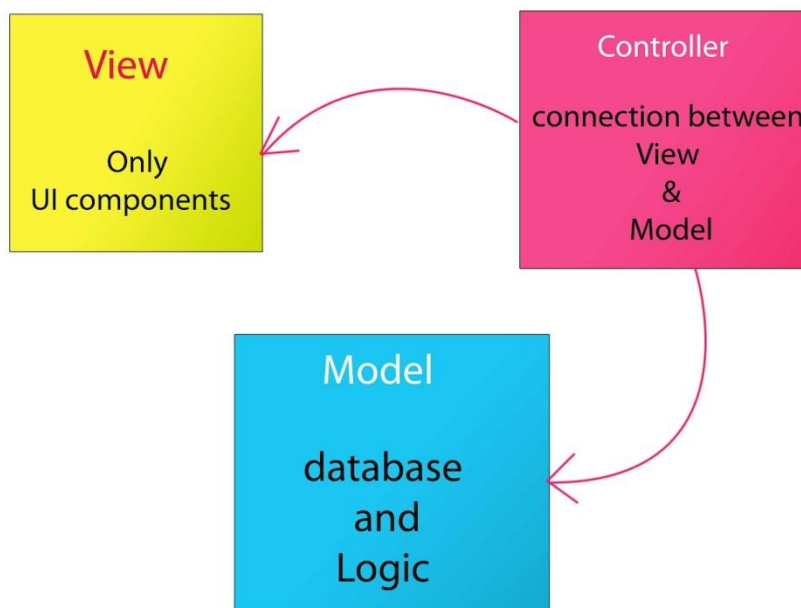


Figure 18 - MVC

L'application dispose de 2 *vues* :

- AdminView : *Vue* d'administration de webradio
- SelectionView : *Vue* de gestion des webradios

Chacune des *vues* à son *controller* : AdminController et SelectionController.

Le *model* est décrit dans le chapitre précédent. C'est la classe nommée « WMMModel ». Elle contient toute la logique et les différentes données du programme.

### 6.2.3 OBSERVATEURS/SUJET

Le patron de conception observateur/observable est utilisé en programmation pour envoyer un signal à des modules qui jouent le rôle d'observateur. En cas de notification, les observateurs effectuent alors l'action adéquate en fonction des informations qui parviennent depuis les modules qu'ils observent (les « observables »).

Source : Wikipédia

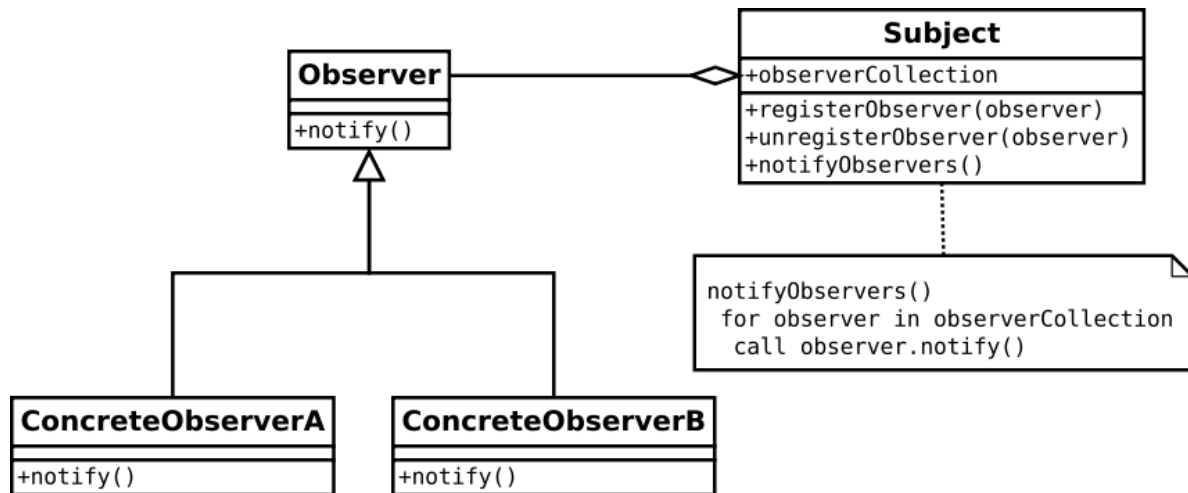


Figure 19 - Pattern observateurs/sujet

Le schéma ci-dessus explique le principe de ce patron de conception. Le sujet dispose d'une liste d'observateurs qui s'inscrivent via la méthode « `registerObserver` » et se désinscrivent avec « `unregisterObserver` ». La méthode « `notifyObservers` » parcourt la liste et appelle la méthode « `notify` » de chaque observateur. Les observateurs peuvent être de différents types, c'est pour cela qu'une interface « **Observer** » est créée afin que la méthode « `notifyObservers` » puisse appeler « `notify` » sans prendre en compte le type réel de l'observateur.

Dans ce projet, le sujet est le *model* et les observateurs sont les *controllers* des différentes *vues* du projet. Ces derniers vont ensuite appeler la méthode `UpdateView` de leur *vue* respective. L'interface `IController` est égale à « **Observer** » dans le schéma précédent. La méthode « `UpdateView` » correspond à « `notify` ». Dans le *model*, c'est la méthode « `UpdateObservers` » qui s'occupe d'appeler la méthode « `UpdateView` » de chaque contrôleur.

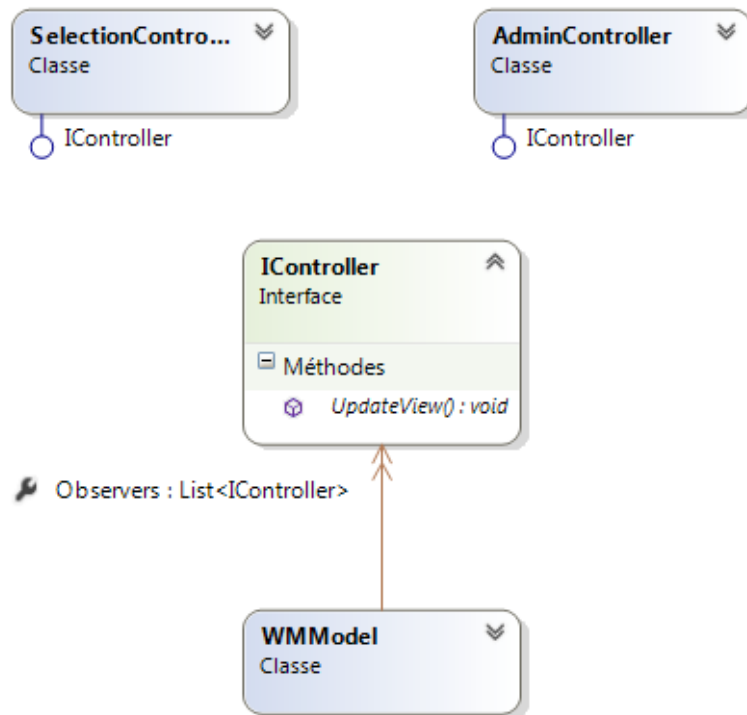


Figure 20 - Observateurs/sujet dans WebradioManager

Il faut noter que le *model* dispose de 2 méthodes « UpdateObservers ». L'un n'a aucun paramètre et l'autre prend l'identifiant d'une webradio. La deuxième permet de mettre à jour seulement les observateurs qui sont liés à la webradio dont l'identifiant est donné en paramètre.

J'ai utilisé ce système, car il permet de lancer plusieurs fenêtres d'administration en même temps (qu'elles soient liées à la même webradio ou non). De ce fait, lorsque l'une apporte des modifications au *model*, les autres sont informées et se mettent à jour.

#### 6.2.4 AUDIOTYPE ET STREAMTYPE

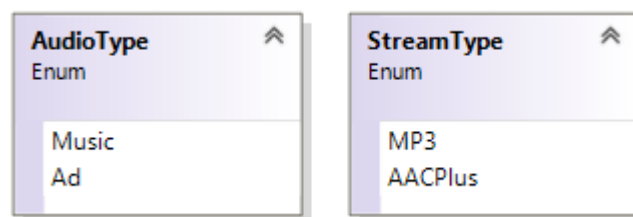


Figure 21 - AudioType et StreamType enum

Le mot clé enum est utilisé pour déclarer une énumération, c'est-à-dire un type distinct constitué d'un ensemble de constantes nommées que l'on appelle « liste d'énumérateurs ».

Source : MSDN

Ces 2 enums contiennent comme valeur, l'identifiant statique de leur valeur dans la base de données. C'est-à-dire, par exemple, que la valeur de la constante « Music » dans l'enum « AudioType » est égale à l'identifiant correspond dans la table « taudiotype » de la base de données.

---

#### 6.2.5 STOCKAGE DES WEBRADIOS

Les données des webradios contenues dans la base de données sont chargées dans le *model* au lancement de l'application.

Ces données sont stockées sous la forme d'un dictionnaire (Dictionary<int, Webradio>). Le fait d'utiliser un dictionnaire avec comme valeur clé, une entier, permet d'y stocker l'identifiant de la webradio référencée comme valeur. Ainsi, la recherche de données dans le *model* est facilitée et accélérée.

## 6.3 BASE DE DONNÉES

La base de données est présente pour sauvegarder les diverses informations de l'application. La plupart des paramètres sont stockés dans des fichiers texte sous forme de configuration utilisable par les différents transcodeurs et serveurs.

### 6.3.1 SQLITE



Figure 22 - Logo SQLite

SQLite est une bibliothèque écrite en C qui propose un moteur de base de données relationnelle accessible par le langage SQL. SQLite implémente en grande partie le standard SQL-92 et des propriétés ACID (voir glossaire 9).

Contrairement aux serveurs de bases de données traditionnels, comme MySQL ou PostgreSQL, sa particularité est de ne pas reproduire le schéma habituel client-serveur, mais d'être directement intégrée aux programmes. L'intégralité de la base de données (déclarations, tables, index et données) est stockée dans un fichier indépendant de la plateforme.

Source : Wikipédia (<http://fr.wikipedia.org/wiki/SQLite>)

Pour mon projet, j'utilise le logiciel SQLite Studio pour éditer ma base de données et y entrer des données de test : <http://sqlitestudio.pl/>

J'ai choisi SQLite, car il est léger et rapide. Il permet aussi d'avoir une base de données locale simple d'utilisation et multiplateforme.

### 6.3.2 UTILISATION

Une bibliothèque sous forme d'une DLL<sup>8</sup> est disponible pour .NET (C#) ici : <https://system.data.sqlite.org/index.html/doc/trunk/www/downloads.wiki>

Une classe fournie par le site web suivant : <http://www.dreamincode.net/forums/topic/157830-using-sqlite-with-c%23/#/> permet l'interaction avec un fichier SQLite. Cette classe prendra la place de BddControls dans le diagramme de classe de l'application. Le fichier de base de données « database.db » est à la racine du logiciel. Une copie « vierge » (sans données à l'intérieur, mise à

---

<sup>8</sup> Dynamic Link Library

part les tables taudiotype et tstreamtype : voir 0) de la base de données est stockée dans les ressources du logiciel afin d'avoir une BDD de base.

La classe Bdd utilise BddControls (voir diagramme de classe **Erreur ! Source du renvoi introuvable.**) Afin d'effectuer des requêtes sur la base de données. Bdd propose des méthodes simples comme par exemple « AddWebradio ». Ainsi la partie traitement des données se fait dans Bdd et la partie exécution dans BddControls.

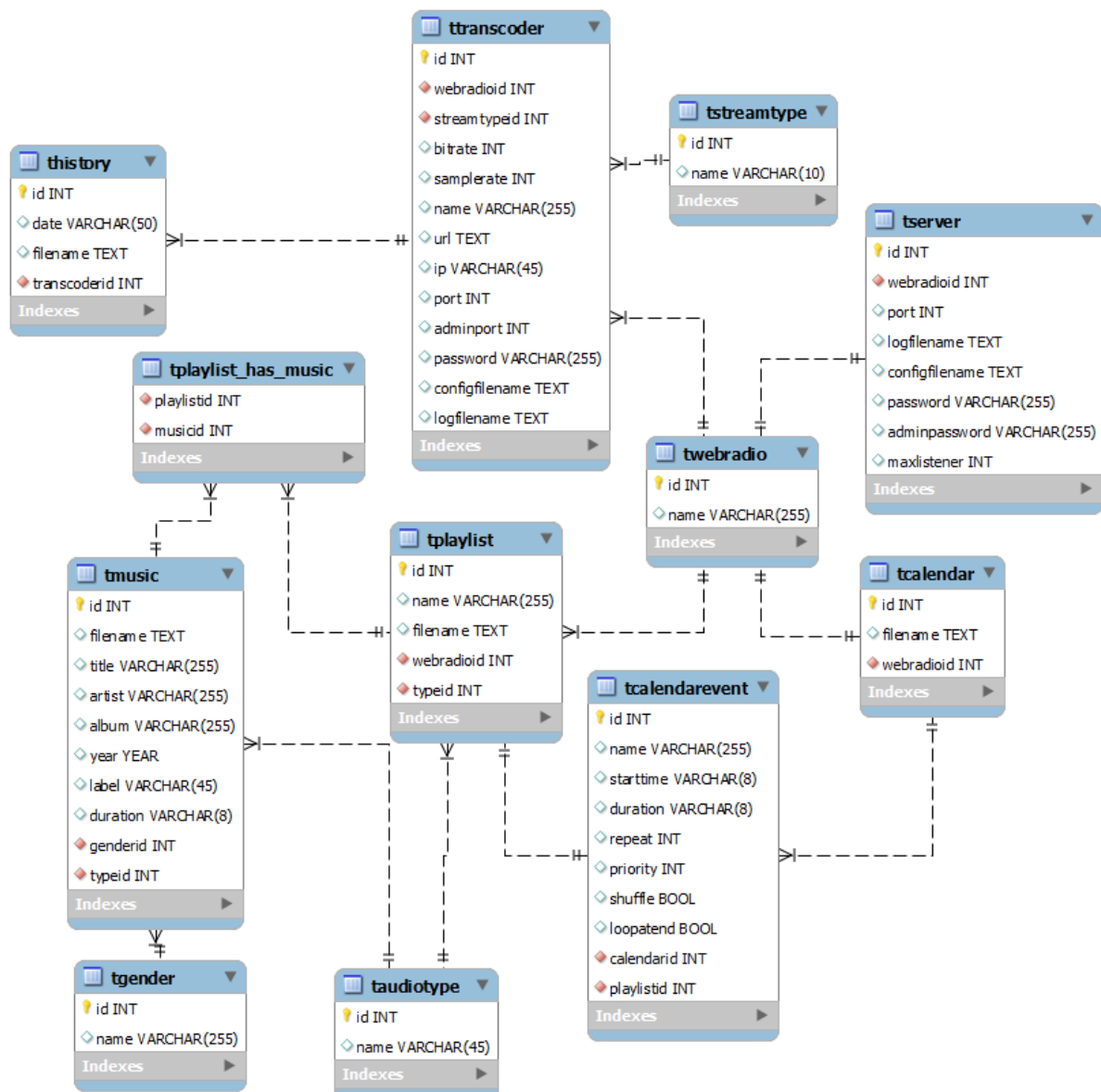
La base de données sert principalement à la sauvegarde des données qui sont récupérées dans le *model* à chaque démarrage de l'application.

---

### 6.3.3 SUPPRESSION EN CASCADE

SQLite permet la suppression en cascade. C'est-à-dire, lorsqu'un enregistrement est supprimé, toutes les références à ce dernier via des clés étrangères sont supprimées automatiquement.

### 6.3.4 SCHÉMA



### 6.3.5 TWEBRADIO

Cette table contient la liste des webradios qui est composée de leur nom.

Nom	Type	Description
<b>Id</b>	Int	Identifiant unique d'une webradio
<b>Name</b>	Varchar(255)	Nom de la webradio

### 6.3.6 TSERVER

Cette table contient les informations concernant les différents serveurs. Un enregistrement est lié à une webradio via une clé étrangère.

Nom	Type	Description
<b>Id</b>	Int	Identifiant unique d'un serveur
<b>Webradioid</b>	Int	Clé étrangère (radio possédant ce serveur)
<b>Port</b>	Int	Numéro du port d'écoute du serveur0
<b>Logfilename</b>	Text	Chemin vers le fichier de log du serveur
<b>Configfilename</b>	Text	Chemin vers le fichier de configuration du serveur
<b>Password</b>	Varchar(255)	Mot de passe de connexion au serveur (pour une source)
<b>Adminpassword</b>	Varchar(255)	Mot de passe de l'administration web du serveur
<b>Maxlistener</b>	Int	Le nombre maximal d'auditeurs sur le serveur



### 6.3.8 TCALENDAR

Cette table contient les informations pour un calendrier. Un enregistrement est lié à une webradio via une clé étrangère.

Nom	Type	Description
<b>Id</b>	Int	Identifiant unique d'un calendrier
<b>Filename</b>	Text	Chemin vers le fichier XML du calendrier
<b>Webradioid</b>	Int	Clé étrangère (radio possédant ce calendrier)

### 6.3.9 TCALENDAREVENT

Cette table contient les différents événements d'un calendrier. Un événement est lié à un calendrier via une clé étrangère et contient une playlist. Le lien vers cette playlist est aussi effectué avec une clé étrangère vers la table tplaylist.

Nom	Type	Description
<b>Id</b>	Int	Identifiant unique d'un événement
<b>Name</b>	Varchar(45)	Nom de l'événement
<b>Starttime</b>	Varchar(8)	Heure du commencement de l'événement
<b>Duration</b>	Varchar(8)	Durée de l'événement
<b>Repeat</b>	Int	Valeur de répétition de l'événement (voir chapitre <a href="#">Grille horaire</a> )
<b>Priority</b>	Int	Priorité de l'événement
<b>Shuffle</b>	Bool	Défini si l'événement lit la playlist de façon aléatoire
<b>Loopatend</b>	Bool	Défini si la playlist recommence une fois que tous les morceaux sont écoutés
<b>Calendarid</b>	int	Clé étrangère (Calendrier possédant cet événement)
<b>Playlistid</b>	Int	Clé étrangère (Playlist jouée par cet événement)

### 6.3.10 TPLAYLIST

Cette table contient les informations sur une playlist (mais pas les fichiers audio qui lui sont liés). Un enregistrement est lié à une webradio via une clé étrangère.

Nom	Type	Description
<b>Id</b>	Int	Identifiant unique d'une playlist
<b>Name</b>	Varchar(255)	Nom de la playlist
<b>Filename</b>	Text	Chemin vers le fichier de la playlist
<b>Webradioid</b>	Int	Clé étrangère (webradio possédant cette playlist)
<b>Typeid</b>	Int	Clé étrangère (le type de la playlist)

### 6.3.11 TAUDIOTYPE

Cette table est liée à l'enum « AudioType », car les valeurs qui y sont enregistrées sont codées en « dur » dans l'enum. Elle définit les types de fichier audio possible (musique ou publicitaire à l'heure actuelle).

Nom	Type	Description
<b>Id</b>	Int	Identifiant unique d'un type audio
<b>Name</b>	Varchar(45)	Nom du type audio

### 6.3.12 TMUSIC

Cette table contient la bibliothèque de fichiers musicaux indexés. Un enregistrement est lié à un type de fichier audio via une clé étrangère ainsi qu'à un genre de la table tgender.

Nom	Type	Description
<b>Id</b>	Int	Identifiant unique d'une musique
<b>Filename</b>	Text	Chemin vers le fichier musical
<b>Title</b>	Varchar(255)	Titre du morceau
<b>Artist</b>	Varchar(255)	Artiste du morceau
<b>Album</b>	Varchar(255)	Album du morceau

<b>Year</b>	Year	Année du morceau
<b>Label</b>	Varchar(45)	Label du morceau
<b>Duration</b>	Varchar(8)	Durée du morceau
<b>Genderid</b>	Int	Clé étrangère (genre du morceau)
<b>Typeid</b>	Int	Clé étrangère (type du morceau)

#### 6.3.13 TGENDER

Cette table contient la liste des genres des fichiers musicaux contenus dans la table tmusic.

Nom	Type	Description
<b>Id</b>	Int	Identifiant unique d'un genre musical
<b>Name</b>	Varchar(255)	Nom du genre

#### 6.3.14 TPLAYLIST\_HAS\_MUSIC

Cette table permet de lier un fichier audio à une playlist via les identifiants de leur enregistrement.

Nom	Type	Description
<b>Playlistid</b>	Int	Clé étrangère (Playlist concernée)
<b>Musicaid</b>	Int	Clé étrangère (Musique concernée, qui fait partie de la playlist ayant l'identifiant Playlistid)

### 6.3.15 THISTORY

Cette table contient l'historique (sous forme de nom de fichier) d'un transcoder (dont la liaison est effectuée avec une clé étrangère).

Nom	Type	Description
<b>Id</b>	Int	Identifiant unique d'un élément d'historique
<b>Date</b>	Varchar(50)	Date de l'événement dans l'historique
<b>Filename</b>	Text	Chemin vers le fichier joué
<b>Transcoderid</b>	Int	Clé étrangère (Transcodeur ayant joué cette musique)

### 6.3.16 TTRANSCODER

Cette table contient les informations sur les différents transcodeurs. Un transcoder est lié à une webradio via une clé étrangère. Une autre clé étrangère permet de définir le type de stream (table tstreamtype). À l'heure actuelle, il y a 2 types : MP3 et AAC+.

Nom	Type	Description
<b>Id</b>	Int	Identifiant unique d'un transcoder
<b>Webradioid</b>	Int	Clé étrangère (Webradio possédant ce transcoder)
<b>Streamtypeid</b>	Int	Clé étrangère (Type du transcodeur MP3 ou autre)
<b>Bitrate</b>	Int	Débit binaire du flux (en bits/s)
<b>Samplerate</b>	Int	Taux d'échantillonnage du flux
<b>Name</b>	Varchar(255)	Nom du flux
<b>Url</b>	Text	URL concernant le flux (site web par exemple)
<b>Ip</b>	Varchar(45)	Adresse IP du serveur de diffusion
<b>Port</b>	Int	Port du serveur de diffusion
<b>Adminport</b>	Int	Port d'administration du transcoder

<b>Password</b>	Varchar(255)	Mot de passe du serveur de diffusion
<b>Configfilename</b>	Text	Chemin vers le fichier de configuration du transcodeur
<b>Logfilename</b>	Text	Chemin vers le fichier de log du transcodeur

## 6.4 SCHÉMA DE DIFFUSION

### 6.4.1 PRINCIPE DE BASE

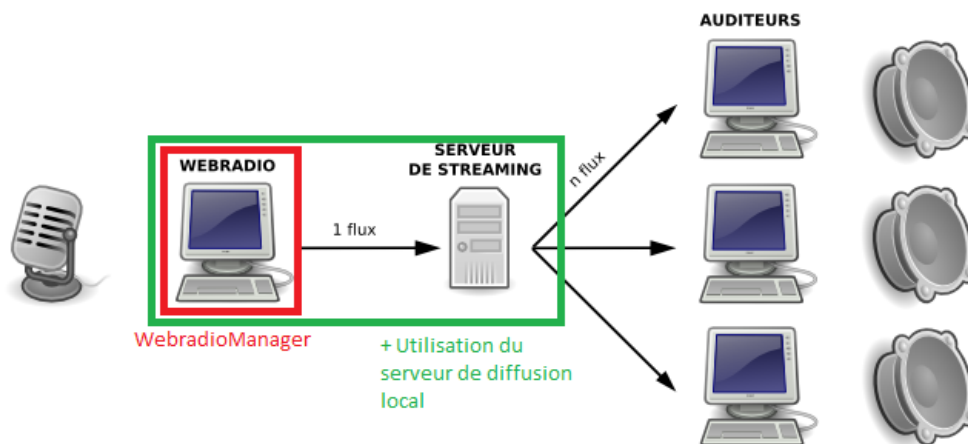


Figure 23 - Principe de base de diffusion

Ce type de diffusion est appelé « client-serveur ». C'est le principe de base de streaming audio/vidéo. Dans WebradioManager, la diffusion est possible sur des serveurs distants ou sur un serveur interne (local).

