

CFPT-I

# WebradioManager

---

Travail de diplôme 2014

**Simon Menetrey – T.IN E2A**

**16/04/2014**

# 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 multiple 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 créée. 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 serveur 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 exécutables 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.

## Contenu

1	Résumé .....	1
1.1	Français.....	1
1.2	English.....	1
2	Introduction.....	7
2.1	Mise en situation .....	7
2.2	Sujet.....	7
2.3	Qu'est-ce qu'une webradio ? .....	7
2.4	Pourquoi ce sujet ? .....	7
2.5	Termes.....	8
3	Cahier des charges.....	9
4	Analyse fonctionnelle .....	10
4.1	Schéma de l'application .....	10
4.2	Interface principale .....	11
4.2.1	Fenêtre d'administration.....	11
4.2.2	Menu principale .....	11
4.3	Gestion de plusieurs webradio.....	12
4.3.1	Création d'une webradio.....	13
4.3.2	Sélection d'une webradio.....	13
4.3.3	Fermeture d'une webradio .....	13
4.4	Serveur de diffusion interne et externe .....	14
4.5	Onglet « Status » .....	14
4.6	Gestion des musiques/pubs .....	15
4.6.1	Affichage.....	15
4.6.2	Importer et indexer .....	15
4.6.3	Ajout à une playlist.....	16
4.6.4	Supprimer .....	16
4.7	Gestion des listes de lecture .....	17
4.7.1	Listes de lecture musicales .....	17
4.7.2	Listes de lecture publicitaires .....	17
4.7.3	Création .....	17
4.7.4	Génération automatique.....	17
4.7.5	Affichage du contenu d'une playlist.....	18
4.7.6	Retirer d'une playlist .....	18

4.7.7	Recherche .....	18
4.8	Gestion des horaires.....	18
4.8.1	Événement.....	19
4.8.2	Événement périodique .....	19
4.8.3	Remplissage manuel.....	19
4.8.4	Remplissage automatique.....	<b>Erreur ! Signet non défini.</b>
4.8.5	Modification d'un événement.....	20
4.8.6	Suppression d'un événement.....	20
4.9	Gestion des transcoders.....	21
4.9.1	Création .....	21
4.9.2	Affichage.....	22
4.9.3	Modification .....	22
4.9.4	Contrôles .....	22
4.9.5	Historique de diffusion .....	22
4.10	Gestion du serveur .....	23
4.10.1	Contrôles .....	23
4.10.2	Log .....	23
4.10.3	Configuration.....	23
4.10.4	Interface web.....	24
4.10.5	Administration web .....	24
5	Analyse organique .....	26
5.1	Environnement.....	26
5.2	Diagramme de classes .....	26
5.2.1	Diagramme .....	26
5.2.2	Observateurs/Sujet .....	26
5.2.3	AudioType et StreamType .....	27
5.2.4	Classes abstraites .....	27
5.2.5	Stockage des webradios .....	27
5.3	Base de données.....	27
5.3.1	SQLite.....	27
5.3.2	Utilisation .....	28
5.3.3	Suppression en cascade.....	28
5.3.4	Schéma .....	29
5.3.5	Twebradio.....	30

5.3.6	Tsever .....	30
5.3.7	Tcalendar .....	30
5.3.8	Tcalendarevent.....	30
5.3.9	Tplaylist.....	31
5.3.10	Taudiotype.....	31
5.3.11	Tmusic.....	31
5.3.12	Tgender.....	31
5.3.13	Tplaylist_has_music.....	31
5.3.14	Thistory.....	32
5.3.15	Ttranscoder .....	32
5.4	Schéma de diffusion .....	33
5.4.1	Principe de base .....	33
5.4.2	Infomaniak.....	33
5.5	ShoutCast.....	34
5.5.1	Présentation .....	34
5.5.2	Pourquoi cet outil ? .....	34
5.5.3	Serveur .....	35
5.5.4	Transcoder.....	35
5.5.5	Schéma de fonctionnement résumé .....	35
5.6	Structures des dossiers/fichiers .....	36
5.6.1	Schéma .....	36
5.6.2	Exécutables Shoutcast.....	37
5.7	Initialisation de l'application .....	38
5.8	Gestion des processus .....	<b>Erreur ! Signet non défini.</b>
5.9	Webradio .....	39
5.9.1	Classes associée.....	39
5.9.2	Affichage des webradios disponibles .....	39
5.9.3	Création .....	40
5.9.4	Chargement .....	40
5.9.5	Duplication .....	42
5.9.6	Suppression .....	42
5.9.7	Génération des configurations .....	42
5.10	Bibliothèque .....	43
5.10.1	Classes utilisées .....	43

5.10.2	MP3 .....	43
5.10.3	Importation .....	43
5.10.4	Tags ID3 .....	44
5.10.5	Analyse des tags .....	45
5.10.6	Indexation.....	45
5.10.7	Suppression .....	45
5.10.8	Vérification des données.....	46
5.10.9	Recherche .....	46
5.11	Listes de lecture.....	47
5.11.1	Classes utilisées .....	47
5.11.2	Génération de configuration .....	47
5.11.3	Création d'une playlist.....	48
5.11.4	Génération automatique.....	48
5.11.5	Ajout à une playlist .....	50
5.11.6	Retirer d'une playlist .....	50
5.11.7	Affichage du contenu .....	50
5.11.8	Suppression d'une playlist.....	51
5.12	Grille horaire.....	52
5.12.1	Outil utilisé .....	52
5.12.2	Gestion du calendrier par ShoutCAST .....	53
5.12.3	Classes utilisées .....	<b>Erreur ! Signet non défini.</b>
5.12.4	Génération de configuration .....	54
5.12.5	Affichage du calendrier .....	54
5.12.6	Sélection depuis le calendrier .....	56
5.12.7	Création d'un événement.....	56
5.12.8	Modification d'un événement.....	57
5.12.9	Suppression d'un événement.....	58
5.12.10	Génération automatique.....	<b>Erreur ! Signet non défini.</b>
5.13	Transcoder.....	59
5.13.1	Classes utilisées .....	59
5.13.2	Définition des bitrates, taux d'échantillonnage et type d'encoder .....	59
5.13.3	Fichier de configuration et log.....	61
5.13.4	Licence MP3.....	63
5.13.5	Création d'un transcoder .....	64

5.13.6	Affichage d'un transcoder .....	65
5.13.7	Modification d'un transcoder.....	65
5.13.8	Suppression d'un transcoder.....	65
5.13.9	Exécution et fermeture .....	65
5.13.10	Gestion des processus.....	<b>Erreur ! Signet non défini.</b>
5.13.11	Statut et logs.....	<b>Erreur ! Signet non défini.</b>
5.13.12	Administration web (weblet) .....	66
5.13.13	Historique .....	66
5.14	Serveur de diffusion interne.....	70
5.14.1	Classe utilisée .....	70
5.14.2	Outil utilisé .....	70
5.14.3	Configuration.....	71
5.14.4	Mise à jour de la configuration .....	72
5.14.5	Exécution et fermeture .....	72
5.14.6	Affichage des interfaces web .....	73
5.14.7	Log .....	<b>Erreur ! Signet non défini.</b>
6	Gestion des processus.....	74
7	Tests.....	75
8	Plannings .....	76
8.1	Prévu.....	76
8.2	Final .....	76
9	Apports personnels .....	77
10	Conclusion .....	78
11	Améliorations possibles.....	79
12	Références.....	79
13	Annexes .....	79

## 2 Introduction

### 2.1 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 de pouvoir avoir une traçabilité des morceaux qu'elle diffuse avec des informations précises afin de pouvoir faciliter le paiement des droits d'auteur à la Suisa<sup>1</sup>. J'ai donc réalisé le cahier des charges avec KTFM afin de répondre au mieux à leurs besoins.



### 2.2 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

### 2.3 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.

### 2.4 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.

---

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



## 2.5 Termes

Voici une liste de termes qui seront utilisés dans cette documentation :

- Playlist : Liste de lecture
- Stream : Flux
- Log : Journal des événements

### 3 Cahier des charges

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

- Possibilité de gérer plusieurs webradio indépendamment
- Gestion des playlist, horaires et pubs
  - Génération automatique de playlist
  - Génération au format xml
- Gestion des serveurs de diffusions distants et transcoder 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 et le *mindmap*.

# Analyses

---

## 4 Analyse fonctionnelle

### 4.1 Schéma de l'application

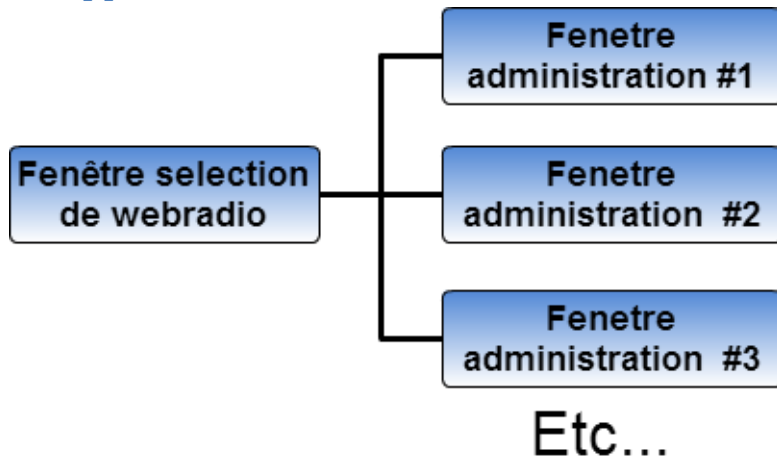


Figure 1 - Schéma application

Lors du lancement de l'application, la fenêtre de sélection de webradio se lance. 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.

## 4.2 Interface principale

### 4.2.1 Fenêtre d'administration



Figure 2 - 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, il est possible d'avoir plusieurs fenêtres d'administration ouvertes simultanément, une pour chaque webradio lancée. Les différents onglets sont décrits dans les chapitre suivants.

### 4.2.2 Menu principale

- File
  - Generate all configs = Génère la configuration de chaque élément de la webradio
- About : Affiche les informations sur le logiciel et son auteur

### 4.3 Gestion de plusieurs webradio

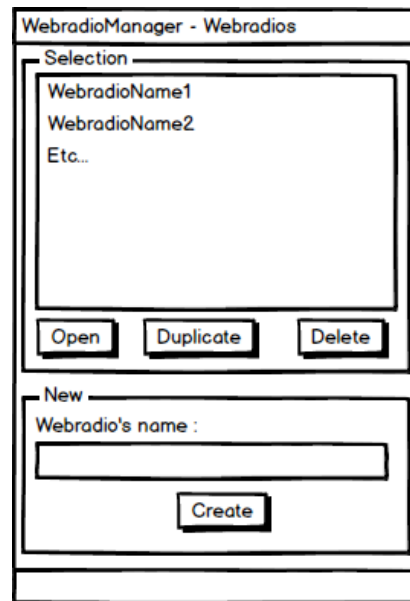


Figure 3 - Interface gestion des webradios SelectionView

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 cliquer pour ouvrir une webradio.

Le bouton « Duplicate » crée une copie exacte de la webradio sélectionnée avec un suffixe « copy » 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 webradio en même temps qu'il le souhaite, en effet, à tous moments, 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 transcoders. Un transcoder 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](#).

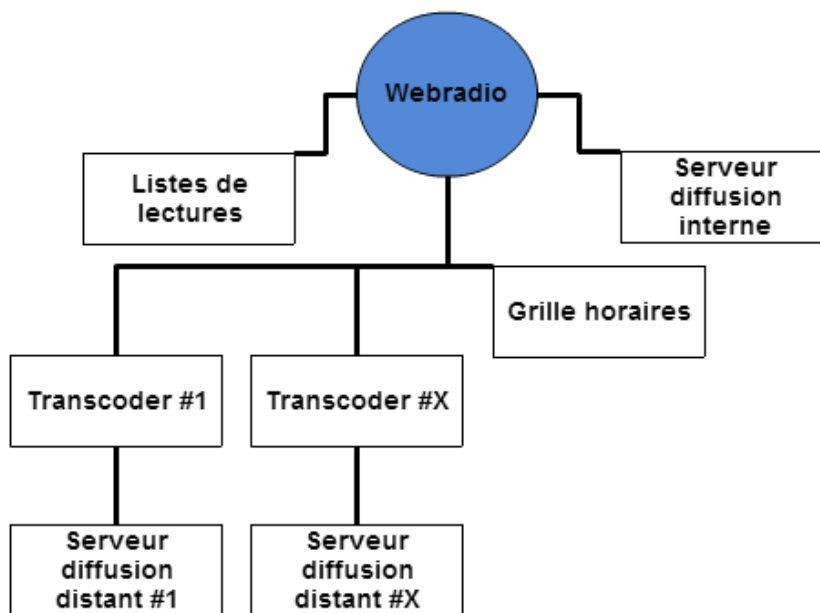
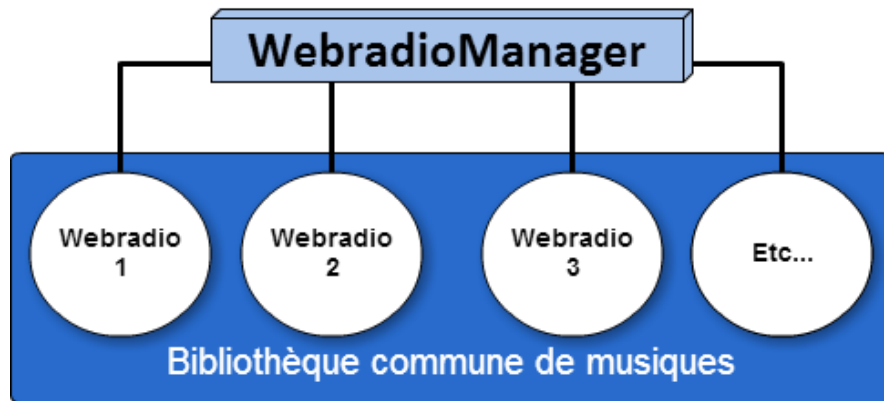


Figure 4 - Schéma webradios

#### 4.3.1 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 créé. Le nom d'une webradio ne peut pas dépasser 255 caractères.

#### 4.3.2 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.

#### 4.3.3 Fermeture d'une webradio

Lors de la fermeture d'une fenêtre d'administration de webradio, l'utilisateur doit confirmer si il est sur de vouloir fermer la fenêtre en sachant que cela arrêtera le serveur de diffusion et les transcoders.

#### 4.4 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>2</sup> (exemple : infomaniak<sup>3</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.

#### 4.5 Onglet « Status »

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

Transcoder	Status
Transcoder1	Off
Transcoder2	On

Figure 5 - 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. Un tableau affiche l'état des différents transcoders de la webradio actuelle.