### 6.4.2 INFOMANIAK

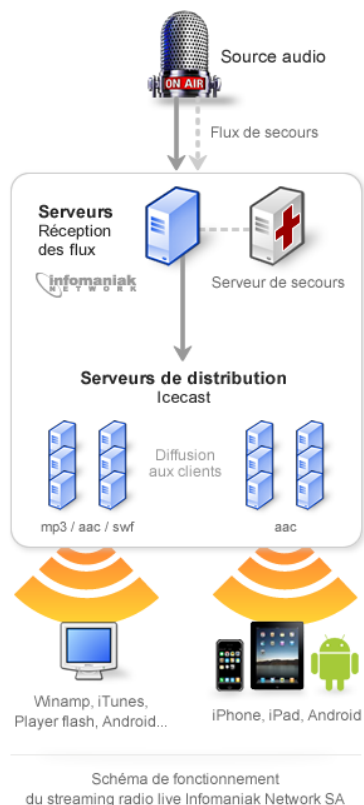


Figure 24 - Schéma de diffusion

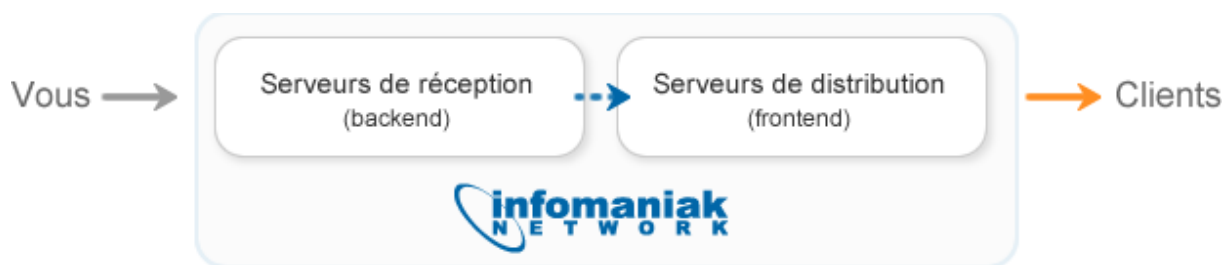


Figure 25 - Schéma de diffusion 2

Ces 2 schémas montrent le fonctionnement de la diffusion des webradios via infomaniak. Le logiciel avec les transcodeurs est la « source audio ». Infomaniak demande seulement un flux audio, peut importer le logiciel utilisé pour le diffuser. C'est ensuite leurs serveurs qui s'occuperont de diffuser le flux aux différents auditeurs.

## 6.5 SHOUTCAST



Figure 26 - Logo Shoutcast

### 6.5.1 PRÉSENTATION

SHOUTcast est le nom d'un protocole et d'un serveur de diffusion pour webradio ou pour webtv. Il a été créé par la société Nullsoft en même temps que le logiciel client Winamp pour l'écoute. Le protocole s'appuie sur deux protocoles, HTTP et ICY pour supporter les ID tag (« title streaming »).

Source : Wikipédia (<http://fr.wikipedia.org/wiki/SHOUTcast>)

SHOUTcast a été racheté récemment par la société Radionomy (<http://www.pcworld.fr/business/actualites/societe-belge-radionomy-confirme-rachat-winamp-shoutcast,545553,1.htm>). Cela a pour répercussion que les fichiers ne sont plus disponibles sur le site web de Shoutcast. Mais malgré cela, les fichiers de la version 2 sont toujours disponibles sur le forum de Winamp<sup>9</sup>. Ce sont donc ces derniers qui seront utilisés pour le projet.

### 6.5.2 POURQUOI CET OUTIL ?

<sup>9</sup> Lecteur multimédia développé par Nullsoft

J'ai choisi Shoutcast, car il propose des outils en ligne de commande ainsi qu'une gestion facilitée via des fichiers XML<sup>10</sup> ou texte basiques. Cela permet d'interagir plus facilement avec des applications externes telles que la mienne.

---

#### 6.5.3 SERVEUR

Shoutcast, parmi ses outils, propose un serveur en ligne de commande qui sera utilisée dans ce projet. Il permet de diffuser un flux qu'il reçoit et qui est envoyé par, par exemple, un [transcodeur](#). Ce flux est distribué aux clients (auditeurs dans ce cas) qui désirent écouter la webradio. Ce serveur propose aussi une interface web pour visualiser le statut (données sur le flux actuel, musique en cours, etc.) et d'administrer (il faut être authentifié en tant qu'administrateur avec le mot de passe défini dans la configuration du serveur) le serveur. Ces détails sont expliqués dans [l'analyse fonctionnelle \(5\)](#).

Plus de détails dans la partie consacrée au [serveur interne de diffusion](#). (6.13)

---

#### 6.5.4 TRANSCODER

Tout comme le serveur, le transcoder est un outil fourni par Shoutcast en ligne de commande. Il permet de diffuser un flux sur un serveur de diffusion. Ce flux peut-être en MP3 ou AAC+. Il donne aussi la possibilité de créer des playlists et de les agencer dans un calendrier XML. Les détails concernant ce système sont expliqués dans la suite de cette analyse organique.

Il gère de façon indépendante les horaires, les priorités entre les playlists et la lecture des fichiers musicaux. Le programme de ce projet va s'occuper de générer les fichiers nécessaires au transcoder en fonction des paramètres définis par l'utilisateur ainsi que d'afficher ces informations de façon visuelle (exemple : calendrier) afin de faciliter la manipulation et la configuration.

---

<sup>10</sup> eXtensible Markup Language : Langage de balisage extensible



## 6.5.5 SCHÉMA DE FONCTIONNEMENT RÉSUMÉ

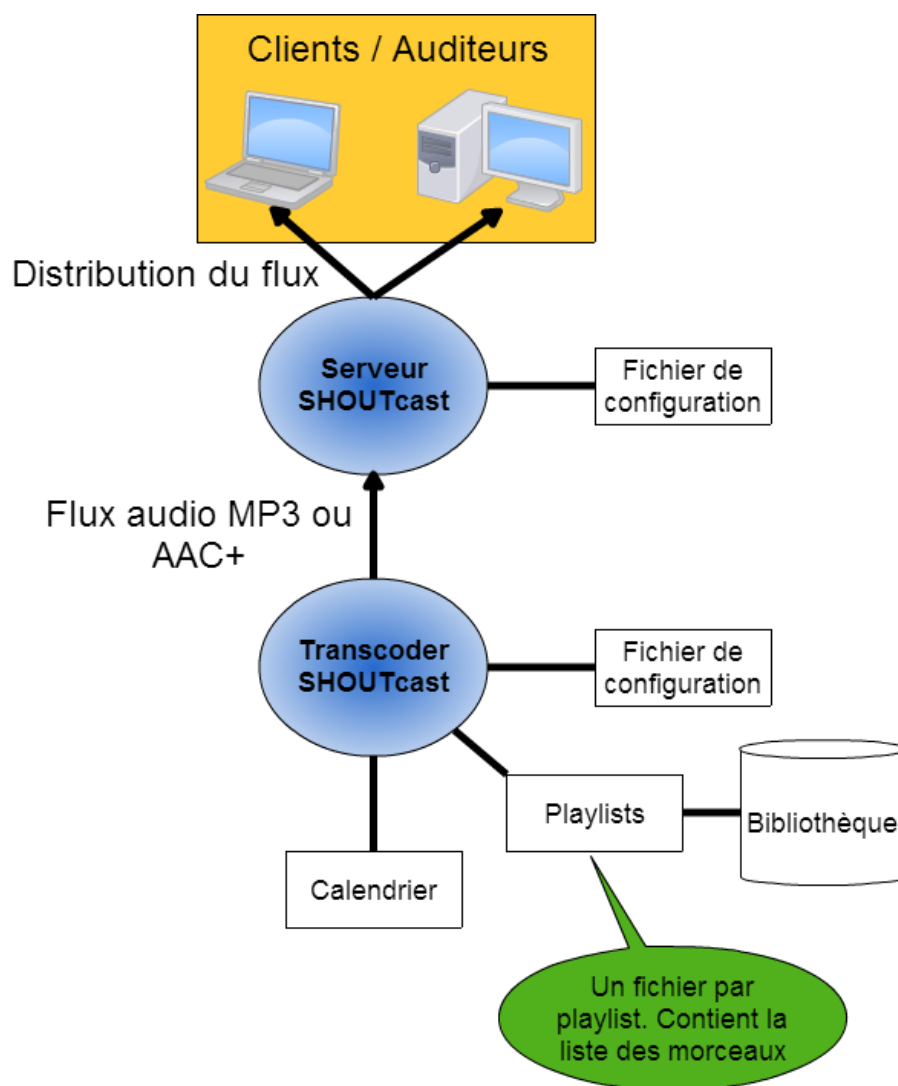


Figure 27 - Schéma shoutcast

## 6.6 STRUCTURES DES DOSSIERS/FICHIERS

### 6.6.1 SCHÉMA

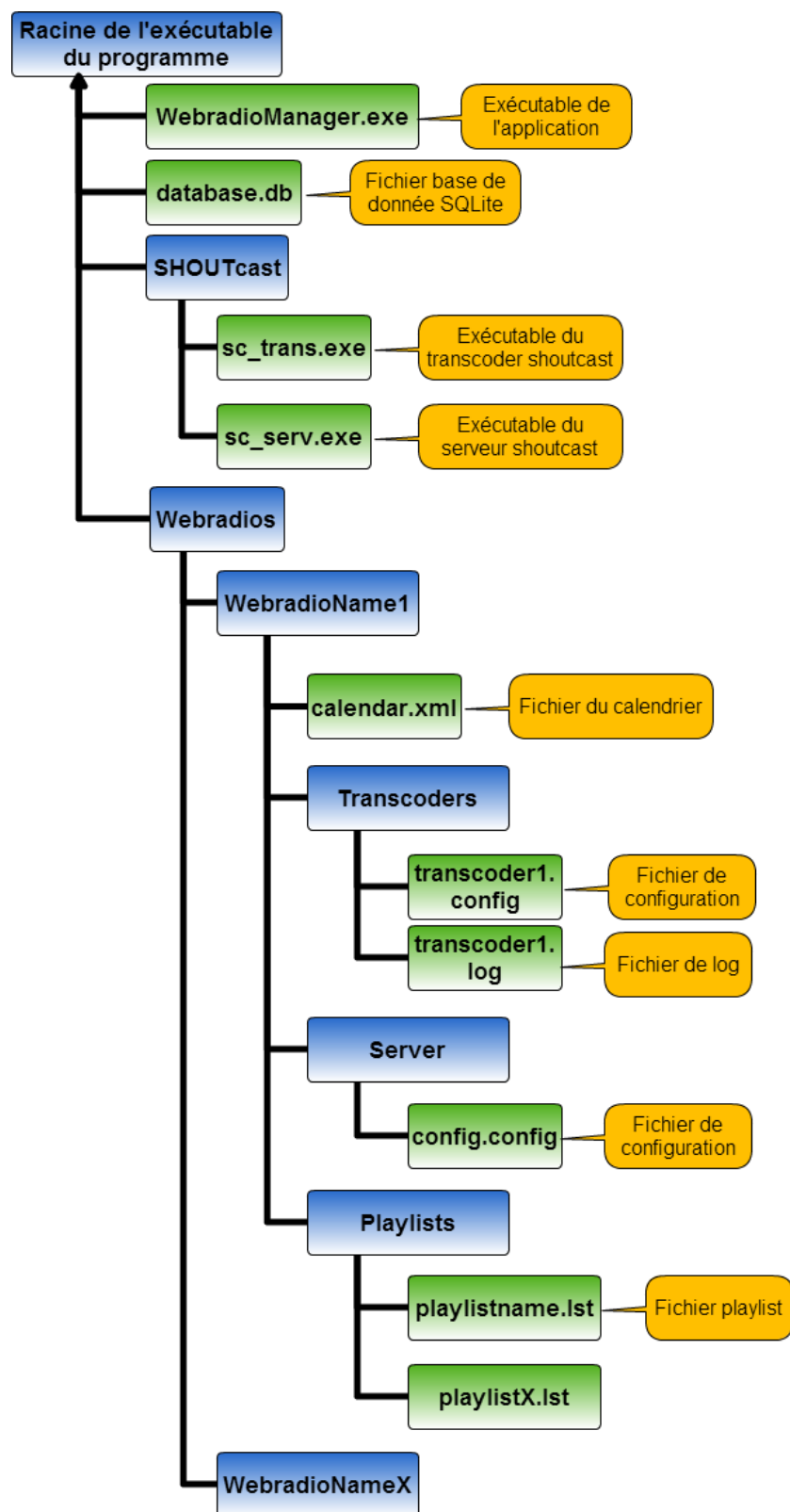


Figure 28 - Schéma structure des fichiers/dossiers

Ce schéma décrit l'organisation des fichiers dans le dossier de l'application. La base de données contient les informations pour les transcodeurs et les serveurs ainsi que le chemin vers les différents fichiers de ces dernières (configuration, calendrier, etc.). Cela permet à l'application de savoir où trouver les fichiers lors des traitements.

### 6.6.2 EXÉCUTABLES SHOUTCAST

Attention : Il n'y a pas un exécutable de transcodeur par transcodeur de webradio ni un exécutable de serveur par webradio. Il existe seulement un seul et unique exécutable transcodeur et serveur dans le dossier « shoutcast ». Ces exécutables peuvent être lancés avec le chemin vers un fichier de configuration en paramètre. Par exemple : À chaque fois qu'un transcodeur devra se lancer, une nouvelle instance de l'exécutable `sc_trans.exe` présent dans le dossier « shoutcast » sera lancée dans un nouveau processus et utilisera le fichier de configuration du transcodeur en question. Cela a été décidé, car si une mise à jour des exécutables doit être faite, seuls les 2 exécutables du dossier « shoutcast » seront mise à jour.

Voici ce principe illustré :

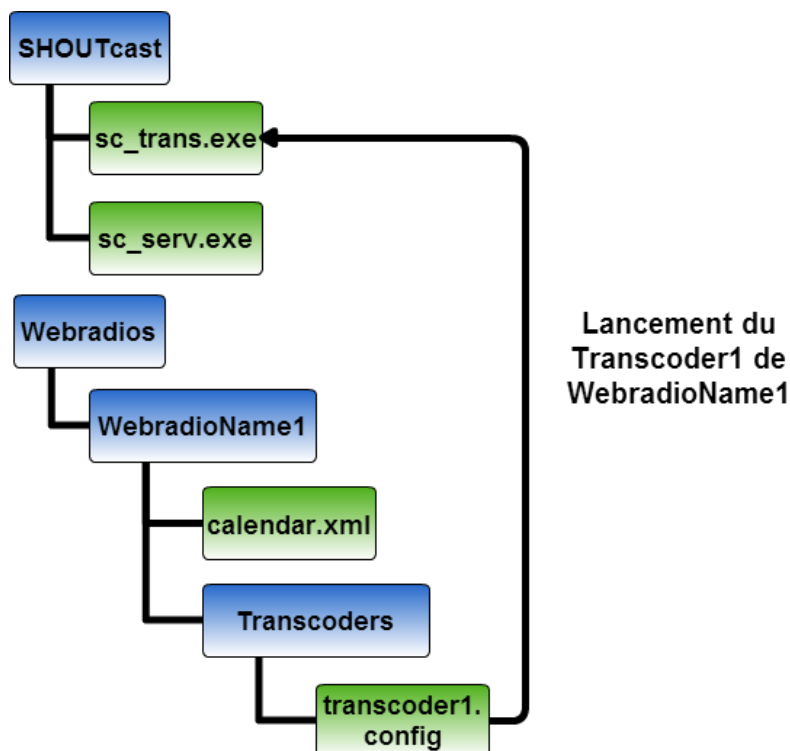


Figure 29 - Exemple lancement transcoder

## 6.7 INITIALISATION DE L'APPLICATION

La première fenêtre à se lancer est la *SelectionView*. C'est elle qui va appeler les différentes méthodes du *model* (via son *controller*) servant à l'initialisation.

C'est la classe *Bdd* qui s'occupe du traitement des données pour les passer ensuite au *model* pour que ce dernier remplisse ses champs « webradios » et « library ». Le diagramme de séquence suivant explique le déroulement de façon simplifiée.



Figure 30 - Diagramme de séquence initialisation application

À la fin de ce diagramme, SelectionView est affichée à l'utilisateur avec les informations recueillies avec UpdateView(). Cette méthode met à jour la *vue* avec les informations disponibles dans le *model* la concernant.

Le *model* vérifie avant tout que les dossiers de « base » (webradios, shoutcast) sont présents. Si ce n'est pas le cas, il les crée. Cette vérification est effectuée dans la méthode « LoadWebradios » du *model*.

Le *model* est donc rempli au démarrage de l'application. Toutes les informations contenues dans la base de données sont récupérées, traitées et ajoutées au *model*. Cela permet d'éviter un nombre de requêtes inutiles vers la base de données et de travailler avec les informations stockées en mémoire sous forme d'instances de classes dans le *model*.

Les 2 méthodes « LoadWebradios() » et « LoadLibrary() » de la classe Bdd s'occupent du traitement des informations. La première charge toutes les informations de façon hiérarchique (une webradio a un calendrier, qui lui dispose d'événement, etc.) pour chaque webradio de la base de données. La 2<sup>e</sup> charge les musiques présentes dans la bibliothèque ainsi que les playlists qui leur sont associées.

## 6.8 WEBRADIO

### 6.8.1 CLASSES ASSOCIÉES

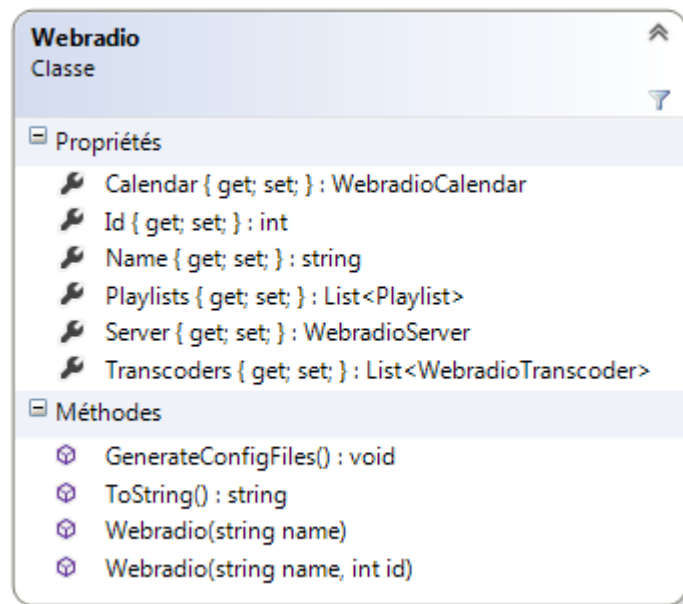


Figure 31 - Classe "Webradio"

La méthode « ToString » écrase (override) la méthode du même nom implémentée dans la classe parente de Webradio (Object). Elle retourne le nom de la webradio avec son identifiant. Cela permet lors de l'ajout d'objets de type Webradio à un ListBox, par exemple, d'afficher le nom de cette dernière, car les composants de type ListBox, ComboBox ou autres utilisent la méthode « ToString » des objets qu'ils contiennent pour afficher leur nom.

### 6.8.2 AFFICHAGE DES WEBRADIOS DISPONIBLES

Les webradios disponibles sont affichées dans la ListBox centrale de la fenêtre SelectionView. Pour la remplir, cela est effectué dans la méthode « UpdateView() » qui va récupérer les webradios du *model* et remplir la liste « d'items<sup>11</sup> » avec les objets de type Webradio obtenus. La ListBox appelle la méthode « ToString » des objets qu'elle contient pour afficher sa liste. Dans ce cas, la méthode « ToString » des objets Webradio sera appelée pour chacun d'entre eux.

### 6.8.3 CRÉATION

Une webradio ne peut pas avoir un nom qui dépasse 255 caractères. Pour se faire, la limitation est directement configurée dans la propriété « MaxLength » du TextBox permettant à l'utilisateur de nommer sa webradio.

<sup>11</sup> Voir [glossaire](#)

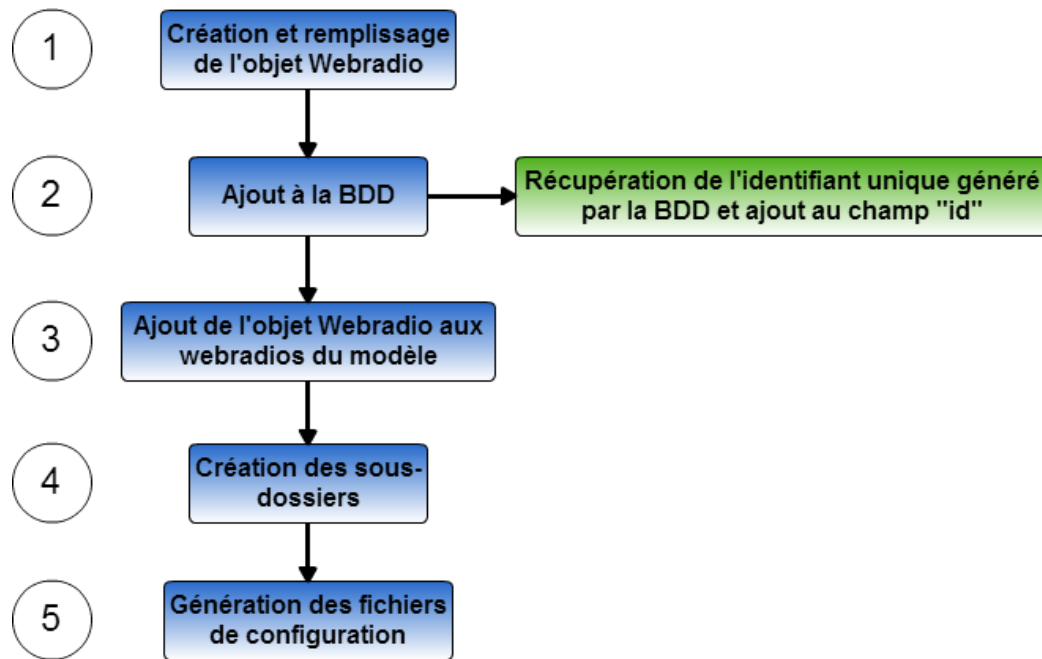


Figure 32 - Schéma création webradio

Comme présenté dans le schéma ci-dessus, la création se divise en 5 étapes, ces dernières s'effectuent dans le *model* et la méthode « CreateWebradio » :

- 1 : Instanciation<sup>12</sup> d'une classe Webradio avec le nom donné en paramètre à la méthode « CreateWebradio() ». Instanciation des classes nécessaires à la webradio (WebradioServer, WebradioCalendar et les différentes listes d'objet tel que la propriété Playlists). Seule la propriété « id » de l'objet webradio n'est pas remplie, car il s'agit de son identifiant dans la base de données, il sera donc rempli par la suite.
- 2 : L'objet créé est passé à la classe Bdd qui s'occupe de l'ajout de toutes ces informations dans la base de données via sa méthode « AddWebradio() » qui retourne l'identifiant qui a été attribué à cette nouvelle webradio par la base de données. Cet id est récupéré dans la méthode « CreateWebradio() » précédente et ajouté à l'objet webradio.
- 3 : L'objet webradio final est ajouté à la liste d'objets de type « Webradio » du *model*.
- 4 : Création des différents sous-dossiers pour stocker les fichiers de la nouvelle webradio.
- 5 : La méthode « GenerateConfigFiles() » est appelée afin de créer les fichiers de configuration nécessaires à la webradio. Plus d'informations sur cette méthode dans [ce chapitre](#). (6.8.8)

<sup>12</sup> Voir [glossaire](#)

#### 6.8.4 CHARGEMENT

Une webradio est chargée lorsqu'elle est sélectionnée via *SelectionView*. L'identifiant de la webradio à ouvrir est passé au *controller* de *SelectionView*. Ce dernier instancie un nouvel *AdminController* avec l'identifiant. Une fenêtre de type *AdminView* est ensuite ouverte avec les informations de la webradio sélectionnée. Ces informations viennent du *model*. Voici le diagramme de séquence pour la création d'une nouvelle instance d'une *AdminView* :

☞

Figure 33 - Diagramme de séquence instanciation *AdminView*

Après cette initialisation, le nouvel *AdminController* est ajouté à la liste d'observateurs du *model*.

À la création de la fenêtre, cette dernière appelle la méthode « *CheckFolders* » de son *controller*. Cette méthode va faire de même sur le *model*. En fin de compte, cette méthode vérifie la présence de tous les dossiers nécessaires à la webradio lancée avec *AdminView* (serveur, playlists et transcodeurs). Si un d'eux n'est pas présent, il est créé.

Comme montrée dans le diagramme, la méthode « *UpdateView* » de *AdminView* est appelée afin de charger les informations dans ses différents composants. La *vue* contient l'id de la webradio qui lui est attribuée. C'est avec cet id qu'elle va pouvoir effectuer des actions sur le *model* via le *controller*.

En ce qui concerne le remplissage des différentes *ComboBox* et *ListBox*, leurs *Items* ne seront pas de simples chaînes de caractères, mais des objets complets. Par exemple, les *ComboBox* affichant les



playlists disponibles comme dans l'onglet « Library », sont remplis avec des objets de type Playlist. Il est important que les classes ajoutées à des composants de ces types implémentent une méthode « ToString », car c'est celle qui est appelée par le composant lorsqu'il affiche, sous forme de chaîne de caractères, les éléments de sa liste. Cette méthode « override<sup>13</sup> » celle héritée par la classe parente « Object<sup>14</sup> ». De cette façon, il est facile de personnaliser les informations retournées par classe quand un composant utilise sa méthode ToString.

Le fait d'utiliser directement des objets dans les ListBox ou ComboBox permet de garder un pointeur sur les objets présents dans le *model*. Cela permet de manipuler un objet et ses modifications seront répercutées partout où il est utilisé.

Le calendrier est un composant spécial et il est aussi mis à jour lors de l'UpdateView. Pour plus d'information sur son fonctionnement, rendez-vous [au chapitre le concernant](#).

---

### 6.8.5 DUPLICATION

La duplication s'effectue en plusieurs étapes :

- Création du nom de la webradio « clone » : « Copy of » + nom de la webradio qui est clonée.
- Utilisation de la méthode du *model* nommée « [CreateWebradio](#) » puis récupération de la nouvelle webradio fraîchement ajoutée au *model* avec la méthode « GetWebradioByName ».
- Copie des informations (configuration) des différents éléments (serveur, playlists, calendrier et transcoder) de la webradio clonée, vers la webradio « clone ». Cette copie s'effectue en fait en utilisant les différentes méthodes de création d'éléments déjà présentes dans le *model*.
- UpdateObservers quand l'objet « clone » est à jour.

Plus de précision concernant la copie des informations. Exemple avec les transcoders :

```
foreach(WebradioTranscoder transcoder in webradio.Transcoders)
{
    this.CreateTranscoder(transcoder.Name, transcoder.StreamType, transcoder.SampleRate,
        transcoder.Birate, transcoder.Url, transcoder.Ip, transcoder.Port,
        transcoder.AdminPort, transcoder.Password, newWebradio.Id);
}
```

WMMModel.cs

Le foreach parcourt les transcodeurs de la webradio clonée et appelle la méthode création de transcoder avec les paramètres du transcoder courant, mais utilise l'identifiant de la nouvelle

---

<sup>13</sup> Le modificateur override est nécessaire pour étendre ou modifier l'implémentation abstraite ou virtuelle d'une méthode, d'une propriété, d'un indexeur ou d'un événement hérité(e).

<sup>14</sup> Voir [glossaire](#)

webradio. De ce fait, le nouveau transcoder sera créé pour la nouvelle webradio, mais avec les paramètres de la webradio clonée.

Concernant les playlists, par exemple, il y a une instance d'objet qui s'ajoute. La méthode « CreatePlaylist » utilise un paramètre avec le mot clé « out » (<http://msdn.microsoft.com/fr-fr/library/ee332485.aspx>). Cela permet, comme un pointeur, de modifier la valeur de la variable qui a été donnée en paramètre. Ce paramètre est de type Playlist. C'est-à-dire que l'objet de type Playlist donné en paramètre à la fonction sera rempli par la Playlist créée dans « CreatePlaylist » :

```
Playlist newPlaylist;  
  
if(this.CreatePlaylist(playlist.Name,newWebradio.Name,newWebradio.Id,playlist.Type,  
out newPlaylist))  
{  
    newPlaylist.AudioFileList = new List<string>(playlist.AudioFileList);  
}
```

WMMModel.cs

---

#### 6.8.6 SUPPRESSION

La suppression d'une webradio s'effectue via son identifiant. La méthode « DeleteWebradio() » du *model* va en premier temps supprimer la webradio de la base de données, puis de sa liste (dictionnaire) de webradios. La suppression dans la liste se fait via la méthode « Remove » proposée par les listes de type Dictionary qui prend l'identifiant de la webradio à supprimer.

### 6.8.7 CHANGEMENT DE NOM

Le schéma de traitement principal est le suivant :

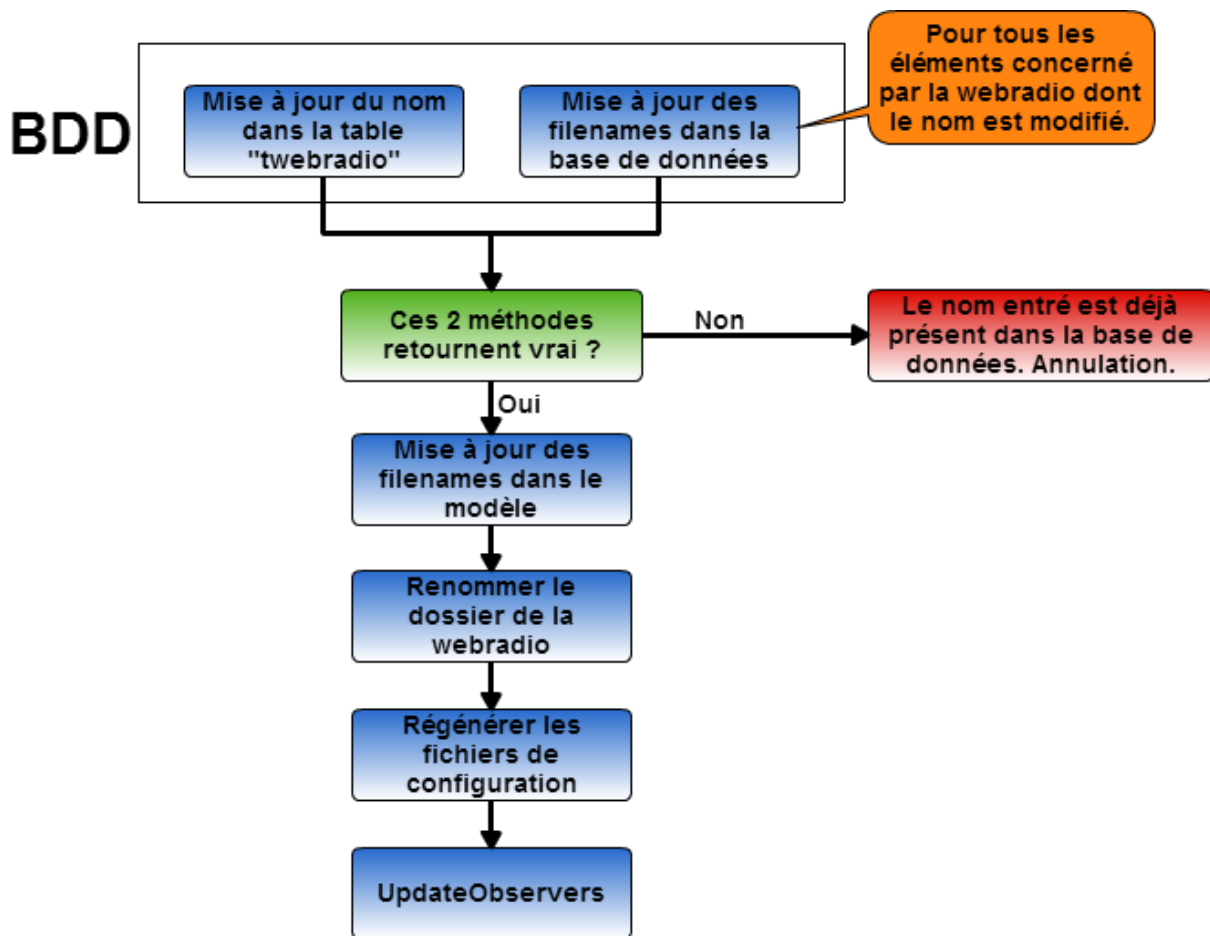


Figure 34 - Schema changement nom webradio

Les filenames (chemin vers les différents fichiers de configuration et de log des éléments d'une webradio) sont mis à jour, car ils référencent des fichiers qui se situent dans un dossier qui porte le nom de la webradio. Ce dossier est renommé, les anciens filenames seront donc corrompus. La mise à jour se fait simplement avec un replace :

```
transcoder.ConfigFilename = transcoder.ConfigFilename.Replace(oldName, newName);
```

WMMModel.cs

Lors d'une modification de nom, tous les processus de la webradio sont arrêtés, car renommer un dossier est impossible si les fichiers à l'intérieur sont utilisés.

### 6.8.8 GÉNÉRATION DES CONFIGURATIONS

La classe Webradio dispose d'une méthode « GenerateConfigFiles() » qui appelle la méthode « GenerateConfigFile() » de chacun de ses membres (Server, Playlists, etc.) ayant besoin d'un fichier de configuration.

Ces méthodes suppriment le fichier de configuration existant (s'il y en a un) et en créent un nouveau avec les informations contenues dans les champs de la classe en question. En fonction de la classe, le type de fichier de configuration sera différent (fichier texte ou fichier XML).

La génération de configuration est appelée lors de la sauvegarde ou le changement d'informations depuis l'AdminView, mais encore lors de la création d'une nouvelle webradio. Plus d'informations pour la génération de configuration de chaque composant du programme dans la suite de la documentation.

## 6.9 BIBLIOTHÈQUE

### 6.9.1 CLASSES UTILISÉES

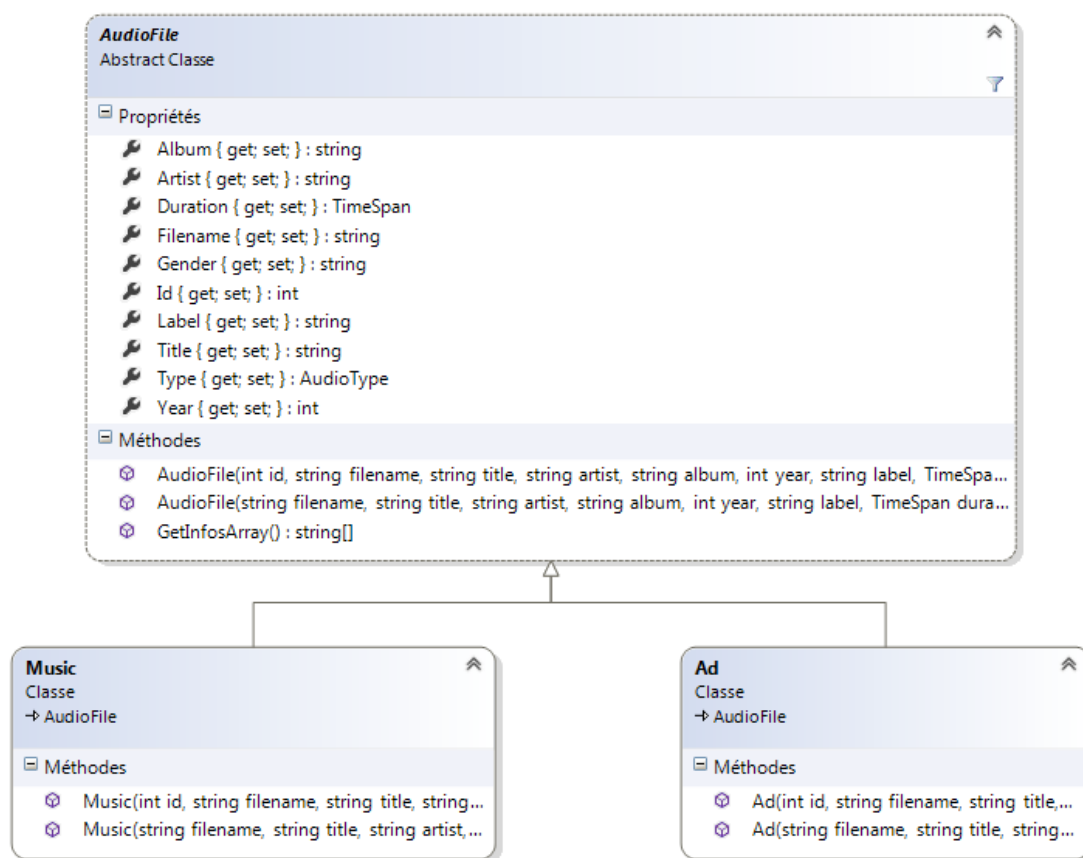


Figure 35 - Classes bibliothèque

### 6.9.2 MP3

À l'heure actuelle, seuls les fichiers MP3 peuvent être importés. La raison principale est que le flux de sortie de la webradio sera de toute façon compressé donc il n'y a pas d'intérêt à importer des fichiers de meilleure qualité (WAV par exemple). Des fichiers MP3 encodés en 320 kbits/s conviennent très bien et offrent une très bonne qualité pour une taille de fichier restreinte. De plus, le mp3 est une norme courante et utilisée par tout le monde.

Plus d'informations sur la technique de codage des fichiers MP3 : [http://fr.wikipedia.org/wiki/MPEG-1/2\\_Audio\\_Layer\\_3#Technique\\_de\\_codage](http://fr.wikipedia.org/wiki/MPEG-1/2_Audio_Layer_3#Technique_de_codage)

### 6.9.3 IMPORTATION

Depuis l'onglet « library », l'utilisateur choisit s'il veut importer un dossier ou des fichiers à la section « music » ou « ad » (publicité, top-horaire, etc.). Chaque bouton contient une valeur dans sa propriété « tag » qui est soit « Music » ou soit « Ad », car les événements « OnClick » des boutons (regroupés par type d'importation, c'est-à-dire, par dossier ou fichiers) pointent sur une même

méthode et cette propriété « tag » permet de différencier si le bouton cliqué est dans la section « music » ou « ad ».

Dans le cas d'un dossier, le traitement va rechercher tous les fichiers MP3 contenus dans le dossier sélectionné. Si l'utilisateur a désiré d'importer de façon récursive, les sous-dossiers du dossier sélectionné seront aussi analysés.

La méthode statique « GetFiles » de la classe .NET « Directory » permet de récupérer un tableau de string contenant les « filename » (chemin absolu vers les fichiers) des fichiers correspondants au pattern donné en paramètre dans le dossier spécifié. Ce pattern se présente sous la forme : « \*.mp3 » pour récupérer seulement les fichiers dont l'extension est « mp3 ». Une option permet de faire cette recherche de façon récursive. Voici un exemple de code pour des fichiers mp3 et récursivement :

```
Directory.GetFiles(FBD.SelectedPath, "*.mp3", SearchOption.AllDirectories);
```

Pour une recherche non récursive, l'option « SearchOption » doit être changée en : `SearchOption.TopDirectoryOnly`.

La *vue* va ensuite passer le tableau de string au *model* via son *controller*. Le *model* va effectuer le traitement (analyse de tags ID3<sup>15</sup> (version 1 et 2) et ajout à la base de donnée/*model*) avec sa méthode « ImportFilesToLibrary ». Cette méthode est générique, il suffit de lui envoyer un tableau de filenames pour fonctionner. Cela permet qu'une importation par fichiers ou dossier puisse utiliser la même méthode. Voici le schéma récapitulatif :

---

<sup>15</sup> Voir chapitre concernant [les tags ID3](#)

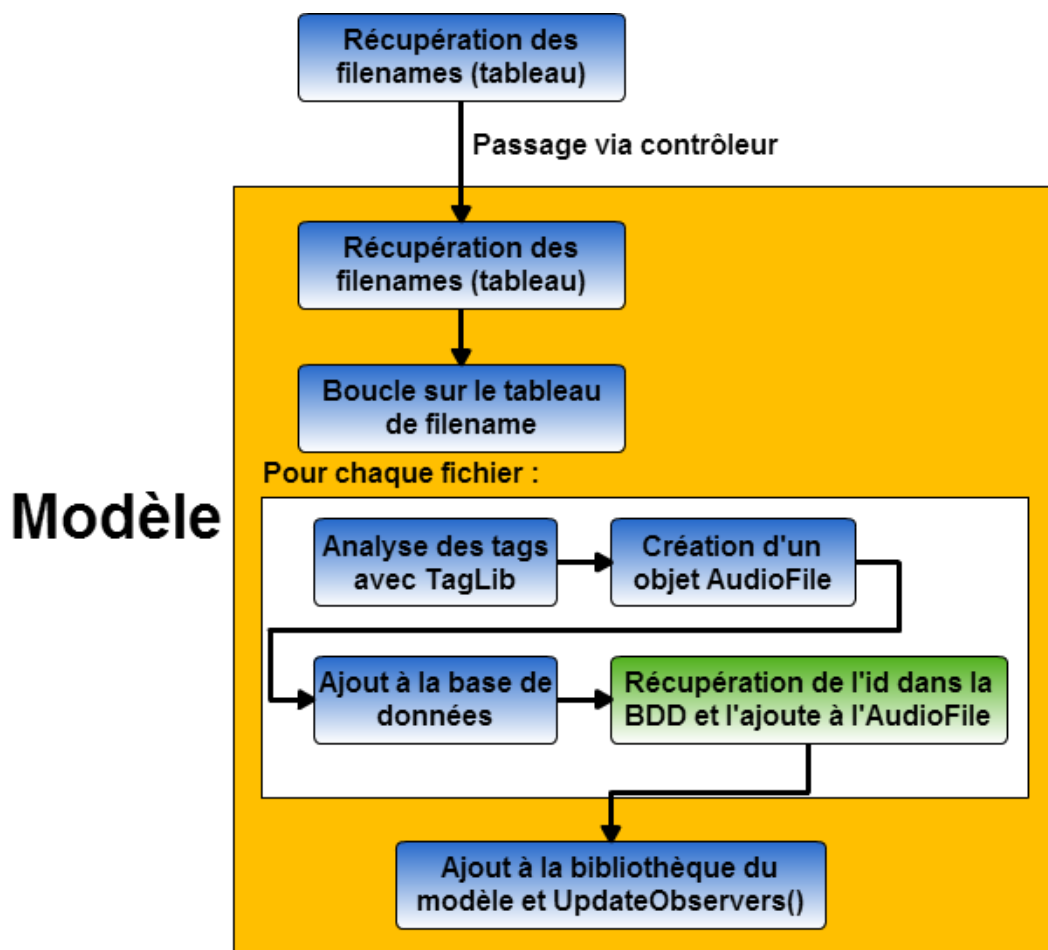


Figure 36 - Schéma importation fichier

La partie « analyse de tag » est expliquée plus précisément [ici](#). Concernant l'ajout à la base de données, [ce chapitre](#) décrit le procédé.

#### 6.9.4 TAGS ID3

ID3 est le nom des métadonnées pouvant être insérées dans un fichier audio comme MP3. Ces métadonnées<sup>16</sup> permettent d'avoir des informations sur le contenu du fichier comme le titre, le nom de l'interprète, les commentaires, ou encore la date de sortie.

Source : Wikipédia

#### 6.9.5 ANALYSE DES TAGS

La récupération des tags ID3 est effectuée à l'aide de la bibliothèque « TagLib-Sharp » :

<https://github.com/mono/taglib-sharp>

<sup>16</sup> Une métadonnée est une donnée servant à définir ou décrire une autre donnée quel que soit son support (papier ou électronique).

Le fonctionnement de cette bibliothèque est simple. Pour analyser et récupérer les tags ID3 d'un fichier, elle fournit une classe File (à ne pas confondre avec la classe File fournie par .NET) qui propose une méthode statique<sup>17</sup> « Create » qui prend un filename en paramètre. Cette dernière retourne donc un objet instancié de type « TagLib.File » qui contient toutes les informations sur le fichier donné.

2 types d'informations importantes se distinguent et qui seront utilisées :

- La propriété « Tag » qui contient des sous-propriétés comme « Title », « Year », etc. Elles sont les tags à proprement parler et donnent des informations au sujet du contenu musical.
- La propriété « Properties » qui contient des sous-propriétés comme « Duration », « BitRate », etc. Ce sont les informations concernant le fichier en lui-même.

---

#### 6.9.6 INDEXATION

Indexation est la partie qui consiste à enregistrer les informations des musiques de la bibliothèque dans la base de données. Pour se faire, quelques règles sont établies :

- Il ne peut y avoir qu'une occurrence par fichier. C'est-à-dire, pas de doublon. Plusieurs musiques peuvent avoir les mêmes informations dans les tags, mais c'est le nom de fichier qui fait foi.
- Chaque genre musical à son enregistrement dans la table « tgender ». Si lors de l'ajout d'une musique/pub à la base de données, son genre n'est pas encore dans cette table, il y est ajouté et son identifiant est récupéré. Dans l'autre cas, si le genre est déjà présent dans la table, son identifiant est juste trouvé dans la table. Si aucun genre n'est trouvé dans le tag, un genre vide lui est attribué.