TODO : expliquer de « current tracks »

TODO : expliquer la modification de non : fermes tous les processus de la webradio

TODO : expliquer les stats serveur + kick

<sup>2</sup> Fournisseur de services internet

<sup>3</sup> <http://www.infomaniak.com/>

## 4.6 Gestion des musiques/pubs

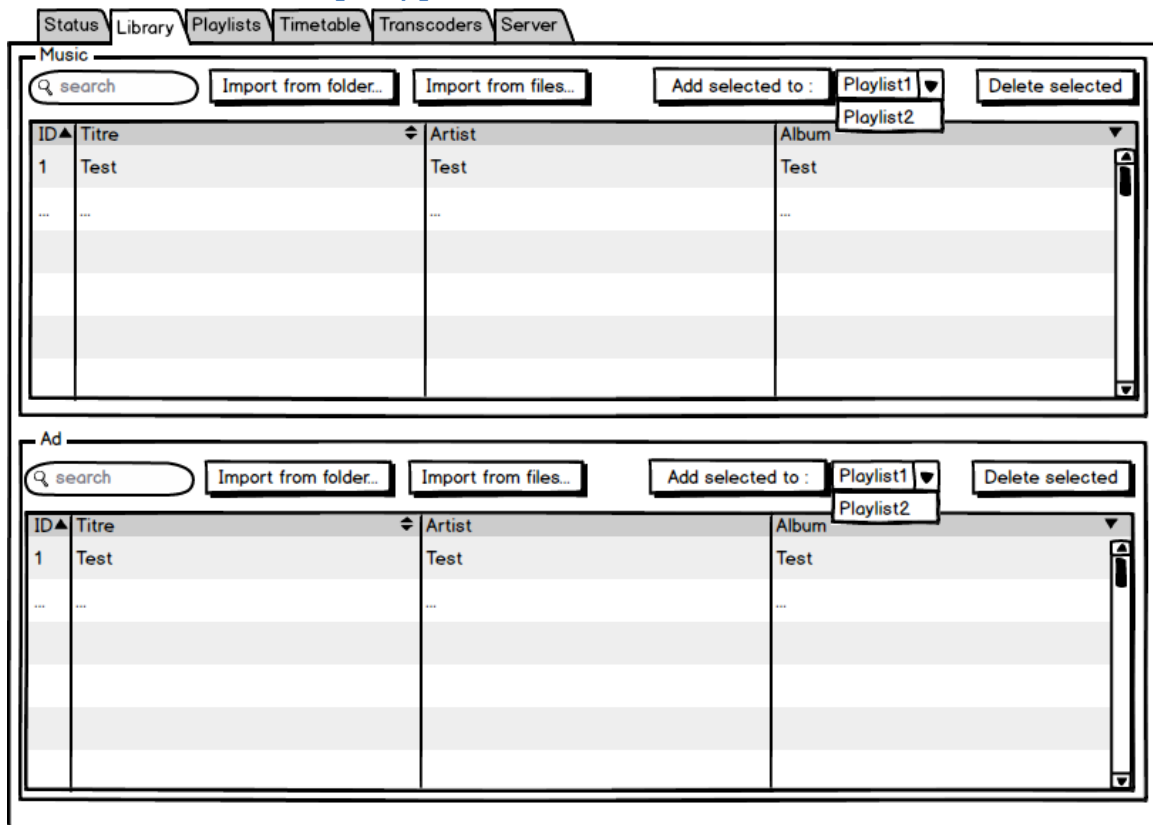


Figure 6 - Bibliothèque de musiques

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.

### 4.6.1 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'une liste à entrées. Il est possible de sélectionner un ou plusieurs éléments.

### 4.6.2 Importer et indexer

L'utilisateur peut importer des fichiers musicaux dans sa bibliothèque. 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. A 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.



#### 4.6.3 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.

#### 4.6.4 Supprimer

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

## 4.7 Gestion des listes de lecture

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

Figure 7 - Onglet "playlists"

### 4.7.1 Listes de lecture musicales

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

### 4.7.2 Listes de lecture publicitaires

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

### 4.7.3 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.

### 4.7.4 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.

#### 4.7.5 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 telle que le temps total de lecture, le nombre de morceaux présents etc.

#### 4.7.6 Retirer d'une playlist

L'utilisateur peut sélectionner un ou plusieurs morceaux dans la partie affichant le contenant 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.

#### 4.7.7 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.

### 4.8 Gestion des horaires

The screenshot shows a software interface with a top navigation bar containing tabs: Status, Library, Playlists, Timetable (selected), Transcoders, and Server. Below the tabs is a 'Create event' section. This section includes a 'Name' field, a 'Playlist' dropdown menu (currently showing 'Playlist1'), a 'Start time' field, a 'Duration' field, and a 'Priority' field (set to 3). There are checkboxes for days of the week: MO (checked), TU, WE, TH, FR, SA, and SU. There is also a 'Shuffle' checkbox. A 'Create' button is located to the right of the 'Priority' field. The main area of the interface is labeled 'Timetable' and contains a large yellow box with the text 'Composant calendrier semaine'.

Figure 8 - Onglet "timetable"

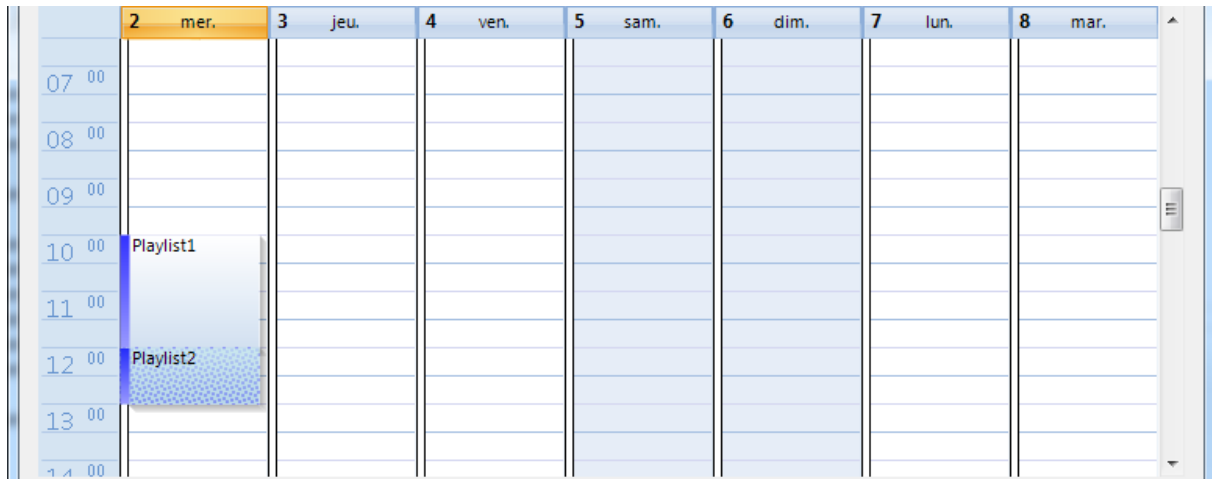


Figure 9 - Calendrier

#### 4.8.1 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.

#### 4.8.2 Événement périodique

Un événement périodique se produit à intervalles réguliers.

TODO

#### 4.8.3 Remplissage manuel

L'utilisateur peut utiliser le formulaire de création d'événement 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.

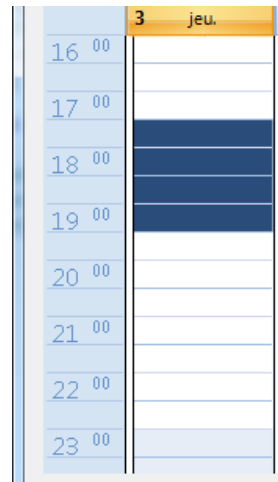


Figure 10 - Sélection multiple calendrier

La partie bleu foncée est la sélection faite par l'utilisateur. Dans ce cas, l'heure de début sera 17h30 et sa durée sera 2h car la sélection finie à 19h30. Ces informations sont automatiquement affichées dans le formulaire de création. Evidemment, 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 lira la liste de lecture de façon aléatoire. « Priority » définit le niveau de priorité de l'événement par rapport à un autre qui sera prévu au même moment.

#### 4.8.4 Modification d'un événement

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 transcoders pour que la nouvelle configuration soit prise en compte.

#### 4.8.5 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.

## 4.9 Gestion des transcoders

Figure 11 - Onglet "Transcoders"

Cet onglet permet de gérer les différents transcoders 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.