Voici ces 2 règles représentées sous forme de code :

```
if(this.AudioFileExist(file.Filename))  
  
    return ERROR;  
  
int genderId = this.GetGenderId(file.Gender);  
//If return error, gender doesn't exist in DB, so add it  
if (genderId == ERROR)  
  
    //Get the new id  
    genderId = AddGender(file.Gender);
```

Bdd.cs

---

<sup>17</sup> Méthode qui ne nécessite pas que sa classe soit instanciée pour être utilisée.



Ensuite, l'ajout s'effectue avec un nouvel enregistrement dans la table « tmusic » puis l'identifiant de ce nouvel enregistrement est retourné à la fin de la méthode.

---

#### 6.9.7 MODIFICATION

Les colonnes « id », « duration » et « path » sont en lecture seule (propriété ReadOnly).

Lorsque l'utilisateur valide ses modifications, l'événement « CellEndEdit » du DataGridView en question est appelé. Un objet de type Music ou Ad (en fonction du DataGridView qui appelle l'événement) est créé avec les informations de la ligne modifiée (modifications comprises), puis la méthode « UpdateAudioFile » du *controller* est appelée. C'est ensuite dans le *model* que les modifications sont appliquées à la base de données, au fichier MP3 (tags) et à la bibliothèque contenus dans le *model* lui-même. Les étapes de modification s'effectuent comme ceci :

1. Mise à jour dans la base de données. Si la méthode « UpdateAudioFile » de la classe Bdd retourne vrai, passage à l'étape suivante.
  - a. La méthode située dans la classe Bdd effectue la même vérification que pour l'ajout de fichier audio concernant le genre. C'est-à-dire que si le genre inscrit n'existe pas, il est ajouté à la table tgender.
2. Modification des tags du fichier MP3 à l'aide de la bibliothèque C# TagLib.
3. Modification des informations dans le *model*
4. Mise à jour des observateurs

La mise à jour via TagLib s'effectue comme ceci :

```
TagLib.File tagFile = TagLib.File.Create(file.Filename);
tagFile.Tag.Title = file.Title;
tagFile.Tag.Performers = new string[] { file.Artist };
tagFile.Tag.Album = file.Album;
tagFile.Tag.Year = (uint)file.Year;
tagFile.Tag.Copyright = file.Label;
tagFile.Tag.Genres = new string[] { file.Gender };
tagFile.Save();
```

WMMModel.cs

---

#### 6.9.8 SUPPRESSION

Pour la suppression, le même principe que l'importation par rapport à la différenciation entre le bouton « Delete selected » de la section « Music » et « Ad » avec la propriété « Tag ».

L'utilisateur peut supprimer plusieurs occurrences d'un seul coup, pour cela il faut parcourir la propriété « SelectedRows » du composant « DataGridView » en question. Voici la boucle :

```
foreach(DataGridViewRow row in ((type ==  
AudioType.Music)?dgvMusics.SelectedRows:dgvAds.SelectedRows))  
{  
    if (!this.Controller.DeleteAudioFile(int.Parse(row.Cells[0].Value.ToString()),  
row.Cells[row.Cells.Count - 1].Value.ToString()))  
        state = false;  
}
```

AdminView.cs

La présence d'un test ternaire<sup>18</sup> dans la partie « in » du foreach permet de sélectionner le bon composant DataGridView en fonction de la section (Music ou Ad) dans laquelle le bouton de suppression a été pressé.

Pour chaque occurrence, la musique/pub va être supprimée de la base de données ainsi que du *model* et des playlists la concernant. Pour se faire, toutes les playlists de toutes les webradios du programme vont être bouclées et dans la liste de filename de chacune, les occurrences du filename de la musique/pub supprimée seront enlevées.

Ensuite, la suppression s'effectue dans la bibliothèque du *model* puis dans la base de données.

---

#### 6.9.9 VÉRIFICATION DES DONNÉES

Dans le menu « File », le bouton « Check library » permet de vérifier l'intégrité de la bibliothèque. C'est-à-dire, supprimer les références vers des fichiers musicaux qui n'existeraient plus à l'emplacement auquel ils sont référencés. La méthode du *model* « CheckLibrary » parcourt toute la bibliothèque et vérifie que le filename stocké dans l'objet AudioFile référence bien un fichier existant. Si ce n'est pas le cas, l'enregistrement est supprimé de la base de données et du *model*.

---

#### 6.9.10 RECHERCHE

La recherche consiste à afficher seulement les lignes ayant au moins une correspondance (n'importe quel champ de la ligne) avec la chaîne de recherche entrée par l'utilisateur. Cette chaîne est en premier temps mise en minuscule afin de ne pas prendre la casse en compte.

Une boucle parcourt les lignes du DataGridView et pour chacune, une autre boucle parcourt les cellules. Ensuite, pour chaque cellule, la valeur de cette dernière est prise, un « ToString » suit d'un « ToLower » y est appliqué (pas de casse<sup>19</sup>) et enfin la méthode « Contains » va retourner un booléen si elle trouve une occurrence dans la valeur de la cellule. Si c'est le cas, cette ligne est valide et pourra être affichée. La propriété « Visible » de la ligne est donc modifiée à *true* mais dans le cas contraire elle sera *false*. Et ainsi de suite pour chaque ligne.

---

<sup>18</sup> Un test ternaire utilise l'expression ternaire pour effectuer un test : test ? expression1 : expression2.

<sup>19</sup> Pas de différence entre lettres majuscules et minuscules.

```
searchString = (sender as TextBox).Text.ToLower();
foreach(DataGridViewRow row in ((type == AudioType.Music)?dgvMusics.Rows:dgvAds.Rows))
{
    foreach(DataGridViewCell cell in row.Cells)
    {
        if (cell.Value.ToString().ToLower().Contains(searchString))
        {
            valid = true;
            break;
        }
    }
    row.Visible = (valid) ? true : false;
    valid = false;
}
```

AdminView.cs

## 6.10 LISTES DE LECTURE

Les listes de lecture sont utilisées par le calendrier, qui lui, est utilisé par un transcoder.

### 6.10.1 CLASSES UTILISÉES

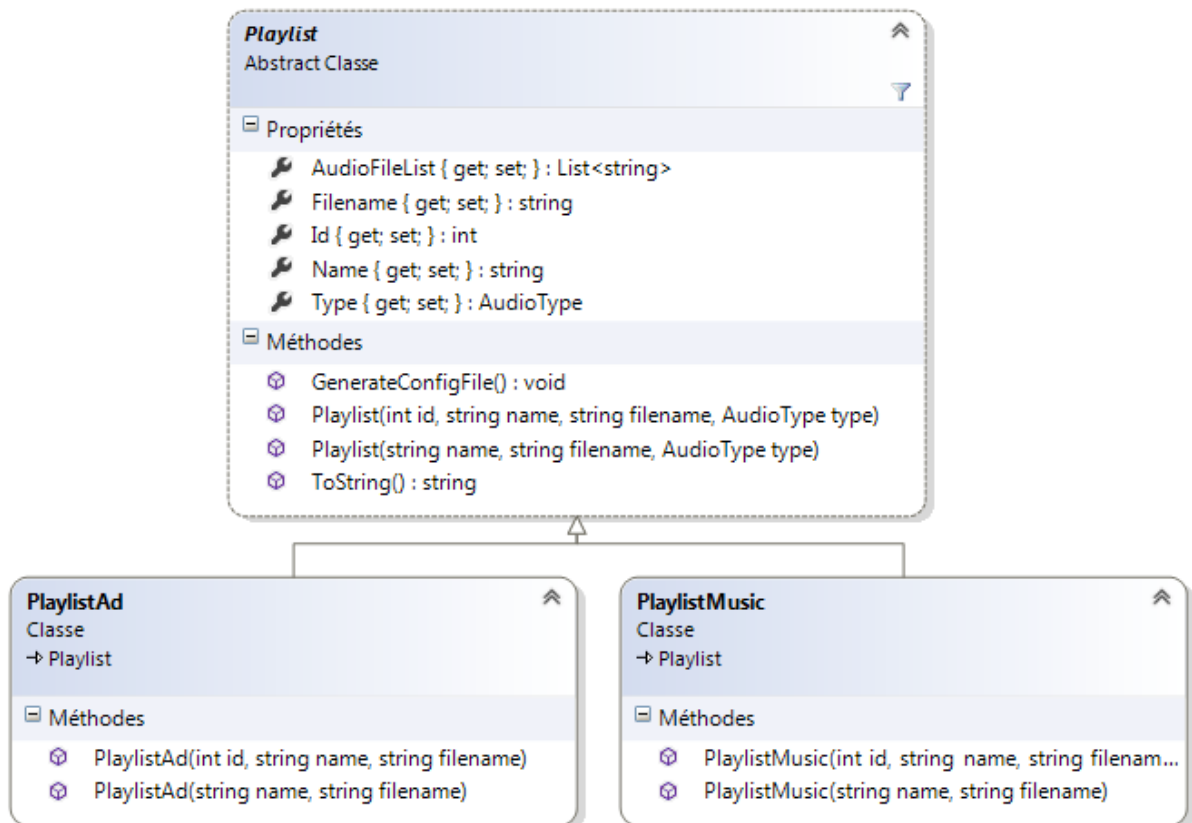


Figure 37 - Classes Playlist

### 6.10.2 GÉNÉRATION DE CONFIGURATION

Une liste de lecture comprend un fichier de configuration dont la forme est une simple liste des fichiers à utiliser. Elle est stockée dans un fichier texte avec l'extension « .lst ». Voici un exemple de fichier :

```

C:\Users\MENETREYS_INFO\Music\test\Wasted Penguinz - Wistfulness\Wasted_Penguinz-
Those_Were_The_Days_Original_Instrumental_Mix-ToffMusic.mp3

C:\Users\MENETREYS_INFO\Music\test\Wasted Penguinz - Wistfulness\Wasted_Penguinz-
Everlasting_Outro-ToffMusic.mp3

C:\Users\MENETREYS_INFO\Music\test\Wasted Penguinz -
Wistfulness\Wasted_Penguinz_and_Toneshifterz-Together_Extended_Version-ToffMusic.mp3

C:\Users\MENETREYS_INFO\Music\test\Wasted Penguinz - Wistfulness\Wasted_Penguinz-
Falling_Extended_Version-ToffMusic.mp3
  
```

C:\Users\MENETREYS\_INFO\Music\test\Wasted Penguinz - Wistfulness\Wasted\_Penguinz-Endless\_Extended\_Version-ToffMusic.mp3

Playlisttest.lst

Chaque ligne du fichier correspond au « filename » (chemin de fichier) vers la musique/pub de la playlist. C'est grâce à cela que le transcoder pourra aller chercher les musiques/pubs pour les jouer et les diffuser.

Concernant la génération, c'est le champ « AudioFileList » de la classe « Playlist » qui contient les filenames des musiques/pubs de la playlist, qui est parcouru afin de générer le fichier.

---

### 6.10.3 CRÉATION D'UNE PLAYLIST

Pour la création d'une nouvelle playlist, une règle importante a été établie :

- Le nom d'une playlist doit être unique (au sein de la même webradio et avec le même type (Music ou Ad)).
- Par exemple : Il peut y avoir 2 playlists qui se nomment « Test » au sein de la même webradio si chacune d'entre elles a un type différent.
- Le nom ne doit pas contenir de caractères Windows invalides. C'est-à-dire, des caractères interdits pour des noms de fichiers Windows.

Concernant la création à proprement parler, elle se décompose en plusieurs étapes qui se déroulent dans la méthode « CreatePlaylist » du *model* :

1. Création du futur filename du fichier de configuration de la playlist sous cette forme :  
DEFAULT\_WEBRADIOS\_FOLDER + webradioName + "/" + DEFAULT\_PLAYLISTS\_FOLDER + name + ".lst";
2. Instanciation d'une classes PlaylistMusic ou PlaylistAd en fonction du type de playlist créée.
3. Insertion dans la base de données. La méthode d'ajout à la BDD retourne l'identifiant dans la BDD de la nouvelle playlist. Si cet id correspond « ERROR » (constante définie), une erreur est survenue lors de l'ajout. Dans ce cas, la méthode de création retourne directement « false ». Si un identifiant valide a été retourné, il est configuré à l'objet de type playlist instancié précédemment.
4. La méthode « [GenerateConfigFile](#) » est appelée sur l'objet Playlist afin de créer son fichier de configuration (bien que vide pour le moment).
5. L'objet est ajouté au *model* (dans la propriété « Playlists » de la webradio à laquelle la playlist est ajoutée).
6. « UpdateObservers » est appelé afin de mettre à jour toutes les fenêtres concernées.

---

#### 6.10.4 GÉNÉRATION AUTOMATIQUE

Le principe de la génération automatique est de créer une playlist d'une durée donnée en la remplissant (aléatoirement) de musiques/pubs du genre donné. Dans le cas de pubs, le genre n'est pas pris en compte.

Il faut savoir que plus la durée demandée grande, plus il sera possible de s'en rapprocher le plus possible en utilisant les morceaux disponibles dans la bibliothèque.

Concernant l'algorithme, il consiste à parcourir la bibliothèque du *model* de façon aléatoire (avec l'objet de type Random) dans une boucle de type while. La condition de cette dernière est que la durée de la playlist doit rester plus petite que la durée demandée. Le but étant de se rapprocher le plus possible de la durée demandée sans la dépasser.

Il est déterminé que l'algorithme essaie un nombre de fois (consécutives), défini par la constante « MAX\_TRY\_GENERATE », de remplir la liste de lecture. Si ce nombre de fois est dépassé, il est considéré qu'il n'est plus possible de remplir sans dépasser la limite de durée et la boucle est donc arrêtée.

À chaque tour de boucle, une musique/pub est « piochée » et il est testé si le temps actuel de la playlist + le temps de la musique sélectionnée dépasse la durée demandée. Si c'est le cas, le nombre d'essais s'incrémente de 1 et la boucle refait un tour avec l'instruction « continue ». Si ce n'est pas le cas, le compteur d'essais est remis à zéro et la boucle continue son traitement sans interruption. Ensuite, un test vérifie si le type est Ad « ou » si le type est Music « et » du genre demandé, c'est en passant ce test que la musique/pub est ajoutée à la playlist ainsi que sa durée qui est additionnée à la durée actuelle de la playlist. Une liste d'entier est aussi utilisée pour stocker les identifiants des morceaux ajoutés à la playlist afin de pouvoir les utiliser lors de l'ajout à la base de données.

Voici un schéma récapitulatif de la méthode présente dans le *model* :

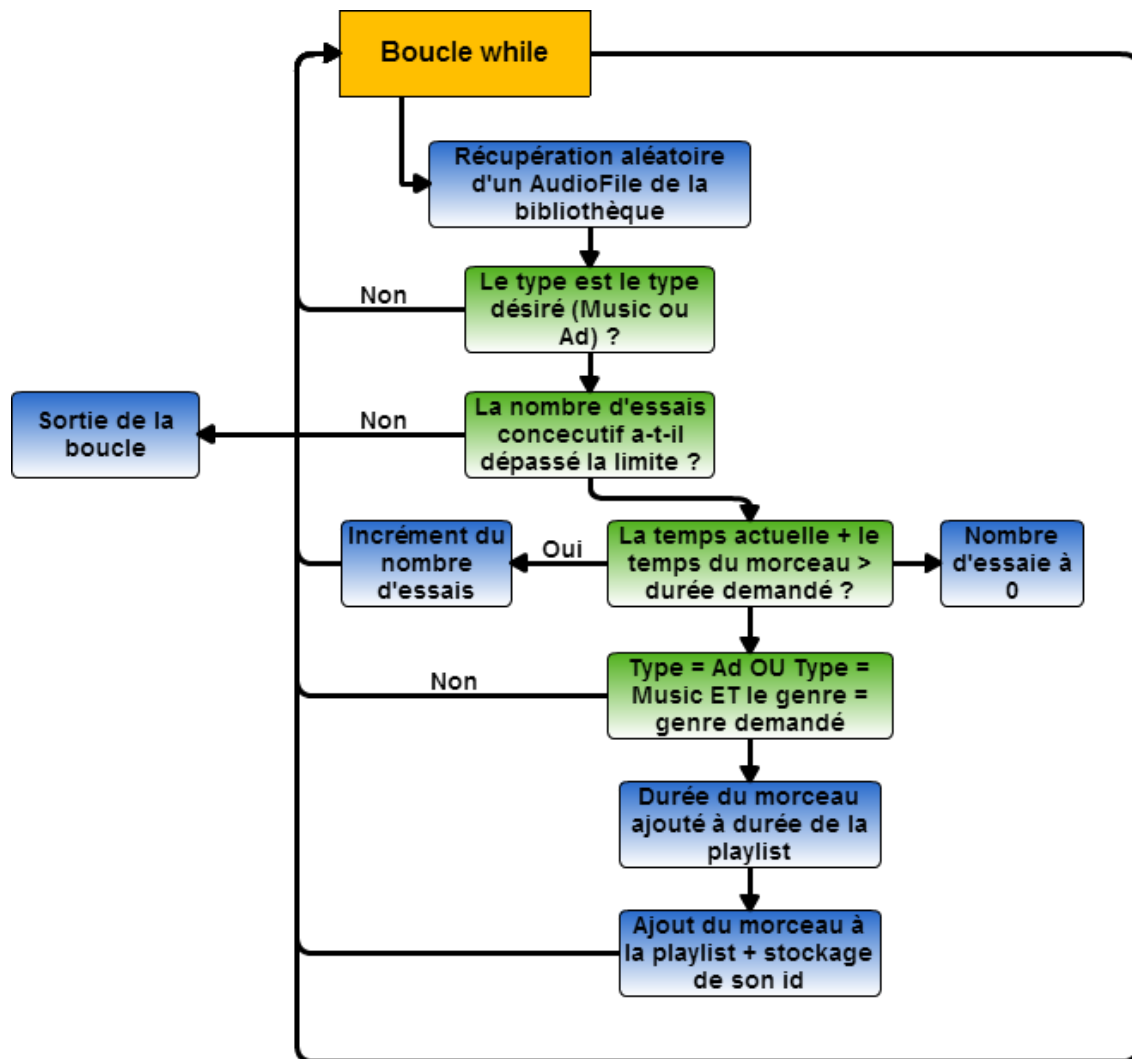


Figure 38 - Algorithme génération playlist

Au final, après la boucle, un test vérifie qu'il y a bien des musiques/pubs présentent dans la playlist. Si n'est pas le cas (l'algorithme n'a pas réussi à combler la durée malgré les essais), la méthode retourne *false*. Dans le cas contraire, la playlist est ajoutée à la base de données et au *model*. Pour finir, le fichier de configuration de la playlist est généré et la méthode « UpdateObservers » est appelée.

#### 6.10.5 AJOUT À UNE PLAYLIST

L'ajout de musique/pub s'effectue depuis l'onglet « Library ».

Les 2 boutons d'ajout (celui dans la section Music ou la section Ad) appellent la même méthode d'événement. Pour différencier quel bouton appelle la méthode, la valeur « Music » ou « Ad » est inscrite dans la propriété « Tag » des boutons. Un test est effectué pour détecter quel bouton a été appuyé.

En premier temps, la *vue* va générer une liste de type `Dictionnaire<int,string>`. La clé (int) correspond à l'identifiant de la musique sélectionnée et la valeur (string) correspond à son filename. Ce

dictionnaire va permettre au *model* d'ajouter les différents éléments dans le *model* et la base de données. Il est généré à l'aide de la propriété « SelectedRows » du composant *dataGridView*. Cette propriété donne la liste des lignes sélectionnées par l'utilisateur.

Par la suite, le dictionnaire est envoyé au *model* via le *controller*. Il est parcouru par la méthode « AddToPlaylist » qui prend en paramètre un objet Playlist (correspondant à la playlist sélectionnée) et le dictionnaire. Pour chacun des éléments, il est ajouté à la base de données via la méthode de la classe Bdd « AddToPlaylist » qui prend en paramètre : la clé de l'événement (la valeur int qui correspond à l'id du morceau) et l'id de l'objet Playlist. Si cette méthode retourne une erreur, la boucle est arrêtée et le *model* retourne *false*. Sinon, la boucle se finit et la playlist génère sa configuration (GenerateConfigFile sur l'objet playlist).

---

#### 6.10.6 RETIRER D'UNE PLAYLIST

La suppression d'éléments d'une playlist se déroule exactement comme l'ajout (dictionnaire avec les morceaux sélectionnés par l'utilisateur qui est parcouru par le *model*) à l'exception que les éléments seront supprimés de la base de données (suppression du lien dans la table « tplaylist\_has\_music ») et du *model*. À la fin, la nouvelle configuration est générée.

---

#### 6.10.7 AFFICHAGE DU CONTENU

Lorsque l'utilisateur choisit un élément dans un des ListBox (section Music ou Ad), l'événement « SelectedIndexChanged » est appelé. Comme pour les autres éléments, chacun des ListBox à son type dans sa propriété Tag afin de différencier dans la méthode de l'événement en question.

La méthode privée de la *vue* « GetPlaylistContent » qui prend un objet de type Playlist en paramètre permet de vider le DataGridView servant d'affichage au contenu des playlists, puis de le remplir avec le contenu de la playlist voulue. Pour se faire, elle récupère une liste d'objet de type AudioFile en provenance du *model* via son *controller*. Le *model* va simplement parcourir sa bibliothèque et prendre les AudioFile dont le nom de fichier (filename) correspond à un des noms de fichier présent dans la playlist :

```
List<AudioFile> audioFiles = new List<AudioFile>();
foreach(string filename in playlist.AudioFileList)
{
    foreach(AudioFile af in this.Library)
    {
        if (af.Filename == filename)
            audioFiles.Add(af);
    }
}
return audioFiles;
```



Ensuite, la *vue* va parcourir cette liste afin de remplir le DataGridView d'affichage.

Les classes Playlist disposent d'une méthode « GetInfosArray » qui retourne un tableau avec les informations de leurs champs. Cela permet de donner un tableau à la méthode d'ajout de ligne au dataGridView : `dgvPlaylistContent.Rows.Add(af.GetInfosArray());`

Par la même occasion, la durée de chaque morceau ajouté est additionnée à une variable de type TimeSpan afin de calculer la durée totale de la playlist et de l'afficher.

---

#### 6.10.8 SUPPRESSION D'UNE PLAYLIST

La suppression de playlist va supprimer l'enregistrement correspondant dans la base de données, la variable dans le *model* ainsi que le fichier de configuration enregistré sur le disque. Les *vues* sont ensuite mises à jour via UpdateObservers.

## 6.11 GRILLE HORAIRE

### 6.11.1 CLASSES UTILISÉES

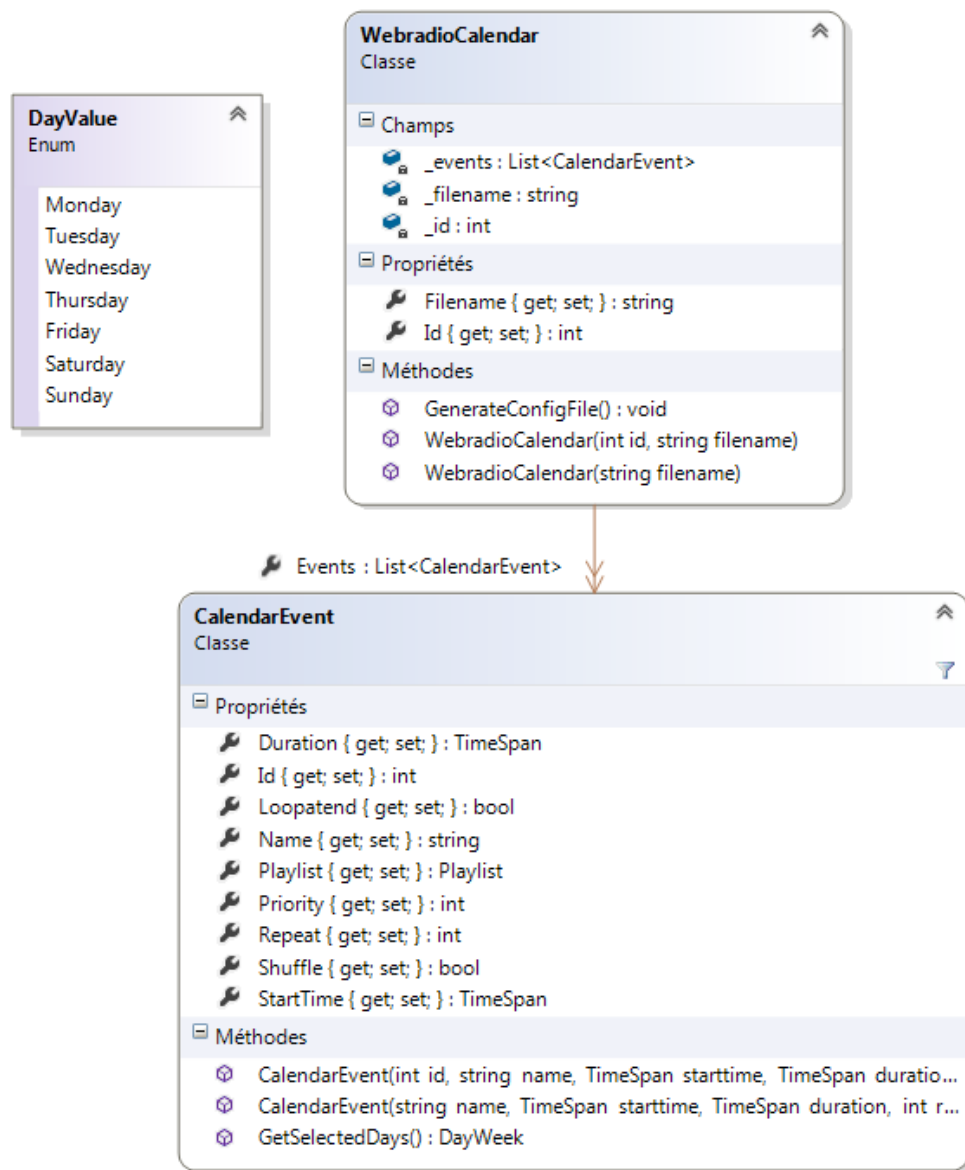


Figure 39 - Classes timetable

L'enum « DayValue » contient des constantes définies par ShoutCAST transcoder :

[http://wiki.winamp.com/wiki/SHOUTcast\\_Calendar\\_Event\\_XML\\_File\\_Specification#Calendar\\_Tag](http://wiki.winamp.com/wiki/SHOUTcast_Calendar_Event_XML_File_Specification#Calendar_Tag)

### 6.11.2 OUTIL UTILISÉ

Pour l’affichage et la gestion de la timetable<sup>20</sup>, un composant tiers est utilisé :

<http://calendar.codeplex.com/>

Il s’agit de « Day View Calendar ». Ce composant a été choisi, car il propose une *vue* sous forme de jour et il est entièrement personnalisable (nombre de jours affichés, découpage des heures, etc.).

Voici un exemple d’utilisation :

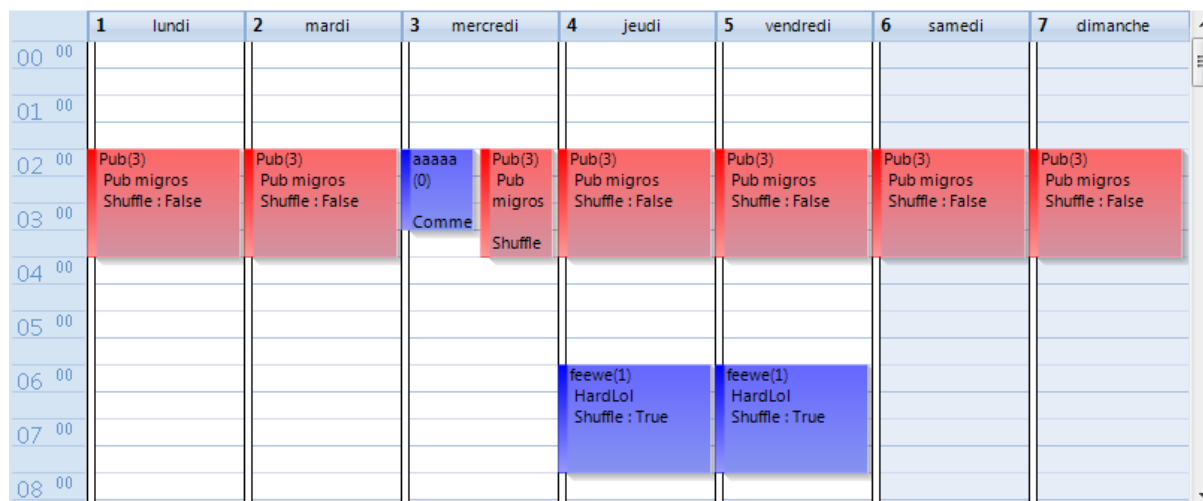


Figure 40 - Day View Calendar

### 6.11.3 GESTION DU CALENDRIER PAR SHOUTCAST

C’est le transcoder fournit par ShoutCAST qui se charge de prendre en compte le calendrier et jouer les playlists aux bons moments. Ce calendrier est sous forme XML et il est composé d’événements (event). Chaque événement est composé des propriétés suivantes (celles présentées sont celles utilisées dans l’application. Pour une description complète, rendez-vous si le site officiel :

[http://wiki.winamp.com/wiki/SHOUTcast\\_Calendar\\_Event\\_XML\\_File\\_Specification](http://wiki.winamp.com/wiki/SHOUTcast_Calendar_Event_XML_File_Specification)) :

- Un type (playlist ou DJ) : Dans le cas de l’application, le type playlist sera toujours utilisé
- Une playlist : La lecture de la playlist dispose de plusieurs paramètres :
  - « loopatend » est un booléen qui définit si la playlist doit être rejouée quand tous les morceaux qui s’y trouvent ont été joués. Dans le cas de l’application, cette valeur est toujours « true ».
  - « shuffle » est un booléen qui définit si les morceaux de la playlist doivent être joués de façon aléatoire.
  - « priority » est une valeur entière non signée qui définit la hauteur de la priorité de l’événement. C’est-à-dire, si 2 événements sont prévus au même moment, celui avec la plus haute priorité est joué.
  - Le nom de la playlist jouée (nom du fichier .lst)
- Un horaire : L’horaire définit les paramètres suivants :

<sup>20</sup> Table du temps ou grille horaire en anglais.

- « starttime » est l'heure (format : hh:mm:ss) de début de l'événement.
- « duration » est la durée (format : hh:mm:ss) de lecture de la playlist.
- « repeat » est une valeur entière non signée servant au transcoder pour savoir quel jour doit être lancé cet événement aux horaires donnés. Chaque jour de la semaine a une valeur et l'addition des valeurs des jours sélectionnés donne la valeur de « repeat ». Voici les valeurs définies par ShoutCAST :
  - 1 - Every Sunday
  - 2 - Every Monday
  - 4 - Every Tuesday
  - 8 - Every Wednesday
  - 16 - Every Thursday
  - 32 - Every Friday
  - 64 - Every Saturday
  - 128 - Time periodic : Cette dernière n'est pas encore implémentée. Pour plus d'information, rendez-vous au lien ci-dessus (lien vers le site officiel).

---

#### 6.11.4 GÉNÉRATION DE CONFIGURATION

ShoutCAST (transcoder) utilise un fichier XML pour la gestion de son calendrier. La génération d'un fichier XML sous C# se fait simplement avec un objet XmlDocument et des XmlElement. Voici un exemple de création de fichier de configuration XML pour un calendrier :

```
XmlDocument document = new XmlDocument();
XmlElement root = document.CreateElement("eventlist");
foreach(CalendarEvent ev in this.Events)
{
    XmlElement eventelement = document.CreateElement("event");
    eventelement.SetAttribute("type", "playlist");
    XmlElement playlist = document.CreateElement("playlist");
    playlist.SetAttribute("loopatend", (ev.Loopatend)? "1" : "0");
    playlist.SetAttribute("shuffle", (ev.Shuffle) ? "1" : "0");
    playlist.SetAttribute("priority", ev.Priority.ToString());
    playlist.InnerText = ev.Playlist;
    eventelement.AppendChild(playlist);

    XmlElement calendar = document.CreateElement("calendar");
    calendar.SetAttribute("starttime", ev.StartTime.ToString("hh:mm:ss"));
    calendar.SetAttribute("duration", ev.Duration.ToString("hh:mm:ss"));
    calendar.SetAttribute("repeat", ev.Repeat.ToString());
}
```

```

    eventelement.AppendChild(calendar);

    root.AppendChild(eventelement);
}

document.AppendChild(root);
document.Save(this.FileName);

```

WebradioCalendar.cs

Voici le XML une fois généré :

```

<eventlist>
  <event type="playlist">
    <playlist loopatend="1" shuffle="1" priority="1">HardLol</playlist>
    <calendar starttime="06:00:00" duration="02:00:00" repeat="48" />
  </event>
  <event type="playlist">
    <playlist loopatend="0" shuffle="0" priority="0">Pub migros</playlist>
    <calendar starttime="01:00:00" duration="00:30:00" repeat="0" />
  </event>
  <event type="playlist">
    <playlist loopatend="0" shuffle="0" priority="0">Commercial</playlist>
    <calendar starttime="02:00:00" duration="01:30:00" repeat="8" />
  </event>
</eventlist>

```

Figure 41 - Exemple XML calendrier

#### 6.11.5 AFFICHAGE DU CALENDRIER

L’affichage du calendrier se fait avec le composant décrit dans le chapitre concernant [l’outil utilisé](#) (0). Il a été configuré pour afficher 7 jours (une semaine) ainsi qu’un quadrillage basé sur les demi-heures.

Les événements sont affichés selon la convention suivante :

- Événement de type « Music » :
  - Couleur de bordure et de fond : Bleue
  - Contenu :
    - « *NomDeLevenement (HauteurPriorité)*
    - NomDeLaPlaylist*
    - Shuffle : *TrueOuFalse* »
- Événement de type « Ad » :
  - Couleur de bordure et de fond : Rouge
  - Contenu :
    - « *NomDeLevenement (HauteurPriorité)*
    - NomDeLaPlaylist*
    - Shuffle : *TrueOuFalse* »

C'est lors de l'appel de la méthode « UpdateView » que le calendrier sera rempli. Ce composant a besoin de disposer d'une variable de type List<Appointment> qu'il utilisera afin de se remplir. Cette liste doit donc être remplie lors d'UpdateView et elle est globale à la *vue* AdminView. C'est en réalité l'événement « ResolveAppointments » du composant qui va par la suite lire cette variable et remplir ce dernier :

```
List<Appointment> m_Apps = new List<Appointment>();  
foreach (Appointment m_App in this.EventsCalendar)  
    m_Apps.Add(m_App);  
  
args.Appointments = m_Apps;
```

AdminView.cs

L'événement est appelé lorsque le composant doit se rafraîchir. Il permet de personnaliser la façon dont on veut remplir le composant.

Dans le cadre du projet, une classe nommée « **EventAppointment** » a été créée. Elle hérite de la classe « Appointment » proposée par le composant, mais elle propose une propriété de type Playlist afin de stocker l'objet Playlist lié à l'événement ainsi qu'une autre propriété de type CalendarEvent qui stocke l'objet CalendarEvent de l'événement. Cela est utile pour la suppression et modification. Cette classe reprend donc à l'identique la forme de sa classe parente, mais seules 2 propriétés sont ajoutées.

Comme décrit précédemment, ShoutCAST utilise une valeur nommée « repeat » pour stocker quels sont les jours où l'événement doit se jouer. Calculer cette valeur se fait à l'aide des valeurs données, dans la documentation, pour chaque jour. Mais lors de l'affichage dans le calendrier et la création des EventAppointment, il faut décoder cette valeur afin d'en ressortir les jours sélectionnés. Pour se faire, une méthode utilisant des masques binaires est utilisée. La classe CalendarEvent propose une méthode « GetSelectedDays » qui retourne une structure « DayWeek » qui contient tous les jours de la semaine sous forme de booléen. Voici un exemple d'utilisation d'un masque :

```
DayWeek dow = new DayWeek();  
dow.Monday = Convert.ToBoolean(this.Repeat & MONDAY_MASK);
```

CalendarEvent.cs

La structure ainsi remplie va être récupérée lors d'UpdateView et utilisée de cette manière :

```
DayWeek dw = ev.GetSelectedDays();  
bool[] days = dw.ToArray();  
for(int i = 0; i < 7;i++)  
{  
    if(days[i])
```

```
{  
  
//Création d'un EventAppointment et ajout à la liste utilisée par le composant  
Calendar  
  
}
```

AdminView.cs

Le principe est le suivant : une boucle est programmée pour tourner 7 fois (pour chaque jour de la semaine). Pour chaque tour (chaque jour), le programme teste si le jour actuel a une valeur booléenne vraie dans le tableau retourné par la structure DayWeek rempli à l'aide de la classe de l'événement courant.

Si la condition est vraie, un nouvel EventAppointment peut être créé, rempli et ajouté à la variable globale contenant les événements du calendrier.

---

#### 6.11.6 SÉLECTION DEPUIS LE CALENDRIER

Le composant permet de faire une sélection (d'une heure à une autre heure) avec la souris. L'événement « SelectionChanged » est appelé si tel est le cas. Les informations du formulaire de création d'événements sont automatiquement mises à jour en fonction de la sélection de l'utilisateur.

---

#### 6.11.7 CRÉATION D'UN ÉVÉNEMENT

Le formulaire de création est composé de 2 MaskedTextBox pour les champs « start time » et « duration ». Celle-ci permet de garantir que le format d'entrée de ces 2 propriétés sera juste afin de créer des TimeSpan. Le masque utilisé est le suivant « 00:00:00 ». Afin de récupérer ces valeurs, un autre test doit être effectué : vérifier que le format de l'heure est réaliste (par exemple, pas de cette façon : 89:20:67. C'est une heure qui n'est pas possible). La méthode statique « TryParse » de la classe TimeSpan permet cette vérification :

```
TimeSpan start = new TimeSpan();  
  
if (!TimeSpan.TryParse(mtbStartTime.Text, out start))  
{  
  
    MessageBox.Show("Start time format is not correct.", "Error");  
  
    return;  
  
}
```

AdminView.cs

Si le test passe, la méthode remplira la variable « start » avec les valeurs du MaskedTextBox.

Une durée minimum de 1 minute est exigée pour un événement. Aussi, le nom de l'événement doit être unique.

La génération de la valeur « repeat » s'effectue en fonction des cases cochées (jours de la semaine) par l'utilisateur. La méthode « GetRepeatValue » présente dans la *vue* va calculer et retourner la valeur. L'enum DayValue fournit les valeurs de chaque jour (défini par ShoutCAST et défini [précédemment](#)) :

```
int repeat = 0;

repeat += (ckbMonday.Checked) ? (int)DayValue.Monday : 0;
```

Et ainsi de suite pour chaque jour. Si la valeur retournée est 0, c'est que l'utilisateur n'a sélectionné aucun jour. Un message d'erreur apparaît. La valeur « loopatend » est toujours vraie.

Ensuite, l'ajout à la base de données et au *model* s'effectue de la même façon que pour les autres éléments du logiciel (ajout à la bdd, récupération du nouvel identifiant, ajout à l'objet puis ajout au *model*. Pour finir, UpdateObservers pour mettre à jour les *vues*. La méthode GenerateConfigFile est aussi appelée pour mettre à jour le fichier XML du calendrier avec les nouvelles valeurs.

---

#### 6.11.8 MODIFICATION D'UN ÉVÉNEMENT

Le composant Calendar permet de manipuler ses éléments avec la souris afin de les déplacer ou modifier leur longueur. Pour le moment, seuls la modification de la longueur, le commencement et les jours d'un événement sont modifiables (voir le [chapitre fonctionnel](#)). Les règles qui y sont décrites ont été faites pour correspondre le mieux possible avec le système de calendrier de ShoutCAST.

Lorsque l'utilisateur déplace un élément de l'événement et le change de jour, l'événement « MouseUp » du composant est appelé. En premier temps, il est testé s'il y a bien un élément (EventAppointment) sélectionné dans le composant. Puis, le traitement suivant est appliqué :



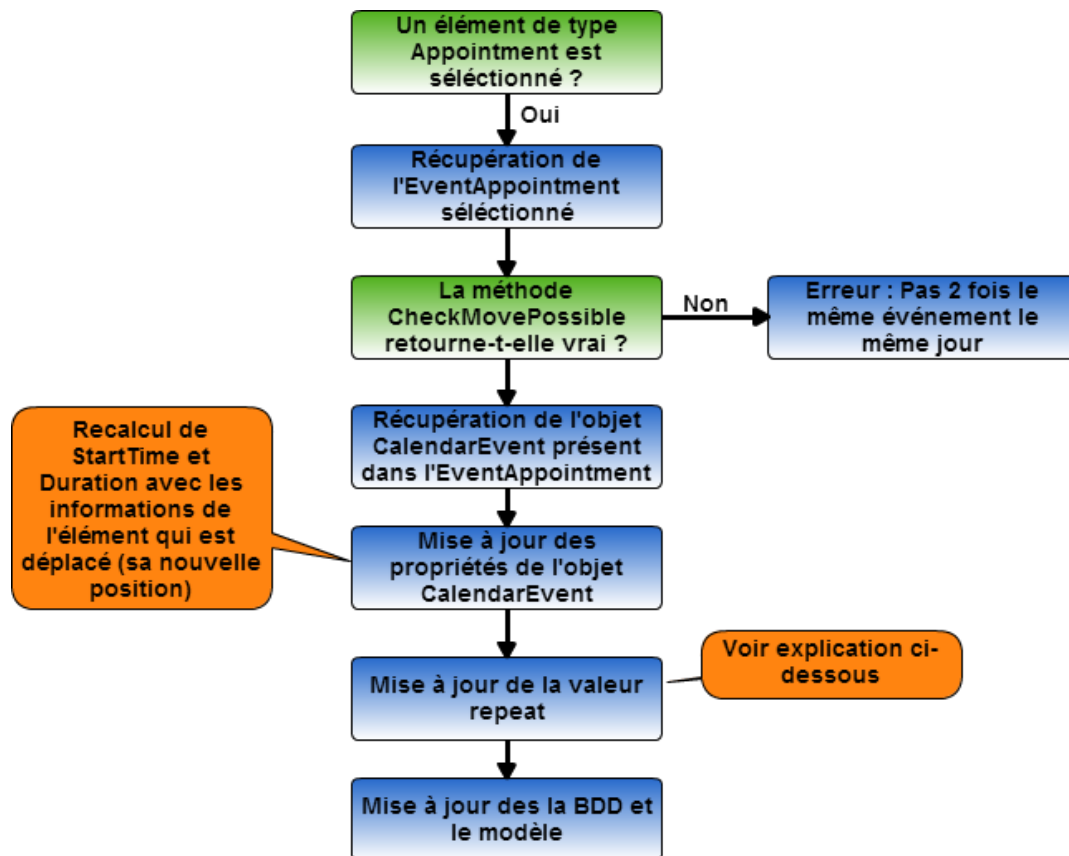


Figure 42 - Modification d'un événement

La méthode « CheckMovePossible » vérifie qu'il n'y a pas déjà un EventAppointment du même événement dans le jour où l'élément a été déplacé. Pour se faire, tous les éléments (EventAppointment) du composant sont vérifiés un par un. Si un d'entre eux a la même date de début (propriété DayOfWeek de celle-ci) et le même identifiant d'événement (stocké dans l'objet CalendarEvent de l'élément), le mouvement sera alors impossible.

Concernant la mise à jour de la valeur « repeat », elle est recalculée par rapport à l'emplacement des différents éléments d'un événement dans le composant. Pour se faire, plusieurs étapes sont exécutées :

1. Récupération de tous les éléments (EventAppointment) du composant qui sont concernés par l'événement en question (l'événement déplacé). C'est la méthode « GetAllRelatedAppointment » de la *vue* qui va rechercher dans la liste d'« EventAppointment » utilisée par le composant.
2. Ensuite, la méthode « GetRepeatValueFromAppointment » va calculer et retourner la valeur de repeat en fonction des « EventAppointment » qui lui sont donnés en paramètre. Pour chaque EventAppointment, il est vérifié (pour chaque jour de la semaine) si le « DayOfWeek » de sa propriété « StartDate » est égal au jour en question. De cette façon :  
`repeat += (ev.StartDate.DayOfWeek==DayOfWeek.Monday)?(int)DayValue.Monday : 0;`

Pour finir, les configurations sont régénérées.

---

#### 6.11.9 SUPPRESSION D'UN ÉVÉNEMENT

Un clic droit sur un élément appelle l'événement « `MouseClicked` » du composant. Il est vérifié que le bouton cliqué est bien le droit et si l'élément de type `Appointment` est bien sélectionné sur le composant. Une boîte de dialogue demande confirmation à l'utilisateur puis la méthode « `DeleteEvent` » du *controller* est appelé. L'objet `CalendarEvent` stocké dans l'`EventAppointment` est donné en paramètre. Cette méthode ira appeler la même méthode dans le *model* qui s'occupera, lui, de supprimer l'événement dans la BDD et ses propriétés. `UpdateObservers` est ensuite appelé ainsi que la régénération de configuration.