### 4.9.1 Création

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

- Le type de flux (mp3 ou ACC+)
- Le bitrate<sup>4</sup> du flux
- Le sample rate<sup>5</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
- Adresse IP du serveur (ou le nom dns via bouton « resolve »)
- Port du serveur
- Mot de passe du serveur

<sup>4</sup> Débit binaire : une mesure de la quantité de données numériques transmises par unité de temps.

<sup>5</sup> Taux d'échantillonnage

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.

#### 4.9.2 Affichage

Dans la partie droite, la liste des transcoders 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 transcoder 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 ».

#### 4.9.3 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.

#### 4.9.4 Contrôles

La partie « status » permet de contrôler le transcoder 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 transcoder sélectionné.

#### 4.9.5 Capture live

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

TODO : expliquer quand l'interface sera finie

#### 4.9.6 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 ».

## 4.10 Gestion du serveur

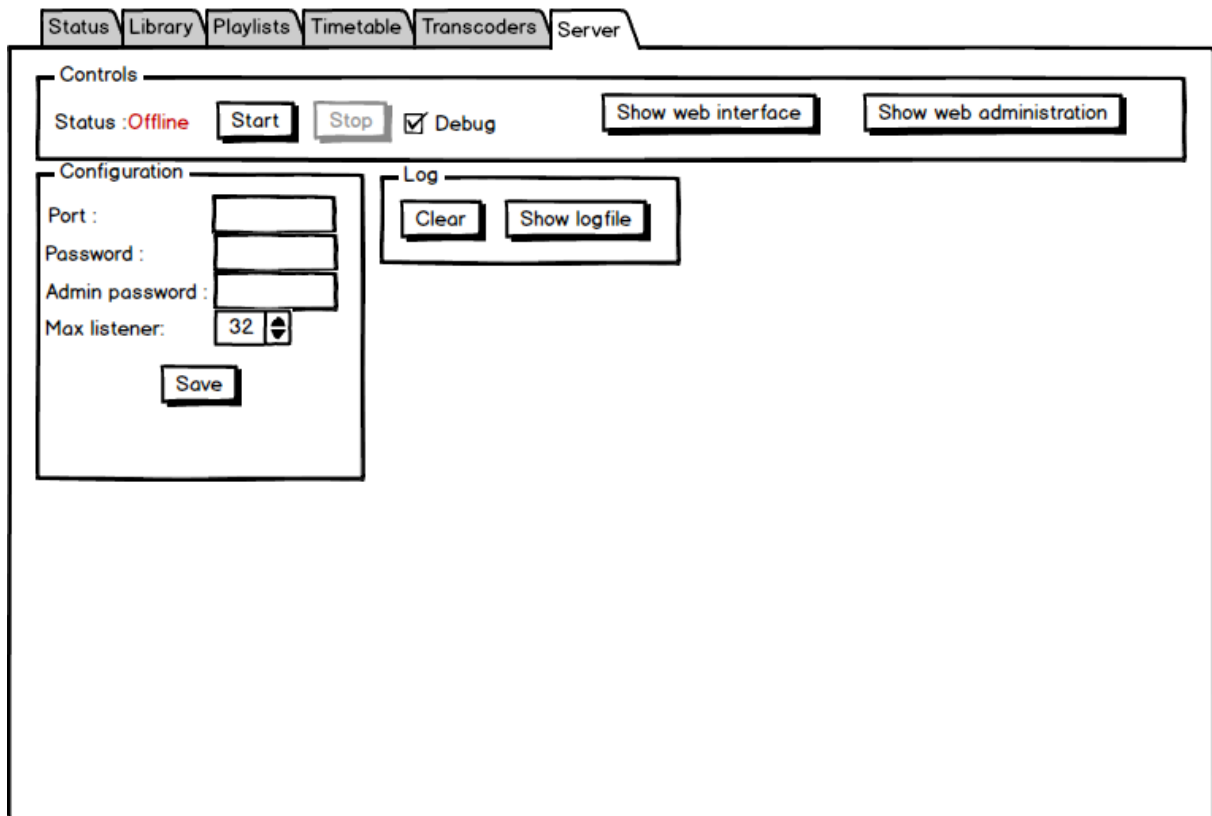


Figure 12 - Onglet "server"

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 diffusion le flux audio.

### 4.10.1 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. A 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.

### 4.10.2 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.

### 4.10.3 Configuration

Différents éléments sont configurables pour le serveur :

- 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 connexion au serveur (connexion client/auditeur). Par défaut : 1. Maximum : 2000.

Attention : Les 2 mots de passe doivent être différents !

#### 4.10.4 Interface web

Une interface web est fournie avec le serveur ShoutCAST. Elle 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.

The screenshot shows the 'SHOUTcast Stream Status' page. At the top, it says 'SHOUTcast Server v2.2.1.109/win64'. Below this is a navigation bar with links: Status, Song History, Listen (with a speaker icon), Stream URL, Admin Login, and Server Login. The main content area is titled 'Current Stream Information' and displays the following details:

Server Status:	Server is currently up and private
Stream Status:	Stream is up at 56 kbps with 0 of 32 listeners
Stream Name:	My Test Server
Content Type:	audio/aacp
Stream Genre:	Misc
Stream URL:	<a href="http://www.shoutcast.com">http://www.shoutcast.com</a>

Figure 13 - Interface web ShoutCAST serveur

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

#### 4.10.5 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.

The screenshot shows the 'SHOUTcast Listeners and Status' page. At the top, it says 'SHOUTcast Server v2.2.1.109/win64'. Below this is a navigation bar with links: Listeners, Log (Tailing | Save), Song History, Ban List, Reserved List, Admin Logout, and Server Login. The main content area is titled 'Current Stream Information' and displays the following details:

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

Figure 14 - 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.

## 5 Analyse organique

### 5.1 Environnement

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

### 5.2 Diagramme de classes

#### 5.2.1 Diagramme

TODO : PAS FINI

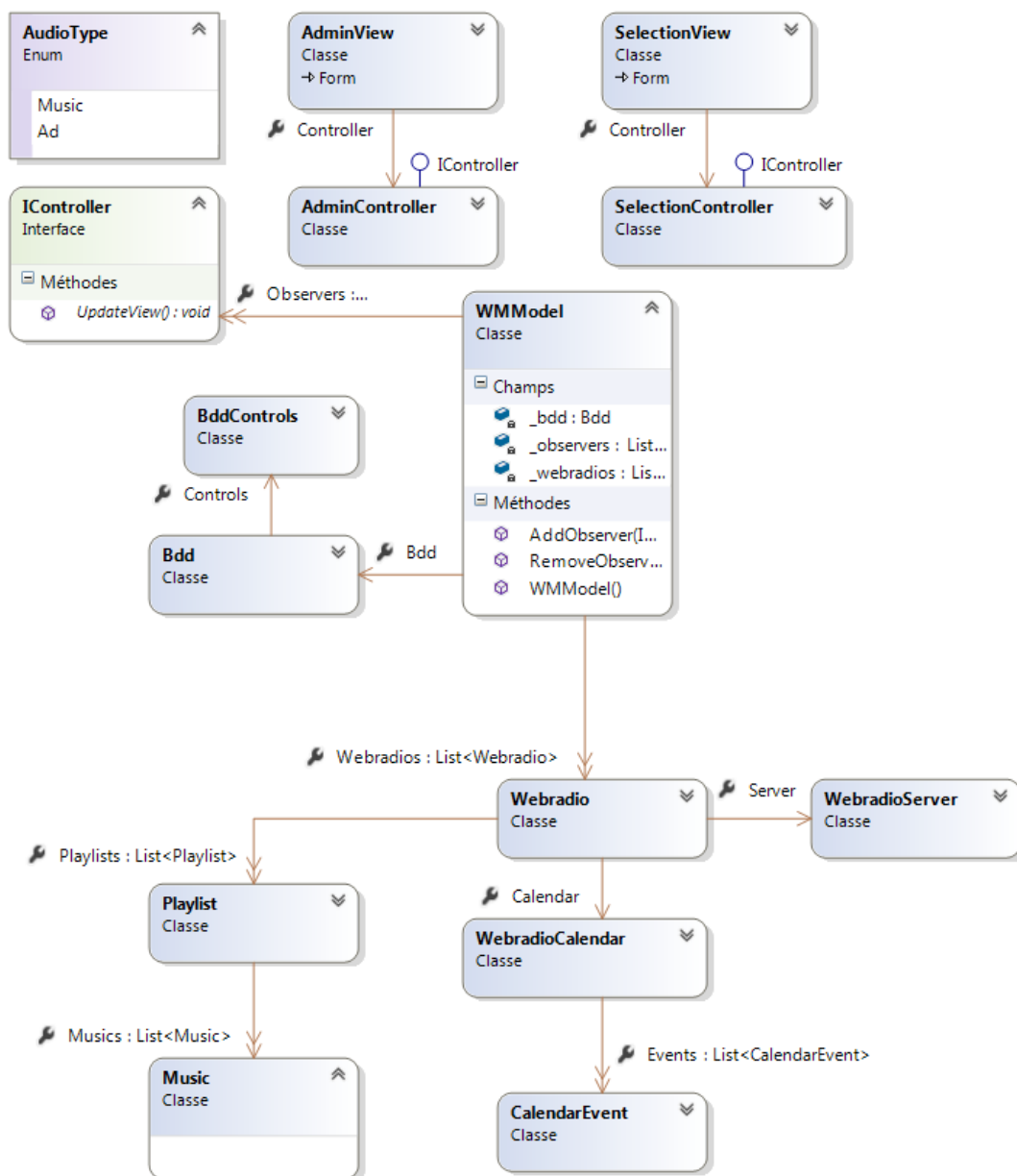


Figure 15 - Diagramme de classes

#### 5.2.2 Observateurs/Sujet

TODO : explication du principe

TODO : expliquer le paramètre « id » qui permet d'update que les fenetre concernée par la wberadio qui a changée

TODO : expliquer updateview

### 5.2.3 AudioType et StreamType

TODO : expliquer que ces enum ont des valeur prise des id de la base de données

### 5.2.4 Classes abstraites

TODO : explication du principe

TODO : explication de l'instanciation

### 5.2.5 Stockage des webradios

TODO : explication des webradio dans le model sous form de dictionnary avec id

## 5.3 Base de données

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

### 5.3.1 SQLite



Figure 16 - 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.

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/>

### 5.3.2 Utilisation

Une bibliothèque sous forme d'une DLL<sup>6</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 part les tables taudiotype et tstreamtype) de la base de données est stockée dans les ressources du logiciel afin d'avoir une BDD de base.

La classe Bdd utilise BddControls 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 modèle à chaque démarrage de l'application.

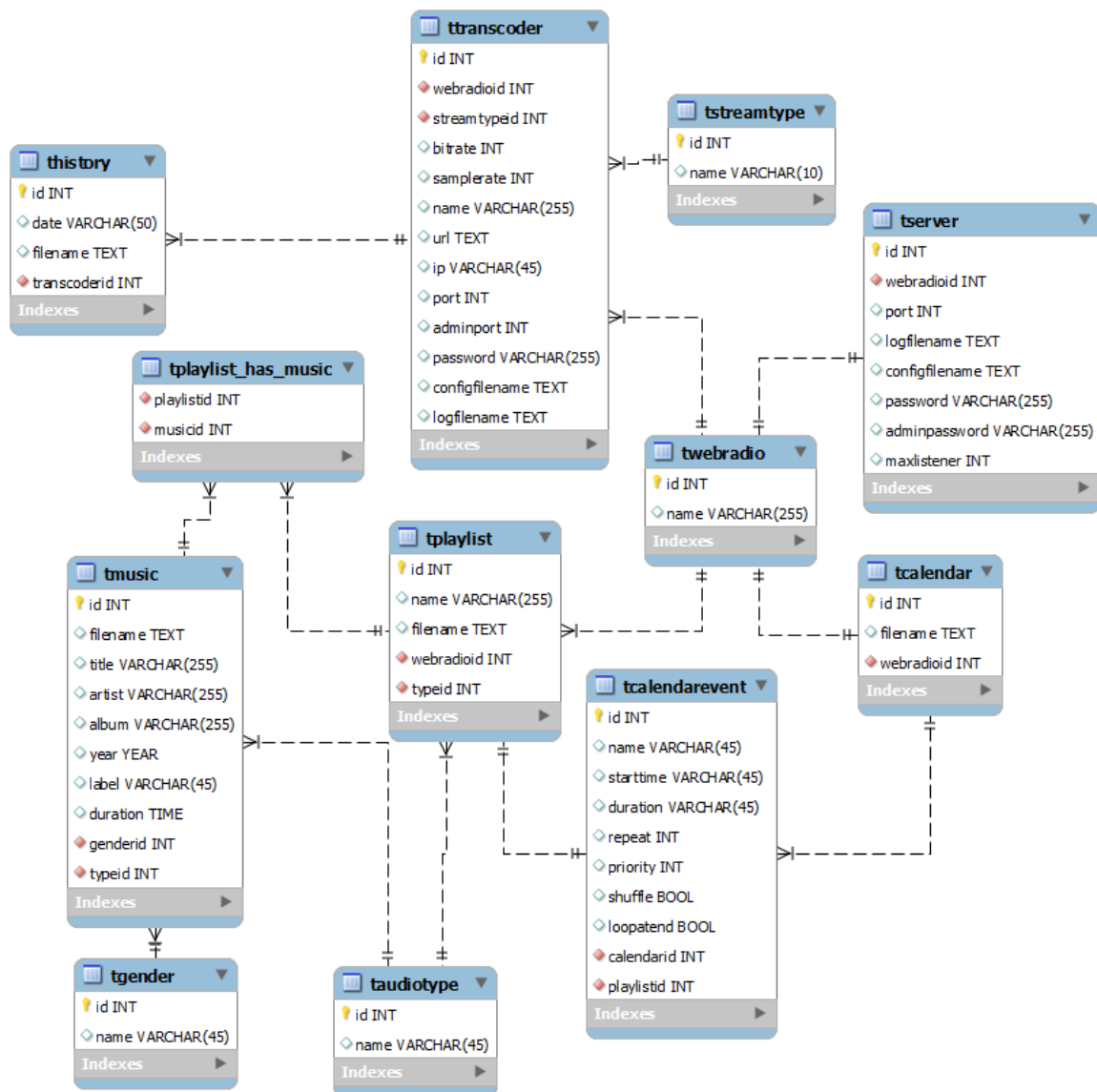
### 5.3.3 Suppression en cascade

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.

---

<sup>6</sup> Dynamic Link Library

## 5.3.4 Schéma



### 5.3.5 Twebradio

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

### 5.3.6 Tsever

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'auditeur sur le serveur

### 5.3.7 Tcalendar

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)

### 5.3.8 Tcalendarevent

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(45)	Heure du commencement de l'événement
<b>Duration</b>	Varchar(45)	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)

### 5.3.9 Tplaylist

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)

### 5.3.10 Taudiotype

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

Cette table est lié à l'enum « AudioType ».