## 6.12 TRANSCODERS

### 6.12.1 CLASSES UTILISÉES

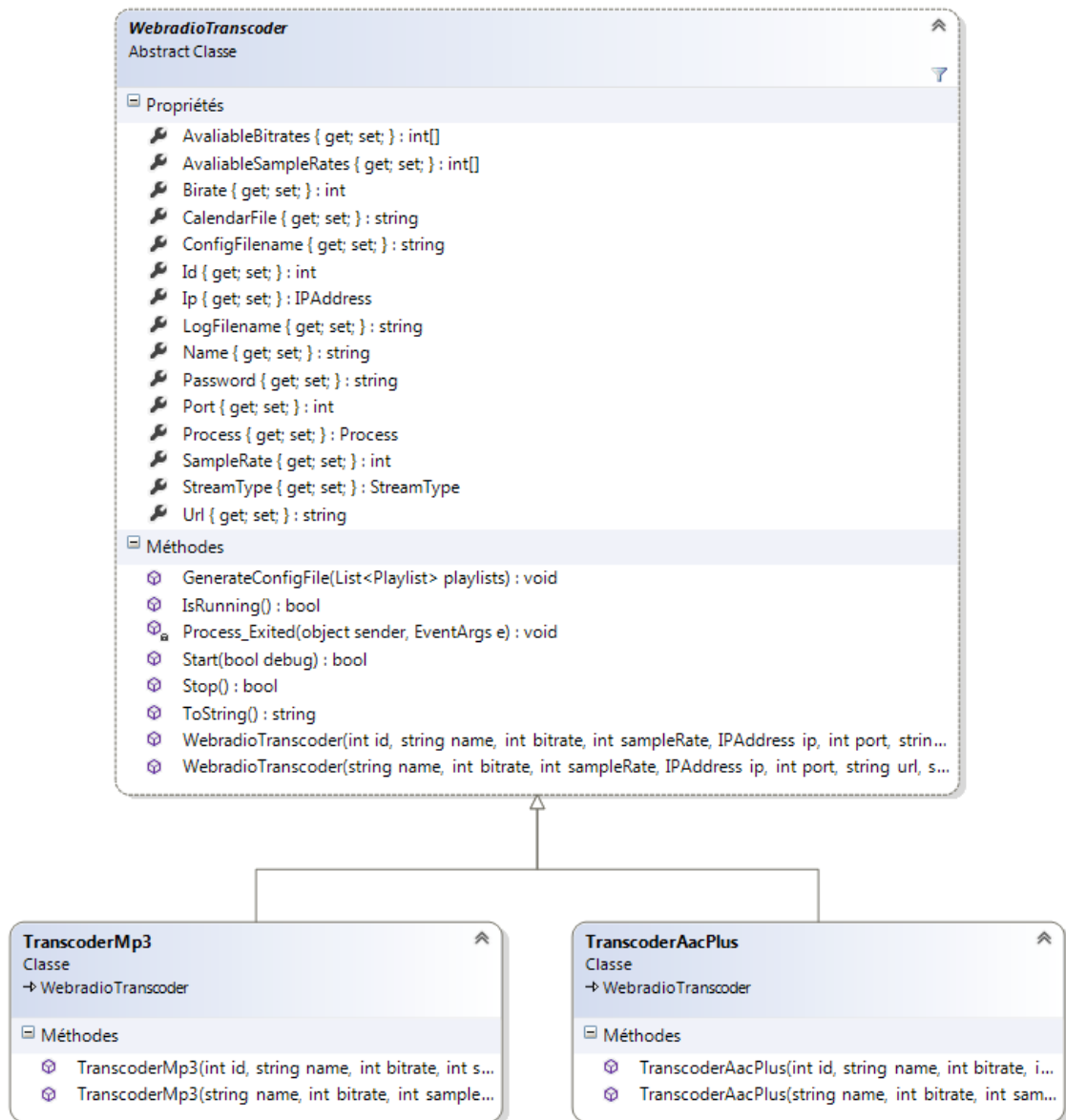


Figure 43 - Classes transcodeurs

## 6.12.2 OUTIL UTILISÉ

ShoutCAST propose un outil en lignes de commandes nommées « TransCAST » qui est un transcodeur :

Le transcodage, en vidéo ou en audio, est le fait de changer le format de codage d'un média (voir aussi codage et codec) utilisé pour comprimer ou encapsuler un média audio ou vidéo dans un fichier ; ou transporter un signal analogique ou numérique. On notera qu'il ne s'agit pas d'un codage au sens strict du terme, car le plus souvent la transformation comporte des pertes.

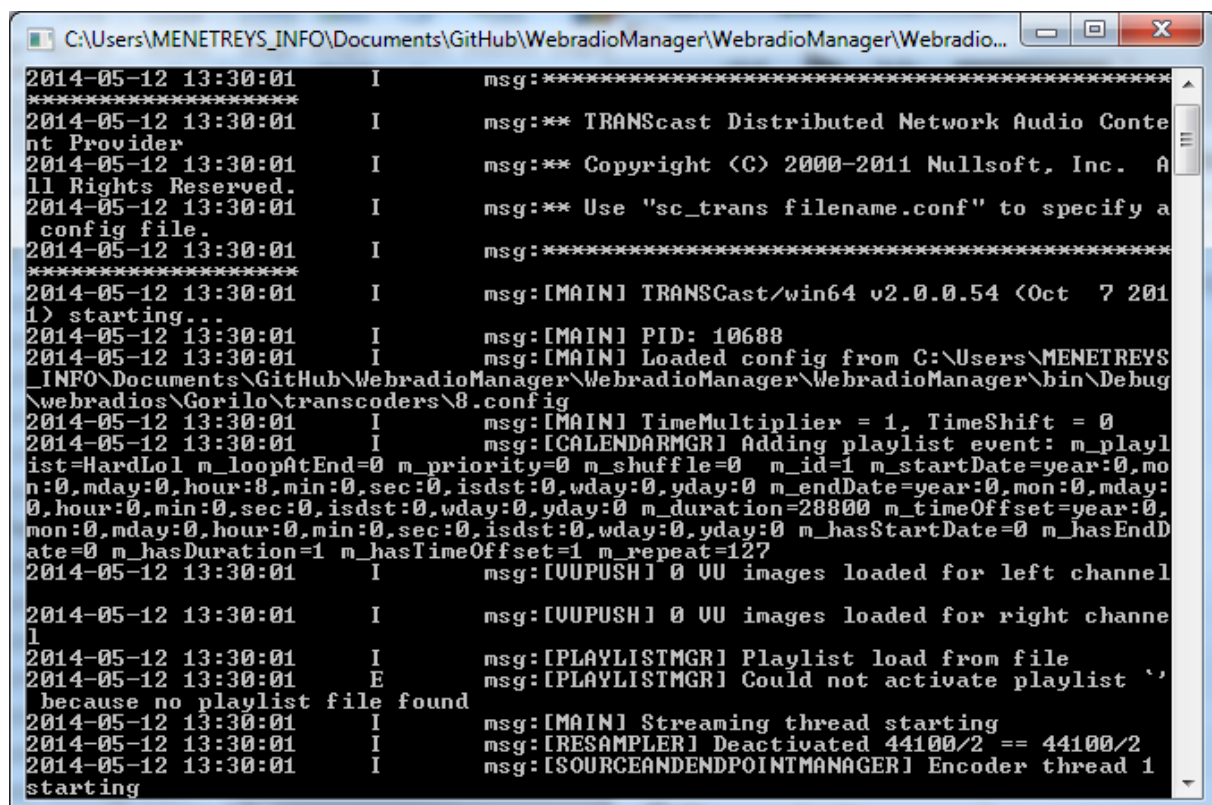
Source : Wikipédia

Dans le cas d'une webradio, le transcodeur sert à créer un flux à partir de fichiers audio afin de l'envoyer sur un serveur (serveur de diffusion) qui s'occupera de la diffusion aux clients/auditeurs. Un transcodeur peut disposer de playlists (listes de lecture) ainsi que d'un calendrier définissant des événements sur une semaine. Ces éléments ont été documentés précédemment. Le logiciel TransCAST fourni par ShoutCAST prend donc en charge toute la gestion de playlists, d'horaires et de lecture de musiques ainsi que le transcodage et la création du flux final.

Le lancement de cet outil se fait de la manière suivante :

```
sc_trans.exe myconfig.config
```

La première partie est le chemin vers l'exécutable. La seconde est le chemin vers le fichier de configuration. Voici la console du logiciel au lancement



```
C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioManager\Webradio...
2014-05-12 13:30:01 I msg:*****
*****
2014-05-12 13:30:01 I msg:** TRANSCast Distributed Network Audio Conte
nt Provider
2014-05-12 13:30:01 I msg:** Copyright (C) 2000-2011 Nullsoft, Inc. A
ll Rights Reserved.
2014-05-12 13:30:01 I msg:** Use "sc_trans filename.conf" to specify a
config file.
2014-05-12 13:30:01 I msg:*****
*****
2014-05-12 13:30:01 I msg:[MAIN] TRANSCast/win64 v2.0.0.54 (Oct 7 201
1) starting...
2014-05-12 13:30:01 I msg:[MAIN] PID: 10688
2014-05-12 13:30:01 I msg:[MAIN] Loaded config from C:\Users\MENETREYS
_INFO\Documents\GitHub\WebradioManager\WebradioManager\WebradioManager\bin\Debug
\Webradios\Gorilo\transcoders\8.config
2014-05-12 13:30:01 I msg:[MAIN] TimeMultiplier = 1, TimeShift = 0
2014-05-12 13:30:01 I msg:[CALENDARMGR] Adding playlist event: m_playl
ist=HardLol m_loopAtEnd=0 m_priority=0 m_shuffle=0 m_id=1 m_startDate=year:0,mo
n:0,mday:0,hour:8,min:0,sec:0,isdst:0,wday:0,yday:0 m_endDate=year:0,mon:0,mday:
0,hour:0,min:0,sec:0,isdst:0,wday:0,yday:0 m_duration=28800 m_timeOffset=year:0,
mon:0,mday:0,hour:0,min:0,sec:0,isdst:0,wday:0,yday:0 m_hasStartDate=0 m_hasEndD
ate=0 m_hasDuration=1 m_hasTimeOffset=1 m_repeat=127
2014-05-12 13:30:01 I msg:[VUPUSH] 0 UU images loaded for left channel
2014-05-12 13:30:01 I msg:[VUPUSH] 0 UU images loaded for right channe
l
2014-05-12 13:30:01 I msg:[PLAYLISTMGR] Playlist load from file
2014-05-12 13:30:01 E msg:[PLAYLISTMGR] Could not activate playlist ''
because no playlist file found
2014-05-12 13:30:01 I msg:[MAIN] Streaming thread starting
2014-05-12 13:30:01 I msg:[RESAMPLER] Deactivated 44100/2 == 44100/2
2014-05-12 13:30:01 I msg:[SOURCEANDENDPOINTMANAGER] Encoder thread 1
starting
```

Figure 44 - ShoutCAST transcoder console

Cet outil propose une API<sup>21</sup> web accessible via le port défini dans le [fichier de configuration](#) (adminport). Certaines informations peuvent être récupérées et modifiées. Plus d'informations sur le site officiel : [http://wiki.winamp.com/wiki/SHOUTcast\\_Transcoder AJAX api Specification](http://wiki.winamp.com/wiki/SHOUTcast_Transcoder AJAX_api_Specification)

Pour le projet, le protocole Ultravox 2.1 a été sélectionné, car il s'agit de la dernière version et la plus stable. Il est évidemment possible de changer cette valeur. Le transcoder ShoutCAST propose 3 protocoles différents. Chacun a une valeur à configurer dans le fichier de configuration :

```
1 = SHOUTcast 1 (Legacy)
2 = Ultravox (Ultravox 2.0)
3 = SHOUTcast 2 (Ultravox 2.1)
```

Plus d'informations sur cette page :

[http://wiki.winamp.com/wiki/SHOUTcast\\_2\\_\(Ultravox\\_2.1\)\\_Protocol\\_Details](http://wiki.winamp.com/wiki/SHOUTcast_2_(Ultravox_2.1)_Protocol_Details)

#### 6.12.3 DÉFINITION DES BITRATES, TAUX D'ÉCHANTILLONNAGE ET TYPE D'ENCODER

Les types d'encodeur sont définis par un énumérateur « StreamType » :

```
public enum StreamType
{
    MP3 = 1,
    AACPlus = 2,
}
```

StreamType.cs

Les valeurs assignées sont statiques et sont identiques aux identifiants de ces valeurs dans la base de données (table tstreamtype).

Concernant les bitrates et les taux d'échantillonnage, 2 tableaux statiques sont définis dans la classe « WebradioTranscoder » :

```
private static int[] _availableBitrates = { 64000, 96000, 128000, 256000 };
private static int[] _availableSampleRates = { 44100 };
```

WebradioTranscoder.cs

#### 6.12.4 FICHIER DE CONFIGURATION ET LOG

---

<sup>21</sup> Application Programming Interface (interface de programmation) : un ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

Voici la liste des paramètres utilisés et configurés dans le fichier de configuration d'un transcoder:

- Logfile : Le chemin vers le fichier de log
- Encoder\_1 : Le type d'encodeur (mp3 ou aacp)
- Bitrate\_1 : Le bitrate de l'encodeur
- Adminport : Le port pour accéder au transcoder (pour envoi de requêtes Ajax<sup>22</sup>). Cette valeur est définie avec la valeur constante « DEFAULT\_ADMIN\_PASSWORD ». Actuellement, elle vaut 9000.
- Adminuser : Nom d'utilisateur pour les requêtes
- Adminpassword : Mot de passe pour les requêtes
- Outprotocol\_1 : Sélection de protocoles utilisés pour le flux. Toujours configuré à 3, car cela correspond au protocole le plus récent : SHOUTcast 2 (Ultravox 2.1). Le serveur distant doit prendre en charge ce protocole
- Serverip\_1 : Adresse IP du serveur de diffusion
- Serverport\_1 : Port du serveur de diffusion
- Password\_1 : Mot de passe du serveur de diffusion
- Streamid\_1 : définit l'id du flux
- Streamtitle : Le nom du flux
- Streamurl : L'adresse du flux (l'adresse du site web de la webradio par exemple)
- Genre : Le genre de musique du flux. Toujours configuré à « Misc ».
- Calendarfile : Le chemin vers le fichier de calendrier de la webradio (tous les transcodeurs d'une même webradio pointent sur le même calendrier)

À savoir que le numéro écrit comme ceci « \_X » après certains paramètres est un numéro qui permet de regrouper tous les paramètres à l'encoder de ce numéro (défini par le paramètre « streamid »). Car un transcoder peut, dans certains cas, avoir différents encodeurs et serveur de diffusion. Dans le cas de WebradioManager, il a été décidé qu'un transcoder avait un seul serveur de diffusion.

En plus de ces paramètres, il faut que le fichier de configuration contienne les playlists utilisées par le calendrier défini. Ces informations sont par couple de 2 paramètres :

- Playlistfilename\_X : Le nom de la playlist (le nom qui est inscrit [dans le calendrier](#))
- Playlistfilepath\_X : Chemin vers le fichier de cette playlist

Le X correspond à un numéro qui permet de retrouver les 2 paramètres qui vont de pair. C'est-à-dire que chacun des 2 paramètres doit avoir le même numéro. Pour générer ces paramètres, un paramètre de type « List<Playlist> » est passé en paramètre à la méthode de génération de configuration.

Les fichiers de configuration sont stockés dans le dossier « transcoders » de la webradio (voir le chapitre concernant [les dossiers/fichiers](#) (0)).

---

<sup>22</sup> Voir [glossaire](#)

Chaque fichier est nommé « IdentifiantDuTranscoder.config » (identifiant dans la base de données).  
La même chose pour son fichier de log, mais avec l'extension « .log »

Exemple de configuration :

```
logfile=C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioManager\Web  
radioManager\bin\Debug\web radios\Gorilo\transcoders\7.log  
  
encoder_1=aacp  
  
bitrate_1=256000  
  
adminport=9000  
  
adminuser=admin  
  
adminpassword=admin  
  
outprotocol_1=3  
  
serverip_1=127.0.0.1  
  
serverport_1=8000  
  
password_1=lol  
  
streamid_1=1  
  
streamtitle=Gorilo 256  
  
streamurl=www.hardstylefm.ch  
  
genre=Misc  
  
calendarfile=C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioManager  
\WebradioManager\bin\Debug\web radios\Gorilo\calendar.xml  
  
playlistfilename_1=Pub migros  
  
playlistfilepath_1=C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioMa  
nager\WebradioManager\bin\Debug\web radios\Gorilo\playlists\Pub migros.lst  
  
playlistfilename_2=Commercial  
  
playlistfilepath_2=C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioMa  
nager\WebradioManager\bin\Debug\web radios\Gorilo\playlists\Commercial.lst  
  
playlistfilename_3=HardLol  
  
playlistfilepath_3=C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioMa  
nager\WebradioManager\bin\Debug\web radios\Gorilo\playlists\HardLol.lst  
  
playlistfilename_4=Test  
  
playlistfilepath_4=C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioMa  
nager\WebradioManager\bin\Debug\web radios\Gorilo\playlists\Test.lst
```

### 6.12.5 LICENCE MP3

Pour diffuser en mp3, l'utilisateur doit obtenir une licence comme expliquée dans la documentation du transcodeur ShoutCAST :

[http://wiki.winamp.com/wiki/SHOUTcast\\_DNAS\\_Transcoder\\_2#Registering\\_for\\_MP3\\_Stream\\_Encoding](http://wiki.winamp.com/wiki/SHOUTcast_DNAS_Transcoder_2#Registering_for_MP3_Stream_Encoding)

Malheureusement, les licences ne sont plus distribuées pour le moment. L'utilisateur peut tout de même sélectionner MP3, mais doit entrer manuellement sa licence dans le fichier de configuration si il en possède une. De ce fait, la diffusion en MP3 n'est pas fonctionnelle à 100%.

### 6.12.6 CRÉATION D'UN TRANSCODER

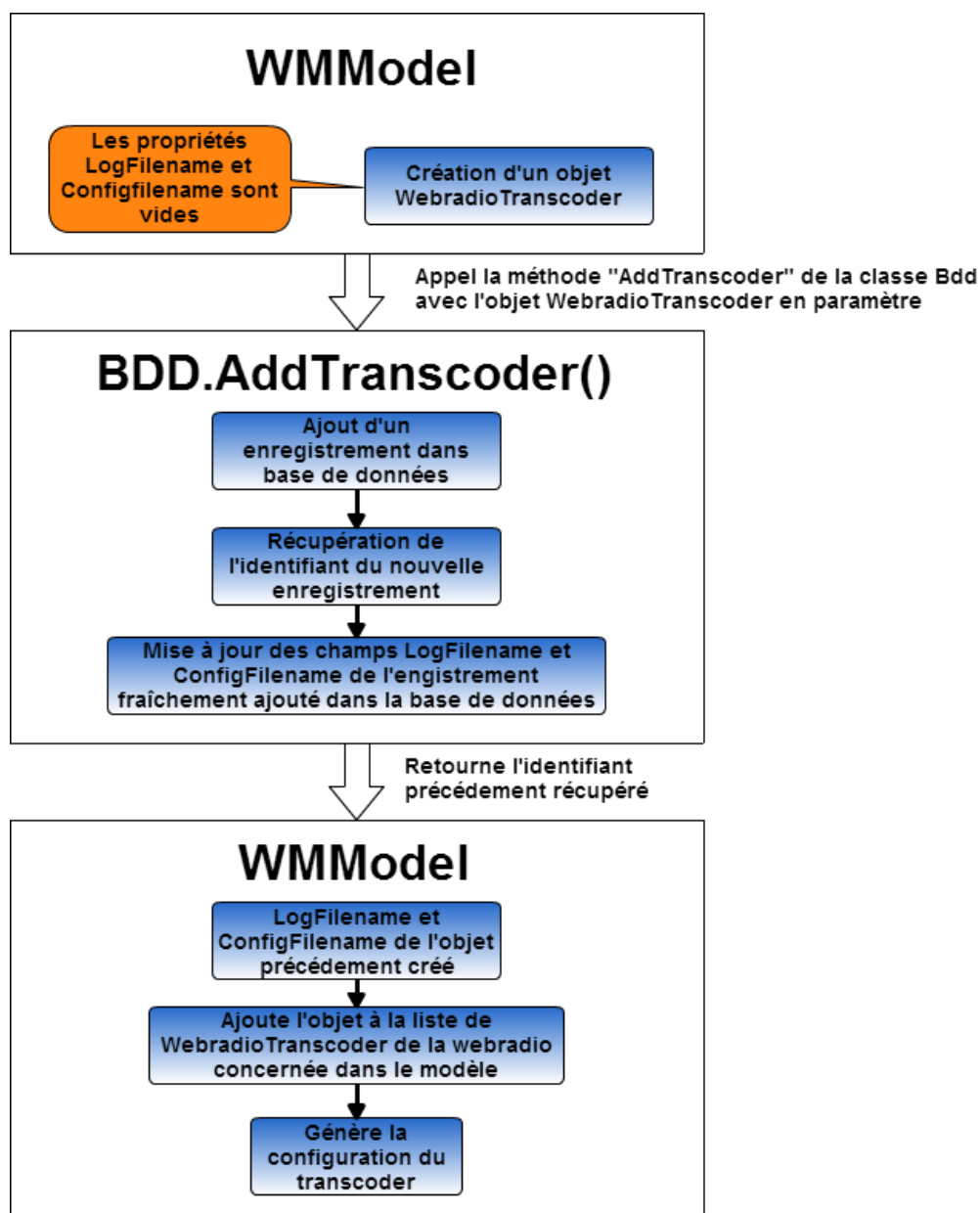


Figure 45 - Schéma création d'un transcoder



Le schéma ci-dessus explique le fonctionnement global de l'ajout d'un nouveau transcoder à une webradio au niveau du *model*. En amont, certaines vérifications sont effectuées au niveau de la *vue*. Pour commencer, les champs concernant le nom du flux et le mot de passe du serveur de diffusion doivent être remplis. L'adresse IP rentrée doit être correcte et elle est vérifiée avec la méthode « TryParse » de la classe IPAddress. L'utilisateur peut aussi entrer le nom DNS du serveur. Dans ce cas, le TryParse va retourner *false*.

Ensuite, dans la méthode « AddTranscoder » de la classe Bdd décrite dans le schéma, un premier test fondamental est effectué avant toute manipulation. Le transcodeur créé doit avoir un nom unique (nom du flux) pour cette webradio. La méthode « TranscoderExists » de la même classe retourne un booléen. Si le nom du flux existe déjà, la méthode retourne directement la valeur constante « ERROR ». De son côté, le *model* va recevoir l'identifiant du transcoder qui été sensé être ajouté. Il vérifie donc que l'id reçu n'est pas égale à la valeur constante « ERROR » de la classe Bdd. Si elle est égale, le *model* retourne « *false* » et la *vue*, dans ce cas, affiche un message d'erreur à l'utilisateur.

---

#### 6.12.7 RÉOLUTION DE NOM

Les 2 boutons « resolve » permettent de résoudre le nom DNS donné par l'utilisateur dans le champ « Server IP ». La méthode « GetResolvedConnexionIPAddress » de la *vue* AdminView (trouvée sur le site : <http://www.codeproject.com/Tips/440861/Resolving-a-hostname-in-Csharp-and-retrieving-IP-v>) va retourner un booléen si l'opération a réussi ou non. La variable « out » de type IPAddress sera remplie avec une nouvelle instance contenant l'adresse trouvée.

---

#### 6.12.8 AFFICHAGE D'UN TRANSCODER

Lors de la sélection d'un transcoder dans la liste proposée située dans l'onglet « transcoders », la méthode de la *vue* nommée « ShowTranscoderInfos » va se charger de remplir la partie d'édition de transcodeur (partie de droite de l'onglet) avec les informations du transcoder sélectionné.

Le reste des informations sont emplies de façon standard. La partie « status » affichant l'état du transcoder sélectionné est aussi remplie :

```
bool running = transcoder.IsRunning();  
lblStatusTranscoder.Text = (running) ? "On" : "Off";  
lblStatusTranscoder.ForeColor = (running) ? Color.Green : Color.Red;  
btnStartTranscoder.Enabled = (running) ? false : true;  
btnStopTranscoder.Enabled = (running) ? true : false;
```

AdminView.cs

---

#### 6.12.9 MODIFICATION D'UN TRANSCODER

Pour la mise à jour des informations d'un transcodeur, la procédure ressemble à celle de l'ajout. Les champs concernant le nom et le mot de passe du serveur de diffusion doivent être remplis et l'adresse IP rentrée doit être valide. Ensuite, le transcoder sélectionné dans le ListBox est récupéré

et les informations contenues dans ses propriétés sont mises à jour avec celles du formulaire de modification. L'objet modifié est ensuite envoyé au *model* via le *controller* afin d'apporter les modifications dans la base de données et les données du *model*.

Si le transcoder était en fonctionnement, il est arrêté et sera redémarré après la mise à jour des informations et la régénération de la configuration.

---

#### 6.12.10 SUPPRESSION D'UN TRANSCODER

Lors de la suppression d'un transcoder, celui sélectionné dans le ListBox (affichant les transcodeurs disponibles) est envoyé au *model* via le *controller*. Les informations concernant le transcodeur sont supprimées de la base de données, puis, les fichiers (configuration et log) sont supprimés et pour finir, l'objet est supprimé du *model*.

---

#### 6.12.11 EXÉCUTION ET FERMETURE

La classe WebradioTranscoder propose 3 méthodes concernant la gestion du processus du transcodeur qu'elle représente :

- Start : Lance le processus
- Stop : Arrête le processus
- IsRunning : Retourne un booléen. « True » si le processus est en fonctionnement et « false » dans le cas contraire.

Le lancement d'un transcoder peut se faire en mode « debug ». C'est le paramètre booléen de la méthode « start » qui définit si ce mode est activé ou non. Si il est actif, le processus se lance avec une fenêtre console affichant le log du transcoder qui est directement minimisée dans la barre de tâche. Sans ce mode, aucune fenêtre ne s'affiche. Voici la préparation au lancement du processus grâce à la classe ProcessStartInfo (voir aussi le chapitre concernant le [serveur](#)) :

```
ProcessStartInfo StartInfo = new ProcessStartInfo(Directory.GetCurrentDirectory() +
SC_SERVER_FILENAME)

{
    CreateNoWindow = true,
    WindowStyle =
(debug)?ProcessWindowStyle.Minimized:ProcessWindowStyle.Hidden,
    Arguments = Directory.GetCurrentDirectory() + "\\\" +
this.ConfigFilename.Replace('/', '\\')
};
```

WebradioTranscoder.cs

Pour l'arrêt d'un processus, la méthode « stop » va en premier voir si le processus fonctionne et répond toujours en tâche de fond. Si c'est le cas, le processus est arrêté. Le calendrier est à nouveau généré, car il a été remarqué que ce dernier était modifié par le transcoder lui-même lors de l'arrêt du processus.

Deux cas d'erreur au lancement sont possibles :

- Un processus « fantôme » de ce transcodeur est encore dans les processus système et le programme n'arrive pas à l'arrêter. Dans ce cas, l'utilisateur doit aller arrêter le processus lui-même (via le gestionnaire de tâche windows).
- Le fichier exécutable ShoutCAST « sc\_trans.exe » n'est plus disponible.

Pour plus d'informations concernant la gestion des différents processus, rendez-vous au chapitre « [gestion des processus](#) ».

---

#### 6.12.12 ADMINISTRATION WEB (WEBLET)

La fonctionnalité « next track » permet de passer à la musique suivante dans la playlist qui est jouée par le transcodeur sélectionné. Cela est possible grâce à l'API Ajax proposée par TransCAST. Voici un exemple de création d'une requête vers l'API :

```
WebClient wb = new WebClient();  
  
var data = new NameValueCollection();  
data["op"] = "nexttrack";  
data["seq"] = "45";  
  
wb.Credentials = new NetworkCredential(WebradioTranscoder.DEFAULT_ADMIN,  
WebradioTranscoder.DEFAULT_ADMIN_PASSWORD );  
  
var response = wb.UploadValues("http://127.0.0.1:"+this.AdminPort+"/api", "POST",  
data);
```

WebradioTranscoder.cs

Chaque action possible sur l'API se compose de la même façon. C'est une requête de type POST sur l'adresse de l'API du transcodeur (IP + PortAdministration /api). Cette requête se compose de 2 valeurs : « op » qui correspond au nom de l'opération et « seq » qui correspond à une valeur (parfois utile pour l'action et parfois non). Il faut être authentifié pour utiliser l'API, c'est pour cela que la requête est effectuée avec des *credentials*<sup>23</sup>.

L'exemple ci-dessus permet de dire au transcodeur de passer à la musique suivante de la playlist actuellement jouée.

---

#### 6.12.13 CAPTURE LIVE

Afin d'effectuer une capture live, le transcodeur ShoutCAST a besoin de savoir quel périphérique audio utiliser pour la capture. La méthode « UpdateAudioDevices » de la *vue* permet de lister les périphériques audio de l'ordinateur dans le ComboBox prévu à cet effet :

---

<sup>23</sup> Un couple nom d'utilisateur/mot de passe servant à accéder à une ressource réseau.

```
ManagementObjectSearcher objSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_SoundDevice");

ManagementObjectCollection objCollection = objSearcher.Get();

cmbAudioDevice.Items.Clear();

foreach (ManagementObject obj in objCollection)
{
    foreach (PropertyData property in obj.Properties)
    {
        if (property.Name == "Caption")
            cmbAudioDevice.Items.Add(property.Value);
    }
}
```

AdminView.cs

Ce code a été trouvé ici : <http://stackoverflow.com/questions/1525320/how-to-enumerate-audio-out-devices-in-c-sharp>

L'API Ajax du transcoder permet d'activer ou de désactiver la capture pendant l'exécution de ce dernier ainsi que de configurer le périphérique audio. La classe WebradioTranscoder dispose d'une méthode « SetCaptureMode » prenant un booléen qui détermine si cette fonctionnalité doit être activée ou non, ainsi qu'une chaîne de caractères contenant le nom du périphérique cible, en paramètres.

Plus d'informations sur l'activation via l'API :

[http://wiki.winamp.com/wiki/SHOUTcast\\_Transcoder\\_AJAX\\_api\\_Specification#Capture](http://wiki.winamp.com/wiki/SHOUTcast_Transcoder_AJAX_api_Specification#Capture)

Cette fonctionnalité ne semble pas tout à fait bien fonctionner. En effet, il se trouve que sur l'ordinateur de développement, malgré le listing des périphériques audio, le transcoder ne trouve pas ce dernier lors de l'activation de la capture.

---

#### 6.12.14 HISTORIQUE

L'historique de chaque transcodeur est enregistré dans la base de données (table thistory).

L'historique référence quelle musique a été jouée par quel transcoder à quelle date et heure.

Le serveur ShoutCAST propose un historique des morceaux joués (les 10 derniers maximum), mais dans le cas de ce projet, il est préférable de regarder au niveau du transcoder, quelles musiques ont été jouées, car dans le cas d'une diffusion vers un serveur de diffusion distant, le logiciel n'a aucun contrôle sur ce dernier (que ce soit pour l'accès aux données ou alors le fonctionnement du serveur). Malheureusement, les transcodeurs ShoutCAST ne proposent pas d'historique. Une solution alternative a été mise en place afin de détecter un changement de fichier audio sur un transcoder afin de noter ses informations dans la BDD.

La méthode appelée par le timer servant de surveillance des processus (voir le chapitre concernant la [Gestion des processus](#) 6.14) est utilisée afin de vérifier le statut de chaque transcoder. Le statut de ces derniers est récupéré à l'aide de l'API Ajax (voir le [chapitre précédent](#) pour l'envoi de commande) avec la commande « getstatus ». Cette dernière retourne les informations suivantes (XML) :

```
<status>

  <activesource source="playlist|capture|dj|relay">

    <currenttrack/>

    <nexttrack/>

    <name/>

    <file/>

    <ip/>

    <port/>

    <url/>

    <sourcetype/>

    <bitrate/>

    <device/>

    <input/>

    <samplerate/>

    <channels/>

  </activesource>

  <endpointlist>

    <endpoint>

      <name/>

      <bytessent/>

      <status/>

    </endpoint>

    <endpoint>

      <name/>

      <bytessent/>

      <status/>

  </endpointlist>

</status>
```

```
</endpoint>  
</endpointlist>  
  
</status>
```

Les différents champs sont décrits dans la documentation officielle :

[http://wiki.winamp.com/wiki/SHOUTcast\\_Transcoder\\_AJAX\\_api\\_Specification#GetStatus](http://wiki.winamp.com/wiki/SHOUTcast_Transcoder_AJAX_api_Specification#GetStatus)

Dans le cas de l'historique, la valeur indiquée par « currenttrack » est récupérée. Il s'agit du chemin vers le fichier audio de la musique jouée actuellement par le transcoder en question. La classe WebradioManager contient une propriété « CurrentTrack » qui contient, justement, la valeur de la musique jouée. Afin de détecter lorsque le transcoder a changé de musique, un test vérifie si la valeur « currenttrack » retournée par l'API est différente de la valeur stockée dans la propriété « CurrentTrack » du transcoder. Si c'est le cas, le transcoder a donc changé de musique et cette dernière doit être ajoutée à l'historique.

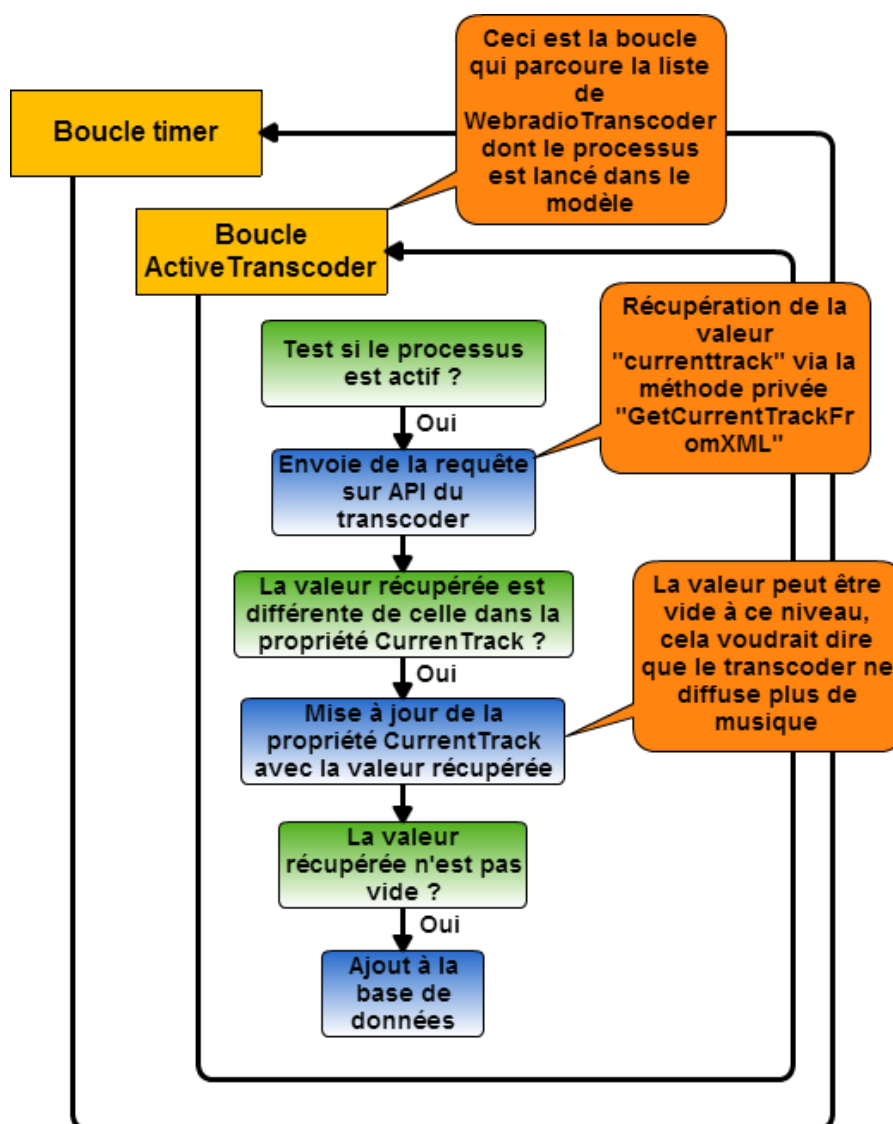


Figure 46 - Schéma gestion de l'historique

L'utilisateur peut ensuite générer un fichier PDF contenant l'historique d'un transcoder. La bibliothèque .NET « iTextSharp » (<https://github.com/itext/itextsharp>) est utilisée pour générer ce document. La méthode « GenerateHistory » du *model* va générer le fichier à l'emplacement sélectionné par l'utilisateur précédemment. L'historique stocké dans la base de données est récupéré sous la forme d'un dictionnaire (string,string) dont la clé correspond à la date et la valeur correspond au chemin vers le fichier joué.

## 6.13 SERVEUR DE DIFFUSION INTERNE

### 6.13.1 CLASSES UTILISÉES

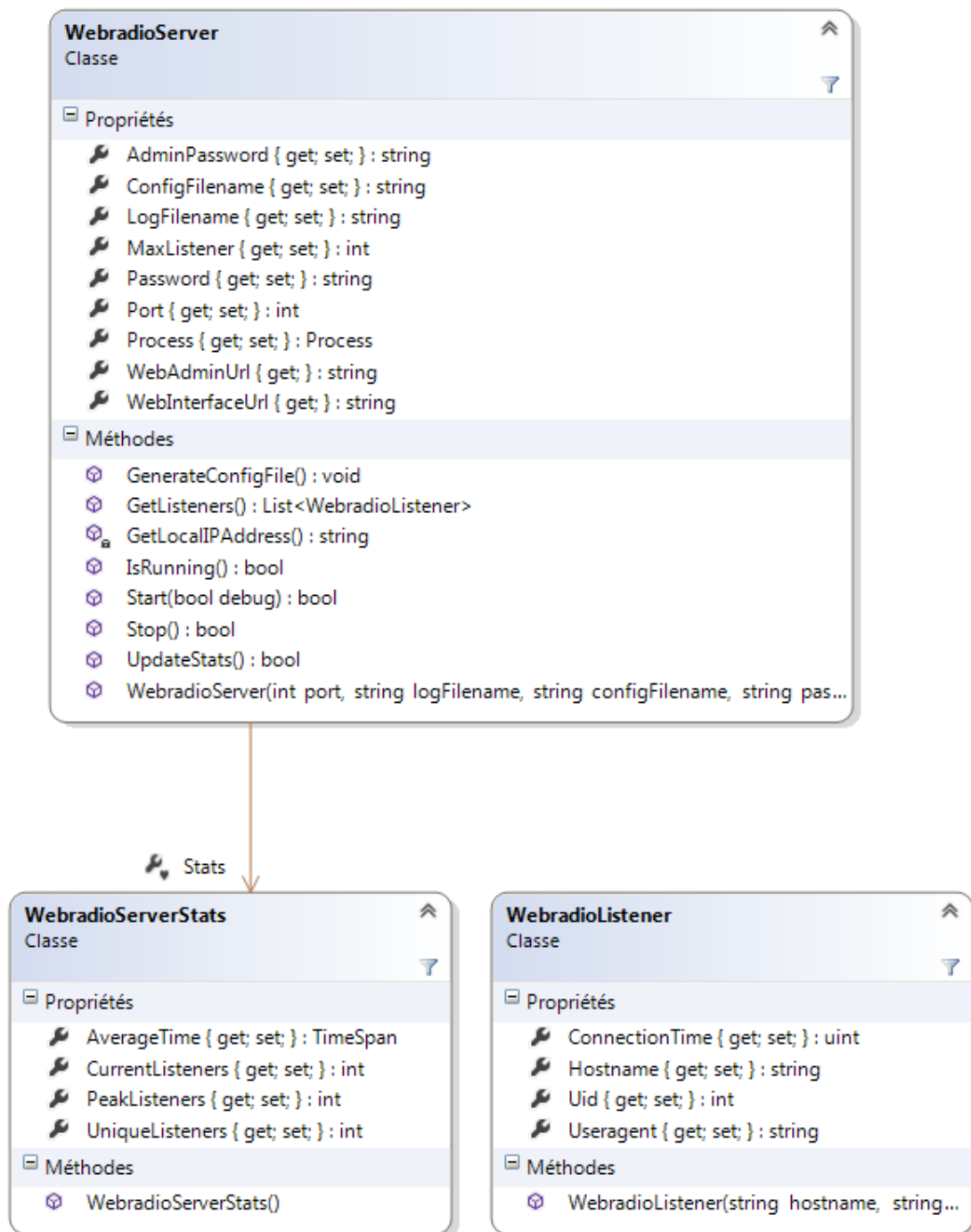


Figure 47 - Classe WebradioServer

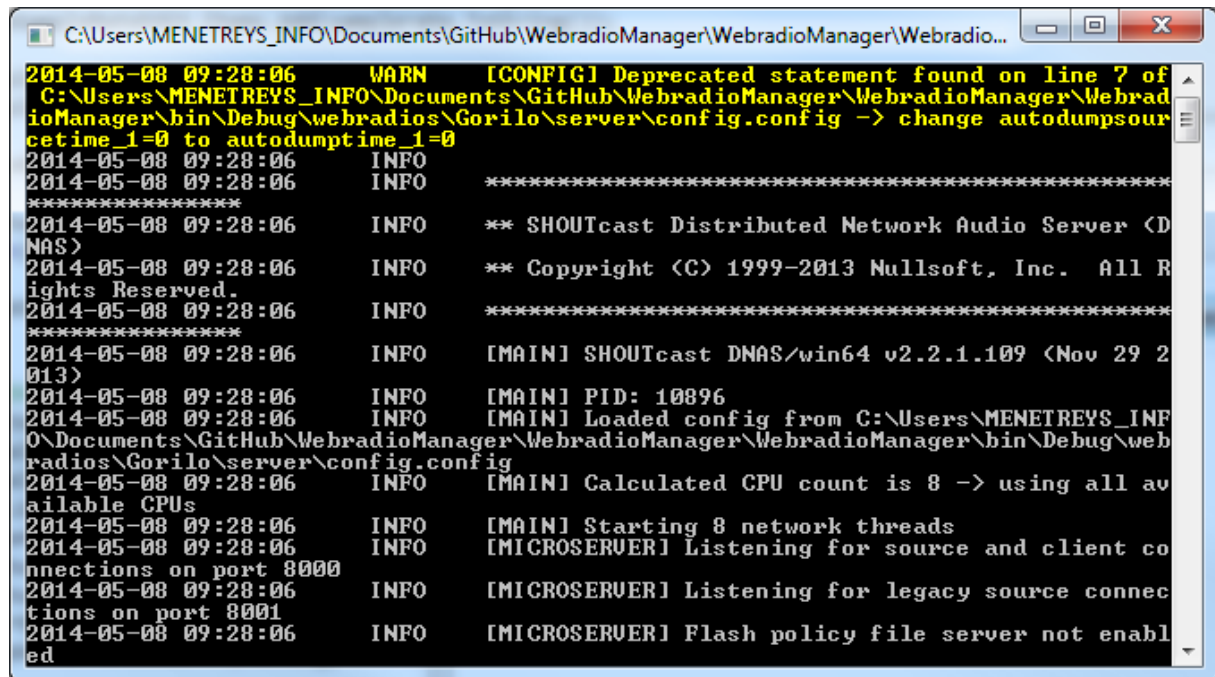
La classe « WebradioListener » permet de transporter les informations concernant un auditeur connecté au serveur afin que la *vue* puisse afficher ses informations.



### 6.13.2 OUTIL UTILISÉ

Shoutcast est avant tout un serveur de diffusion de flux audio ou vidéo (Shoutcast DNAS server 2). L'outil propose un serveur en ligne de commande qui sera utilisée dans ce projet. Le serveur fonctionne avec un fichier de configuration qui lui est donnée en paramètre lorsque l'on l'exécute :

```
sc_serv.exe myconfig.config
```



```
C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioManager\Webradio...
2014-05-08 09:28:06   WARN   [CONFIG] Deprecated statement found on line 7 of
C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioManager\WebradioManager\bin\Debug\web
radioManager\Gorilo\server\config.config -> change autodumpsource
time_1=0 to autodumptime_1=0
2014-05-08 09:28:06   INFO
2014-05-08 09:28:06   INFO   *****
2014-05-08 09:28:06   INFO   ** SHOUTcast Distributed Network Audio Server <D
NAS>
2014-05-08 09:28:06   INFO   ** Copyright (C) 1999-2013 Nullsoft, Inc. All R
ights Reserved.
2014-05-08 09:28:06   INFO   *****
2014-05-08 09:28:06   INFO   [MAIN] SHOUTcast DNAS/win64 v2.2.1.109 <Nov 29 2
013>
2014-05-08 09:28:06   INFO   [MAIN] PID: 10896
2014-05-08 09:28:06   INFO   [MAIN] Loaded config from C:\Users\MENETREYS_INF
O\Documents\GitHub\WebradioManager\WebradioManager\WebradioManager\bin\Debug\web
radioManager\Gorilo\server\config.config
2014-05-08 09:28:06   INFO   [MAIN] Calculated CPU count is 8 -> using all av
ailable CPUs
2014-05-08 09:28:06   INFO   [MAIN] Starting 8 network threads
2014-05-08 09:28:06   INFO   [MICROSERVER] Listening for source and client co
nnections on port 8000
2014-05-08 09:28:06   INFO   [MICROSERVER] Listening for legacy source connec
tions on port 8001
2014-05-08 09:28:06   INFO   [MICROSERVER] Flash policy file server not enabl
ed
```

Figure 48 - ShoutCAST serveur

La documentation concernant la configuration du serveur via le fichier est expliquée ici : [http://wiki.winamp.com/wiki/SHOUTcast\\_DNAS\\_Server\\_2](http://wiki.winamp.com/wiki/SHOUTcast_DNAS_Server_2) . Dans le cadre de ce projet, seules quelques options seront utilisées et configurées. Une partie est configurable par l'utilisateur via l'interface de l'onglet «[server](#)» et l'autre partie est configurée par défaut par l'application :

- L'emplacement du fichier de log
- L'emplacement du fichier de configuration lui-même

Pour plus de détails sur les emplacements des différents fichiers, rendez-vous au chapitre concernant [la structure des dossiers/fichiers](#) 0.

### 6.13.3 CONFIGURATION

Voici la liste des paramètres configurés dans le fichier de configuration d'un serveur dans WebradioManager :

- Logfile : Le chemin vers le fichier de log
- Portbase : Le numéro du port du serveur (sur lequel le transcoder se connecte)
- Password : Le mot de passe de connexion pour une source (un transcoder)

- Adminpassword : Le mot de passe d'administration de l'interface web. Attention, il doit être différent du mot de passe de connexion pour les sources.
- Publicserver : Cette valeur sert à définir sur la webradio sera indexée dans le site de référencement de ShoutCAST. Il n'est pas d'actualité donc la valeur par défaut « always » est définie.
- Maxuser : Le nombre maximum de clients/auditeurs connectés
- Autodumpsourcetime : Nombre de secondes à attendre avant de déconnecter une source (transcoder) si son flux est vide. Dans le cas du projet, la valeur 0 est entrée par défaut (cela correspond à désactiver cette fonction), car il n'est pas voulu que le serveur ferme les connexions vides.

Exemple de fichier de configuration :

```
logfile=C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioManager\WebradioManager\bin\Debug\webradios\Gorilo\server\log.txt

portbase=8000

password=lol

adminpassword=admin

publicserver=always

maxuser=32

autodumpsourcetime=0
```

#### 6.13.4 MISE À JOUR DE LA CONFIGURATION

Lors d'une mise à jour de la configuration du serveur, il est d'abord arrêté puis modifié et ensuite redémarré (seulement dans le cas où il était démarré lors de la sauvegarde de la nouvelle configuration).