### 5.3.11 Tmusic

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>	Time	Durée du morceau
<b>Genderid</b>	Int	Clé étrangère (genre du morceau)
<b>Typeid</b>	Int	Clé étrangère (type du morceau)

### 5.3.12 Tgender

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

### 5.3.13 Tplaylist\_has\_music

Nom	Type	Description
-----	------	-------------



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

#### 5.3.14 Thistory

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 (Transcoder ayant joué cette musique)

#### 5.3.15 Ttranscoder

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 transcoder 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 transcoder
<b>Logfilename</b>	Text	Chemin vers le fichier de log du transcoder

## 5.4 Schéma de diffusion

### 5.4.1 Principe de base

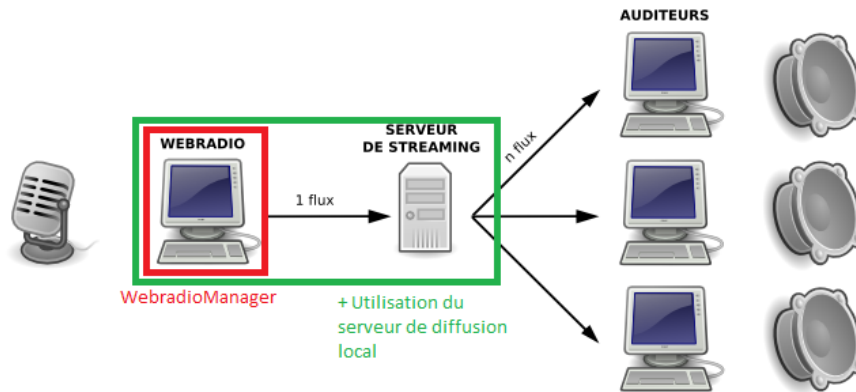


Figure 17 - Principe de base diffusion

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

### 5.4.2 Infomaniak

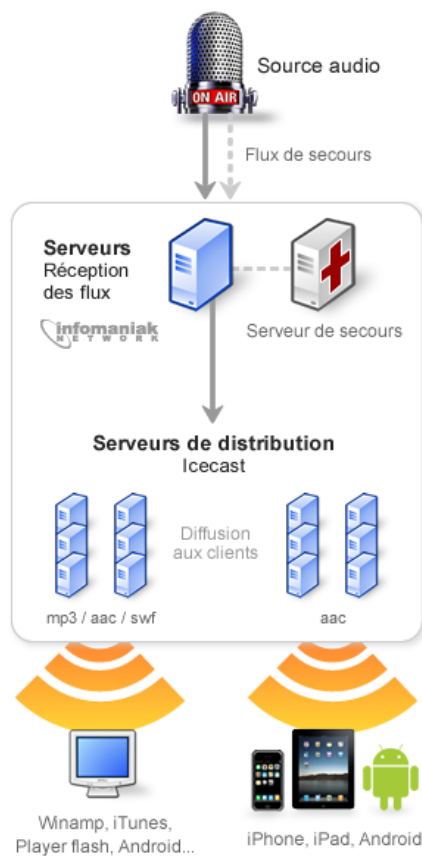


Schéma de fonctionnement  
du streaming radio live Infomaniak Network SA

Figure 18 - Schéma de diffusion

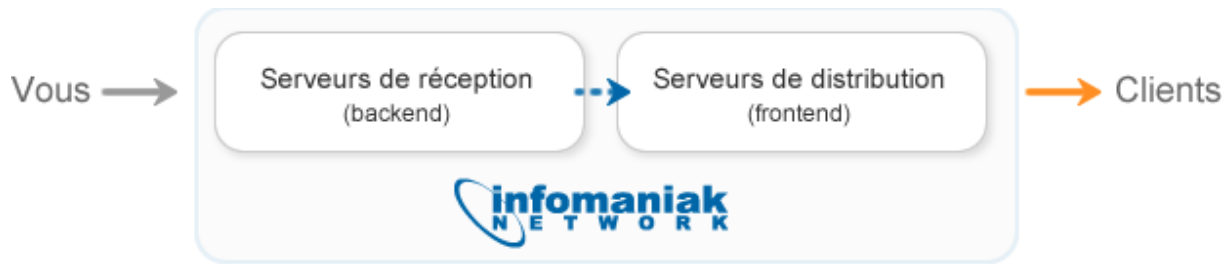


Figure 19 - Schéma de diffusion 2

Ces 2 schémas montrent le fonctionnement de la diffusion des webradio via infomaniak. Le logiciel avec les transcoders sont les « sources 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.

## 5.5 ShoutCast



Figure 20 - Logo Shoutcast

### 5.5.1 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>7</sup>. Ce sont donc ces derniers qui seront utilisés pour le projet.

### 5.5.2 Pourquoi cet outil ?

J'ai choisi Shoutcast car il propose des outils en ligne de commande ainsi qu'une gestion facilitée via des fichiers XML<sup>8</sup> ou texte basiques. Cela permet de pouvoir interagir plus facilement avec via des applications externes telle que la mienne.

<sup>7</sup> Lecteur multimédia développé par Nullsoft

<sup>8</sup> eXtensible Markup Language : Langage de balisage extensible

### 5.5.3 Serveur

Shoutcast, parmi ses outils, propose un serveur en ligne de commande qui sera utilisé dans ce projet. Il permet de diffuser un flux qu'il reçoit et qui est envoyé par, par exemple, un [transcoder](#). Ce flux est distribué aux clients (auditeurs dans ce cas) qui désire écouter la webradio.

Ce serveur propose aussi une interface web pour visualiser ton 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](#).

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

### 5.5.4 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 playlist et de les agencées 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.