Les informations du formulaire sont passées au *model* via le *controller* de la *vue*. Il est testé si le serveur est allumé, si c'est le cas il est éteint et une variable booléenne est enregistrée à « true » afin de savoir, à la fin du traitement, si le serveur doit être redémarré.

Les informations sont modifiées dans la base de données, puis si la modification a réussi, elles sont modifiées dans le *model*. En fin de traitement, la méthode « UpdateObservers » est appelée.

#### 6.13.5 EXÉCUTION ET FERMETURE

L'exécutable du serveur ShoutCAST est lancé depuis la classe « WebradioServer » et sa méthode « Start ». Cette dernière utilise la classe « Process » afin de créer un processus où l'exécutable sera lancé. Une classe nommée « ProcessStartInfo » permet de donner des paramètres d'exécution du processus :

```
ProcessStartInfo StartInfo = new ProcessStartInfo(Directory.GetCurrentDirectory() +
SC_SERVER_FILENAME)
```

```
{  
    CreateNoWindow = true,  
    WindowStyle =  
(debug)?ProcessWindowStyle.Minimized:ProcessWindowStyle.Hidden,  
    Arguments = Directory.GetCurrentDirectory() + "\\\" +  
this.ConfigFilename.Replace('/', '\\')  
};
```

WebradioServer.cs

Il y a 2 modes d'exécution pour le serveur :

- Avec debug : Ouverture de la fenêtre console de l'application serveur ShoutCAST
- Sans debug : Pas d'ouverture de l'application console

Ce mode est choisi par l'utilisateur grâce à la case à cocher qui se trouve dans l'onglet « server ». Dans le code ci-dessus, la variable booléenne « debug » va définir si la propriété « WindowStyle » doit être « minimized » (ouverte, mais minimisée) ou « hidden » (pas de fenêtre apparente). Dans le premier cas, le debug est activé. Le processus est ensuite lancé via `Process.Start(StartInfo)` qui retourne un objet `Process` instancié et lancé qui sera enregistré dans la propriété « Process » de la classe « WebradioServer ».

2 cas d'erreurs sont possibles :

- L'exécutable ShoutCAST serveur (sc\_serv.exe) n'est pas présent.
- Une instance précédente de ce processus (en cas de crash de l'application par exemple) est en tâche de fond et doit être fermée manuellement par l'utilisateur.

Pour plus d'informations concernant la gestion des différents processus, rendez-vous au chapitre [gestion des processus](#) 6.14.

---

#### 6.13.6 STATISTIQUES ET AUDITEURS

Les statistiques du serveur sont récupérées via l'API web du serveur ShoutCAST

([http://wiki.winamp.com/wiki/SHOUTcast\\_DNAS\\_Server\\_2\\_XML\\_Reponses#General\\_Server\\_Summary](http://wiki.winamp.com/wiki/SHOUTcast_DNAS_Server_2_XML_Reponses#General_Server_Summary)) et contiennent les informations suivantes :

- CurrentListeners : Le nombre d'auditeurs connectés (même avec doublons d'adresses IP)
- UniqueListeners : Le nombre d'auditeurs connectés (Adresses IP différentes)
- PeakListeners : Le record d'auditeurs connectés simultanément sur le serveur durant cette session d'exécution.
- AverageTime : Le temps moyen de connexion par auditeur.

La classe « WebradioServerStats » permet de stocker ces informations. La classe « WebradioServer » contient un champ « Stats » de ce type ainsi qu'une méthode « UpdateStats » qui permet de mettre à jour les données contenues dans le champ « Stats ».

Concernant les auditeurs, il est aussi possible, via l'API du serveur, de lister les auditeurs connectés et de récupérer les informations suivantes :

- Hostname : Adresse IP ou nom de domaine de l'auditeur
- Useragent : Décrit le type de lecteur audio utilisé pour écouter la webradio
- ConnectionTime : Le temps de connexion en secondes.
- Uid : Un identifiant unique pour l'auditeur, distribué automatiquement par le serveur.

La classe « WebradioListener » permet de stocker ces informations. Dans la classe « WebradioServer », la méthode « GetListeners » retourne une liste d'objet de type « WebradioListener » avec les informations récupérées.

Depuis la *vue*, le bouton situé dans l'onglet « status » et nommé « Update » permet de lancer cette mise à jour de statistiques et de la liste d'auditeurs. La *vue* appelle 2 méthodes (de son *controller*) différentes et effectue donc les 2 mises à jour séparément :

- UpdateListeners : La *vue* va recevoir la liste d'objet « WebradioListener » en provenance de la classe « WebradioServer » et les afficher dans le DataGridView.
- UpdateServerStats : Va appeler la méthode « UpdateStats » de la classe « WebradioServer » puis le *vue* sera mise à jour via « UpdateObservers » depuis le *model*. Dans sa méthode « UpdateView », il est prévu que les informations des statistiques du serveur soient affichées.

### 6.13.7 AFFICHAGE DES INTERFACES WEB

Les 2 boutons qui permettent l’affichage de l’interface web et l’administration web du serveur appellent le contrôleur qui lui appellera le *model*. Ce dernier utilise la classe `Process` et sa méthode statique « `Start` » pour lancer les URL dans le navigateur par défaut de l’utilisateur. L’adresse de chacune des interfaces est récupérable via la propriété « `WebInterfaceUrl` » et « `WebAdminUrl` » :

```
Process.Start(this.Webradios[webradioId].Server.WebInterfaceUrl);

//Dans la classe WebradioServer :
public string WebInterfaceUrl
{
    get
    {
        return "http://" + this.LocalIpAddress() + ":" + this.Port;
    }
}
```

`WebradioServer.cs`

## 6.14 GESTION DES PROCESSUS

Les processus lancés par le programme sont instancié à l'aide de la classe .NET « Process » :

Fournit l'accès à des processus locaux ainsi que distants, et vous permet de démarrer et d'arrêter des processus système locaux.

Source : MSDN Developer Network<sup>24</sup>

### 6.14.1 ISRUNNING

Chaque classe comportant une priorité de type Process (WebradioTranscoder et WebradioServer) dispose de 3 méthodes : Start, Stop et IsRunning. Voici le fonctionnement de la méthode :

```
foreach (Process prc in Process.GetProcesses())
{
    if (prc.ProcessName.Contains(this.Process.ProcessName))
    {
        result = true;
    }
}
if(this.Process.HasExited || !this.Process.Responding)
    result = false;
return result;
```

WebradioTranscoder.cs et WebradioServer.cs

En premier temps, il est vérifié si le processus est présent dans la liste des processus Windows. Si c'est le cas, un booléen est mis à vrai. Ensuite, un test vérifié si le processus a été fermé et ne répond plus. Si c'est le cas, cela veut dire que le processus présent dans la liste de processus Windows est un ancien processus qui ne correspond plus à celui présent dans la classe. Dans ce cas, le booléen est mis à faux.

### 6.14.2 DÉTECTION DES CRASHS/FERMETURES EXTERNES

Il est possible que les processus lancés par le logiciel soient fermés par une application externe ou qu'ils soient arrêtés par l'utilisateur en passant par le gestionnaire de tâches par exemple. L'application doit donc pouvoir détecter quand cela arrive afin de mettre à jour les informations affichées à l'écran pour l'utilisateur.

<sup>24</sup> [http://msdn.microsoft.com/fr-fr/library/system.diagnostics.process\(v=vs.110\).aspx](http://msdn.microsoft.com/fr-fr/library/system.diagnostics.process(v=vs.110).aspx)

Pour ce faire, le *model* dispose de 2 propriétés de type « List<> ». Une contient des WebradioServer et l'autre des WebradioTranscoder. Ces 2 listes vont être remplies par les objets (référence vers ces objets) dont le processus est censé être en fonctionnement. Ensuite, un timer (une minuterie à laquelle est attaché une méthode qui est appelée toutes les X millisecondes) va, toutes les secondes, vérifier l'état de chacun des processus présents dans les classes contenues dans les listes. Il exécute la méthode « IsRunning » sur chaque processus. Si il ne « run » plus, il est enlevé de la liste. Si au moins une liste a été mise à jour, la méthode UpdateObservers est appelée à la fin de la méthode.

---

#### 6.14.3 FERMETURE AUTOMATIQUE

Tous les processus en cours sont fermés à la fermeture volontaire de l'application par l'utilisateur. Il doit confirmer qu'il désire réellement fermer l'application.

## 7 TESTS

### 7.1 WEBRADIOS

N°	Description	Résultat	Commentaire
1	Affichage des webradios disponibles	OK	
2	Création d'une webradio	OK	
3	Suppression d'une webradio	OK	
4	Duplication d'une webradio	OK	Amélioration
5	Ouvrir une webradio	OK	
6	Ouverture de plusieurs webradios simultanément (différentes ou non)	OK	
7	Applications des modifications à toutes les webradios ouvertes	OK	Mise à jour des informations affichées (observateurs/sujet)
8	Changement du nom de la webradio	OK	Mise à jour dans la liste de webradios et l'administration de la webradio en question

### 7.2 STATUS

Corresponds à l'onglet « status » disponible dans chaque AdminView liée à une webradio.

N°	Description	Résultat	Commentaire
9	Affichage des statuts des transcodeurs de la webradio	OK	
10	Affichage des auditeurs + statistiques du serveur de la webradio	OK	
11	Mise à jour de l'affichage précédent	OK	Ne crash pas lorsque le serveur est arrêté
12	Affichage des musiques actuellement jouées sur chaque transcoder de la webradio	OK	

### 7.3 BIBLIOTHÈQUE

N°	Description	Résultat	Commentaire
13	Affichage des musiques/publicités dans les tableaux	OK	



14	Recherche dans les 2 tableaux	OK	Par n'importe quel critère
15	Importer des morceaux par dossier	OK	
16	Importer des morceaux par dossier de façon récursive	OK	
17	Importer des morceaux par fichiers directs	OK	
18	Ajouter des morceaux à une playlist	OK	Ajout multiple ou simple
19	Suppression de morceaux	OK	Multiple ou simple

## 7.4 LISTES DE LECTURE

N°	Description	Résultat	Commentaire
20	Création de playlists simple	OK	
21	Création de playlists générée	OK	Selon durée et genre (pour les playlists musicales)
22	Affichage des playlists disponibles	OK	Musicales ou publicitaires
23	Affichage du contenu d'une playlist	OK	
24	Affichage du temps total de la playlist	OK	
25	Recherche dans le contenu d'une playlist	OK	N'importe quel critère
26	Retirer morceaux de la playlist	OK	Sélection multiple ou simple
27	Suppression de playlist	OK	

## 7.5 GRILLE HORAIRE

N°	Description	Résultat	Commentaire
28	Affichage des événements du calendrier	OK	Dans le composant : rouge = publicités et bleu = musiques
29	Création d'événements	OK	Avec choix de playlist, etc.
30	Remplissage du formulaire de création automatique	OK	Remplissage en fonction de la sélection faite sur le composant Calendar (début, durée et

			jour)
<b>31</b>	Déplacement d'un événement	<b>OK</b>	Empêcher d'avoir 2 fois le même événement le même jour. Amélioration.

## 7.6 TRANSCODERS

N°	Description	Résultat	Commentaire
<b>32</b>	Création de transcodeurs	<b>OK</b>	Pas 2 fois le même nom par webradio
<b>33</b>	Affichage des transcodeurs disponibles	<b>OK</b>	
<b>34</b>	Affichage de la configuration d'un transcodeur	<b>OK</b>	
<b>35</b>	Modification de la configuration d'un transcodeur	<b>OK</b>	Nécessite le redémarrage du transcodeur
<b>36</b>	Suppression de transcodeurs	<b>OK</b>	
<b>37</b>	Démarrage/arrêt de transcodeur	<b>OK</b>	Mode debug ou non
<b>38</b>	Affichage du statut d'un transcodeur	<b>OK</b>	Allumé ou arrêté
<b>39</b>	Passage au morceau suivant sur le transcodeur	<b>OK</b>	Amélioration
<b>40</b>	Génération et affichage de l'historique	<b>OK</b>	Format PDF
<b>41</b>	Effacement de l'historique	<b>OK</b>	Dans la base de données
<b>42</b>	Affichage du fichier de log	<b>OK</b>	Éditeur par défaut
<b>43</b>	Capture live – Listing des périphériques audio	<b>~OK</b>	Périphérique bien listé, mais pas sûr que la bonne information soit récupérée (nom). Amélioration
<b>44</b>	Capture live - Lancement	<b>OK</b>	Informations bien configurées dans le transcodeur. Amélioration
<b>45</b>	Capture live - fonctionnelle	<b>KO</b>	Le transcodeur n'arrive pas à trouver le périphérique audio qui lui est configuré. Amélioration

## 7.7 SERVEUR DE DIFFUSION

N°	Description	Résultat	Commentaire
46	Affichage des paramètres du serveur	OK	
47	Modification des paramètres du serveur	OK	
48	Démarrage/arrêt du serveur	OK	Mode debug ou non
49	Affichage du statut du serveur	OK	
50	Affichage des interfaces web via bouton	OK	
51	Affichage du fichier de log	OK	

## 7.8 PROCESSUS

N°	Description	Résultat	Commentaire
52	Détection des processus crashés ou arrêtés extérieurement à l'application	OK	
53	Fermeture des processus liés à une webradio lors de la fermeture de sa dernière fenêtre d'administration ouverte	OK	
54	Fermeture de tous les processus à la fermeture de l'application	OK	
55	Détection des processus en cours	OK	IsRunning

## 7.9 AUTRES

N°	Description	Résultat	Commentaire
56	Régénération de toutes les configurations via le menu	OK	
57	Vérification de l'intégrité de la bibliothèque via le menu	OK	

## 8 CONCLUSION

### 8.1 BILAN PERSONNEL

Le projet WebradioManager consistait à concevoir un gestionnaire de webradios et cet objectif a été atteint. Le cahier des charges défini avec l'entreprise KTFM a été rempli. Le projet ayant avancé rapidement durant la durée du travail, des améliorations ont pu être discutées et ajoutées :

- La modification des informations de la bibliothèque (avec tag des fichiers musicaux)
- Ajout de la capture live (expérimentale)
- Duplication (clonage) d'une webradio
- Affichage des auditeurs et musiques en cours

Concernant les problèmes rencontrés, ils furent surtout présents dans la partie « gestion de processus ». C'est-à-dire, faire en sorte que mon application garde un plein contrôle sur les processus qu'elle lance (transcodeurs et serveurs). J'ai dû mettre en place un moyen de « surveiller » le processus afin de toujours savoir leur état et en informer l'utilisateur. J'ai aussi eu quelques soucis à utiliser l'API Ajax des transcodeurs, mais après quelques recherches, j'ai pu maîtriser cet outil. Le dernier problème concerne l'ajout de l'amélioration concernant la capture en live qui ne fonctionne pas parfaitement à cause d'un mauvais listing des périphériques son.

Pour le reste de l'application, les fonctionnalités se sont facilement ajoutées. J'avais la chance d'avoir un cahier des charges bien défini et d'avoir le fonctionnement de l'application au clair dans ma tête. Celle m'a permis de faire une analyse rapide et de pouvoir faire fonctionner toutes les fonctionnalités en « harmonie ». Au final, le projet a pu avancer rapidement et des améliorations ont pu être envisagées.

Je suis satisfait de mon travail, car il est fonctionnel et il m'a permis de réaliser un outil dont j'ai toujours rêvé afin de créer une webradio facilement. C'était une bonne expérience de réaliser un projet en collaboration avec une vraie entreprise possédant elle-même une webradio. Nous avons pu échanger et discuter. En plus de cela, c'est un univers qui m'intéresse beaucoup et j'ai appris de nombreuses choses.

J'ai pu appliquer mes connaissances en C# dans le cadre d'un projet important ainsi que de les consolider pour la suite de mon avenir professionnel. J'ai remarqué qu'avec les temps, mes habitudes de codage s'améliorent de mieux en mieux et cela me permet de faire moins d'erreurs et par conséquent, d'aller plus vite lors du développement.

Pour conclure, ce fut un superbe projet que j'ai pris du plaisir à réaliser dans le cadre de mon travail diplôme. Je compte poursuivre son développement afin d'implémenter les améliorations possibles au projet (elles sont nombreuses).

## 8.2 AMÉLIORATIONS POSSIBLES

- Multiserveur de diffusion par transcodeur
- Évènements périodiques (+ top horaire)
- DJ
- Gestion de semaines « type » (gestion multi semaine)
- Ajouter les exécutables ShoutCAST dans les ressources du logiciel.
- Sérialisation
- Génération de playlist : durée maximum et minimum des morceaux choisis
- Multiplateformes

## 9 REMERCIEMENTS

Je remercie mon professeur Francisco Garcia pour m'avoir suivi et aidé pendant tout mon travail. Je le remercie aussi de m'avoir mis en contact avec l'entreprise KTFM que je remercie également pour avoir pris le temps de prendre part à mon projet.

Je remercie également toute ma classe pour leur soutien et la bonne ambiance durant le travail. Je suis avec la plupart d'entre eux depuis ma première année de CFC.

## 10 APPORTS PERSONNELS

Élément	Apport (%)	Commentaire
<b>Patron MVC et observateurs/sujet</b>	100	
<b>Interaction directe avec la base de données</b>	10	Classe BddControls trouvée sur un tutoriel. J'y ai apporté quelques modifications.
<b>Interfaces</b>	100	
<b>CRUD<sup>25</sup> des webradios</b>	100	
<b>Renommer webradio</b>	100	
<b>Duplication webradio</b>	100	
<b>CRUD des morceaux musicaux et publicitaires</b>	100	
<b>CRUD des playlists</b>	100	
<b>CRUD des événements dans le calendrier</b>	100	

<sup>25</sup> CRUD (pour Create, Read, Update, Delete) désigne les quatre opérations de base pour la persistance des données, en particulier le stockage d'informations en base de données.

<b>Gestion visuel d'un calendrier hebdomadaire</b>	0	Composant trouvé sur internet
<b>CRUD des transcodeurs</b>	100	
<b>Résolution d'adresse IP</b>	0	Code trouvé sur un tutoriel
<b>CRUD serveurs de diffusion</b>	100	
<b>Affichage des différentes informations dans la vue</b>	100	
<b>Génération des configurations de chaque élément</b>	100	
<b>Gestion des divers fichiers sur le disque</b>	90	Sauf différents fichiers temporaires créés par le transcodeur et le serveur
<b>Transcodeur</b>	0	Exécutable transcodeur fournit par ShoutCAST
<b>Serveur de diffusion</b>	0	Exécutable serveur de diffusion fournit par ShoutCAST
<b>Gestion des processus</b>	100	Avec l'aide de la classe .NET « Process ». Regroupe le démarrage/arrêt/surveillance des processus.
<b>Gestion de l'historique</b>	80	Utilisation de l'API Ajax du transcoder pour récupérer des informations
<b>Capture live</b>	25	Capture live gérée par le transcodeur, mais configuration par moi-même

## 11 GLOSSAIRE

- Playlist : Liste de lecture contenant plusieurs fichiers musicaux.
- Stream : Flux
- Log : Journal des événements
- Item : Objet/élément
- Live : En direct
- Instanciation : (Anglicisme) Action d'instancier, d'initialiser en programmation, à partir d'un espace mémoire réservé, un objet à partir d'un ensemble de caractéristiques, appelé «classe». Les objets sont obtenus à partir d'une instanciation de classe.
- Object : Classe qui est à la base de toutes les classes .NET.
- SQL-92 : La 3e révision du langage SQL.
- ACID : les propriétés ACID (atomicité, cohérence, isolation et durabilité) sont un ensemble de propriétés qui garantissent qu'une transaction informatique est exécutée de façon fiable.
- Ajax : (acronyme d'Asynchronous JavaScript and XML) permet de construire des applications Web et des sites web dynamiques interactifs sur le poste client en se servant de différentes technologies ajoutées aux navigateurs web entre 1995 et 2005.

## 12 ILLUSTRATIONS

Figure 1 - Mindmap (discussion avec KTF .....	12
Figure 2 - Schéma application .....	13
Figure 3 - Interface principale AdminView .....	14
Figure 4 - Interface gestion des webradios SelectionView .....	15
Figure 5 - Schéma webradios .....	16
Figure 6 - Onglet "status" .....	18
Figure 7 - Bibliothèque de musiques .....	19
Figure 8 - Onglet "playlists" .....	21
Figure 9 - Onglet "timetable" .....	23
Figure 10 - Calendrier .....	23
Figure 11 - Sélection multiple calendrier .....	24
Figure 12 - Définition d'un événement avec des éléments.....	25
Figure 13 - Onglet "Transcoders" .....	26
Figure 14 - Onglet "server" .....	28
Figure 15 - Interface web ShoutCAST serveur.....	29

Figure 16 - Administration web ShoutCAST serveur .....	30
Figure 17 - Diagramme <i>model</i> .....	31
Figure 18 - MVC .....	32
Figure 19 - Pattern observateurs/sujet .....	33
Figure 20 - Observateurs/sujet dans WebradioManager .....	34
Figure 21 - AudioType et StreamType enum .....	34
Figure 22 - Logo SQLite .....	36
Figure 23 - Principe de base de diffusion .....	45
Figure 24 - Schéma de diffusion .....	45
Figure 25 - Schéma de diffusion 2 .....	46
Figure 26 - Logo Schoutcast .....	46
Figure 27 - Schéma shoutcast .....	48
Figure 28 - Schéma structure des fichiers/dossiers .....	49
Figure 29 - Exemple lancement transcoder .....	50
Figure 30 - Diagramme de séquence initialisation application .....	52
Figure 31 - Classe "Webradio" .....	53
Figure 32 - Schéma création webradio .....	54
Figure 33 - Diagramme de séquence instanciation AdminView .....	55
Figure 34 - Schema changement nom webradio .....	58
Figure 35 - Classes bibliothèque .....	60
Figure 36 - Schéma importation fichier .....	62
Figure 37 - Classes Playlist .....	67
Figure 38 - Algorithme génération playlist .....	70
Figure 39 - Classes timetable .....	73
Figure 40 - Day View Calendar .....	74
Figure 41 - Exemple XML calendrier .....	76
Figure 42 - Modification d'un événement .....	80



Figure 43 - Classes transcodeurs .....	82
Figure 44 - ShoutCAST transcoder console .....	83
Figure 45 - Schéma création d'un transcoder .....	87
Figure 46 - Schéma gestion de l'historique .....	93
Figure 47 - Classe WebradioServer .....	95
Figure 48 - ShoutCAST serveur .....	96

## 13 RÉFÉRENCES

- <https://cacoo.com> : Création de diagrammes en ligne
- <http://balsamiq.com/> : Création de « mokup »
- <http://calendar.codeplex.com/> : Composant C# pour l’affichage du calendrier sur une semaine
- <http://sqlitestudio.pl/> : Logiciel de gestion de base de données SQLite
- <https://github.com/itext/itextsharp> : Bibliothèque .NET pour la génération de document (PDF, etc.)
- <http://www.shoutcast2.com/> : Logiciel de diffusion pour webradio

## 14 ANNEXES

- Plannings
- Documentation et listing du code
- Poster format A4