### 5.5.5 Schéma de fonctionnement résumé

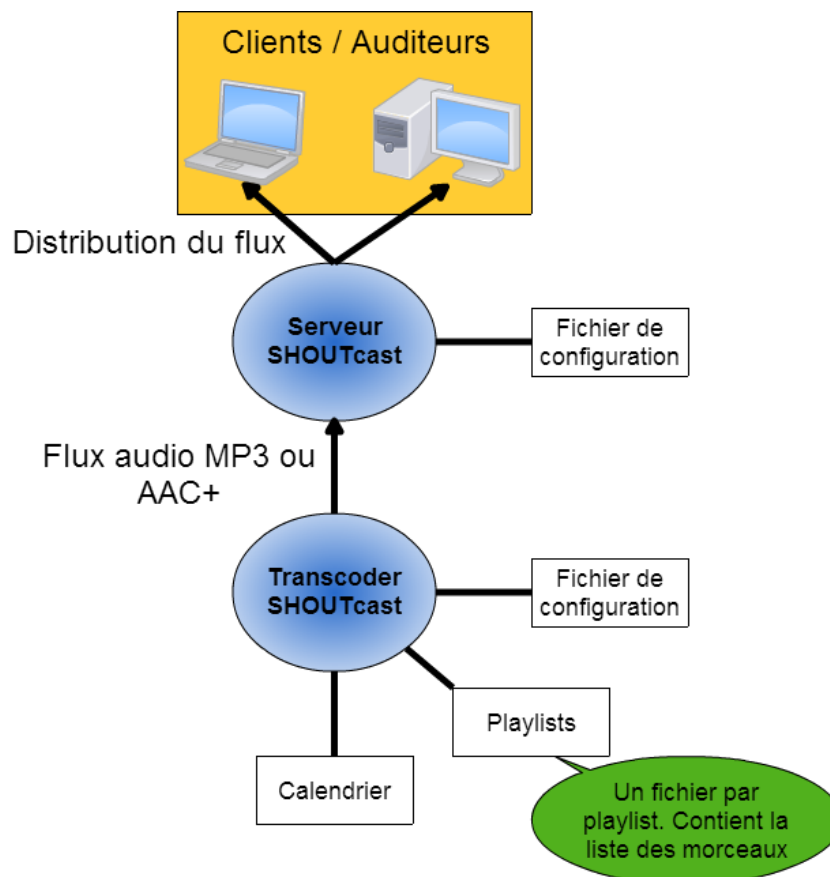


Figure 21 - Schéma shoutcast

## 5.6 Structures des dossiers/fichiers

### 5.6.1 Schéma

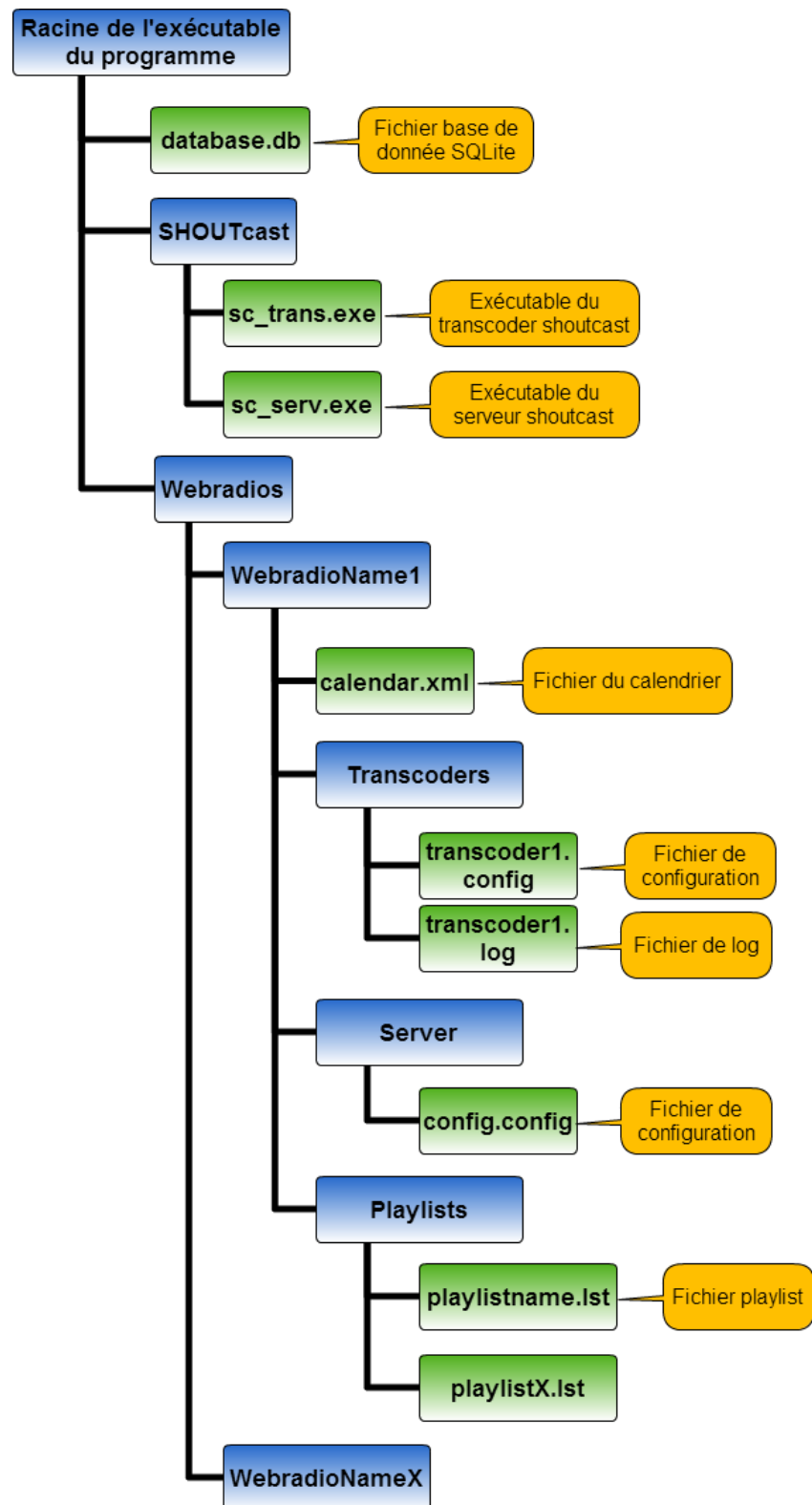


Figure 22 - Schéma structure des fichiers/dossiers

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.

### 5.6.2 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 : A 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 :

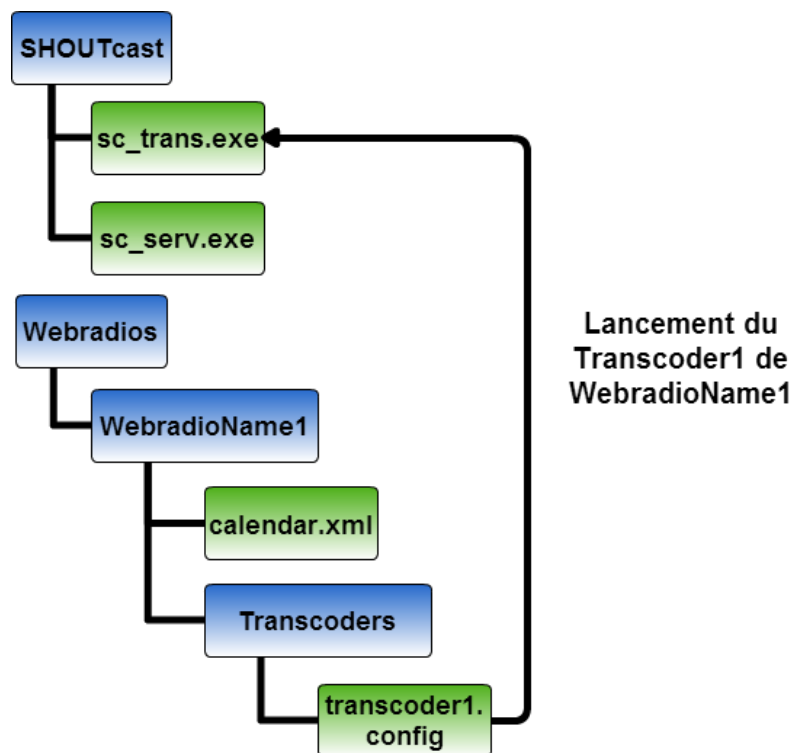


Figure 23 - Exemple lancement transcoder

## 5.7 Initialisation de l'application

La première fenêtre à se lancer est la SelectionView. C'est elle qui va appeler les différentes méthodes du modèle (via son contrôleur) servant à l'initialisation. C'est la classe Bdd qui s'occupe du traitement des données pour les passer ensuite au modèle pour que ce dernier remplisse ses champs « webradios » et « library ». Le diagramme de séquence suivant explique le déroulement de façon simplifiée.

IM

Figure 24 - Diagramme de séquence initialisation application

A la fin de se 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 modèle la concernant.

Le modèle 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 modèle.

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

Les 2 méthodes « LoadWebradios() » et « LoadLibrary() » de la classe Bdd s'occupe 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>ème</sup> charge les musiques présentes dans la bibliothèque ainsi que les playlist qui leur sont associées.

## 5.8 Webradio

### 5.8.1 Classes associée



Figure 25 - Classe "Webradio"

TODO : mettre à jour l'image en fin de rappoport + expliquer le ToString overrid

### 5.8.2 Affichage des webradios disponibles

Les webradios disponibles sont affiché dans la ListBox centrale de la fenêtre SelectionView. Pour la remplir, cella est effectué dans la méthode « UpdateView() » qui va récupérer les webradios du model et remplir la liste « d'items » avec les objets de type Webradio obtenus. C'est la méthode « ToString » de la classe Webradio qui va être appelée pour afficher le nom de la webradio dans la liste.



### 5.8.3 Création

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

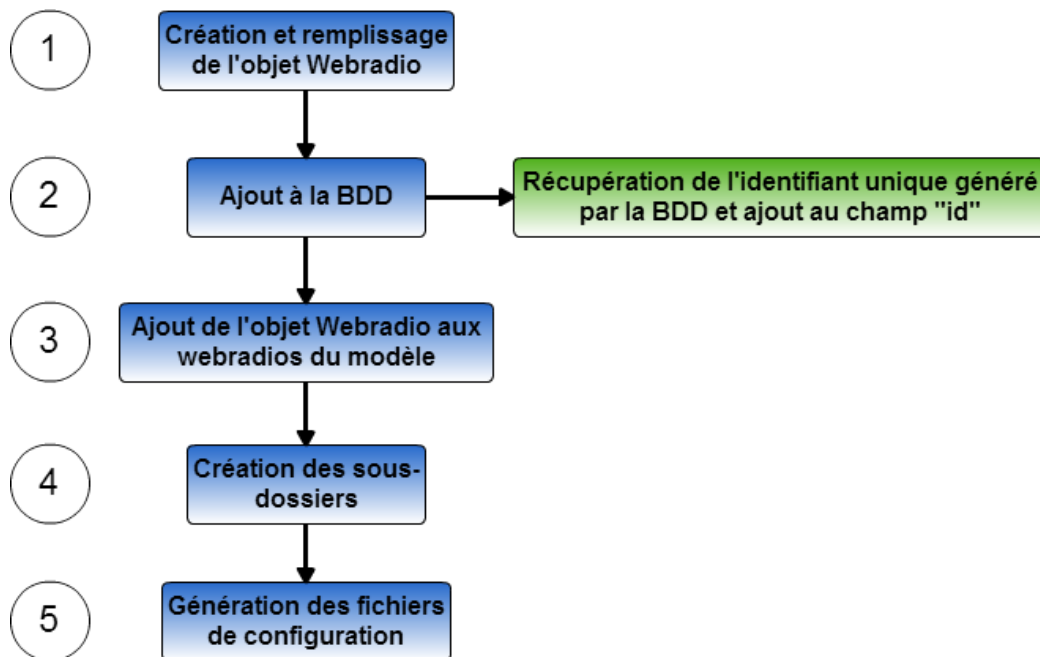


Figure 26 - Schéma création webradio

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

- 1 : Instanciation d'un nouveau objet de type Webradio avec le nom donné en paramètre à la méthode « CreateWebradio() ». Instanciation des objets nécessaires à la webradio (WebradioServer, WebradioCalendar et les différentes listes d'objets telle que la propriété Playlists). Seul la propriété « id » de l'objet webradio n'est pas rempli 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 modèle.
- 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'information sur cette méthode dans [ce chapitre](#).

### 5.8.4 Chargement

Une webradio est chargée lorsqu'elle est sélectionnée via SelectionView. Une fenêtre de type AdminView est ensuite ouverte avec les informations de la webradio sélectionnée. Ces informations

viennent du modèle. Voici le diagramme de séquence pour la création d'une nouvelle instance d'une AdminView :

☞

Figure 27 - Diagramme de séquence instanciation AdminView

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

A la création de la fenêtre, cette dernière appelle la méthode « CheckFolders » de son contrôleur. Cette méthode va faire de même sur le modèle. En fin de compte, cette méthode vérifie la présence de tous les dossiers nécessaires à la webradio lancé avec AdminView (server, playlists et transcoders). Si un d'eux n'est pas présent, il est créé.

Comme montré dans la 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 aller récupérer la webradio en question dans le modèle via son contrôleur.

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 caractère, les éléments de sa liste. Cette méthode « override » celle héritée par la classe parente

« Object ». 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 modèle. 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](#).

### 5.8.5 Duplication

TODO : depuis model -> dis à la bdd de dupliquer et duplique dans son model

Todo : Cloneable . Un clone() par sous objet de webradio ? Modifier la méthode « AddWebradio » pour parcourir tous les éléments de la webradio donnée ? et pas que nle calendrier et le server

### 5.8.6 Suppression

La suppression d'une webradio s'effectue via son identifiant. La méthode « DeleteWebradio() » du modèle va en premier temps supprimer le 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 Dictionnaire qui prend l'identifiant de la webradio à supprimer.

### 5.8.7 Changement de nom

TODO : expliquer le processus :

1. Renommer dans bdd table twebradio (return false si le nom existe déjà)
2. Renommer tous les filenames/configfilename/logfilename dans les enregistrements présents dans la bdd concernant des éléments de la webradio
3. Renommer dans le modèle (element + webradio)
4. UpdateObservers

TODO : dire que les processus seront arrêtés

### 5.8.8 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 (si il y en a un) et en crée 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 (simple 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.

## 5.9 Bibliothèque

### 5.9.1 Classes utilisées

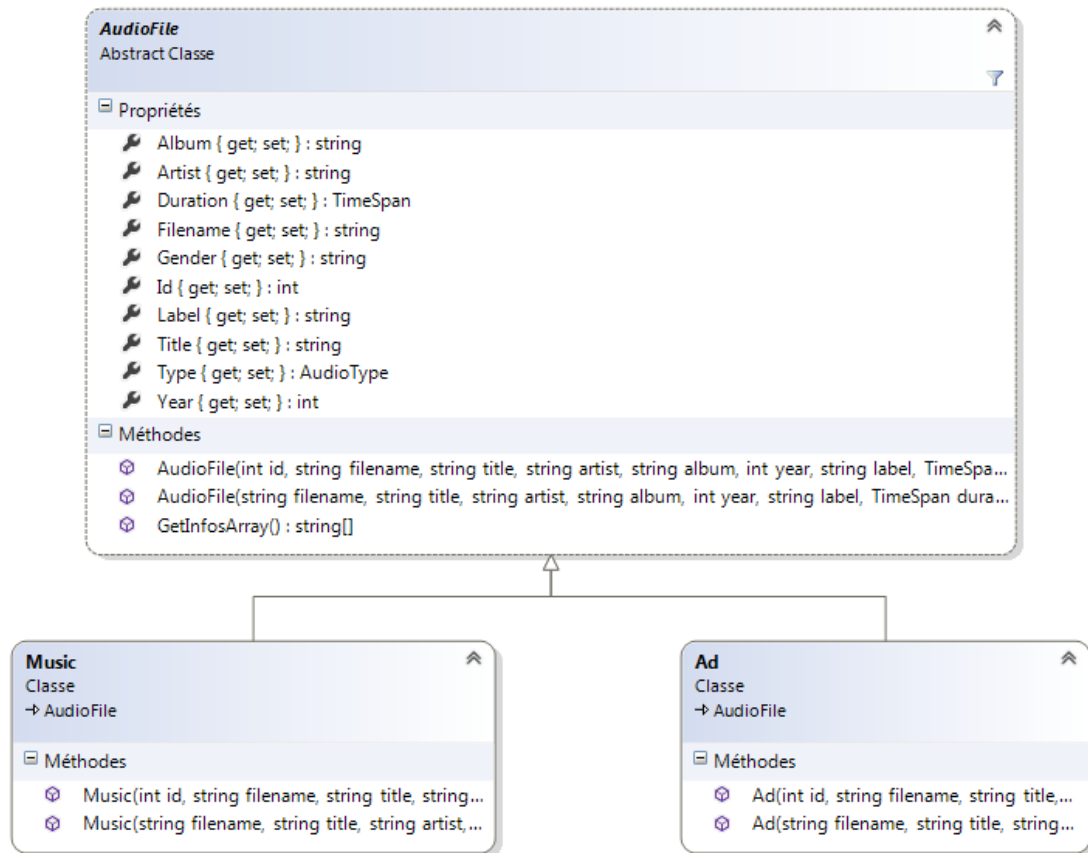


Figure 28 - Classes bibliothèque

### 5.9.2 MP3

A l'heure actuelle, seuls les fichiers MP3 peuvent être importés. Le 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.

TODO : explication algo de compression

### 5.9.3 Importation

Depuis l'onglet « library », l'utilisateur choisit si il veut importer un dossier ou des fichiers à la section « music » ou « ad ». Chaque bouton contient une valeur dans sa propriété « tag » qui est soit « Music » ou soit « Ad » car les événements « OnClick » des boutons (regroupé par type d'importation, c'est-à-dire, par dossier ou fichiers) pointe 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 Directory permet de récupérer un tableau de string contenant les « filename » (chemin absolu vers les fichiers) des fichiers correspondant au pattern donné en paramètre dans le dossier spécifié. Ce pattern se présente sous la forme : « \*.mp3 » pour

recupérer 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 modèle via son contrôleur. Le modèle va effectuer le traitement (analyse de tags ID3 et ajout à la base de donnée/modèle) 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 :

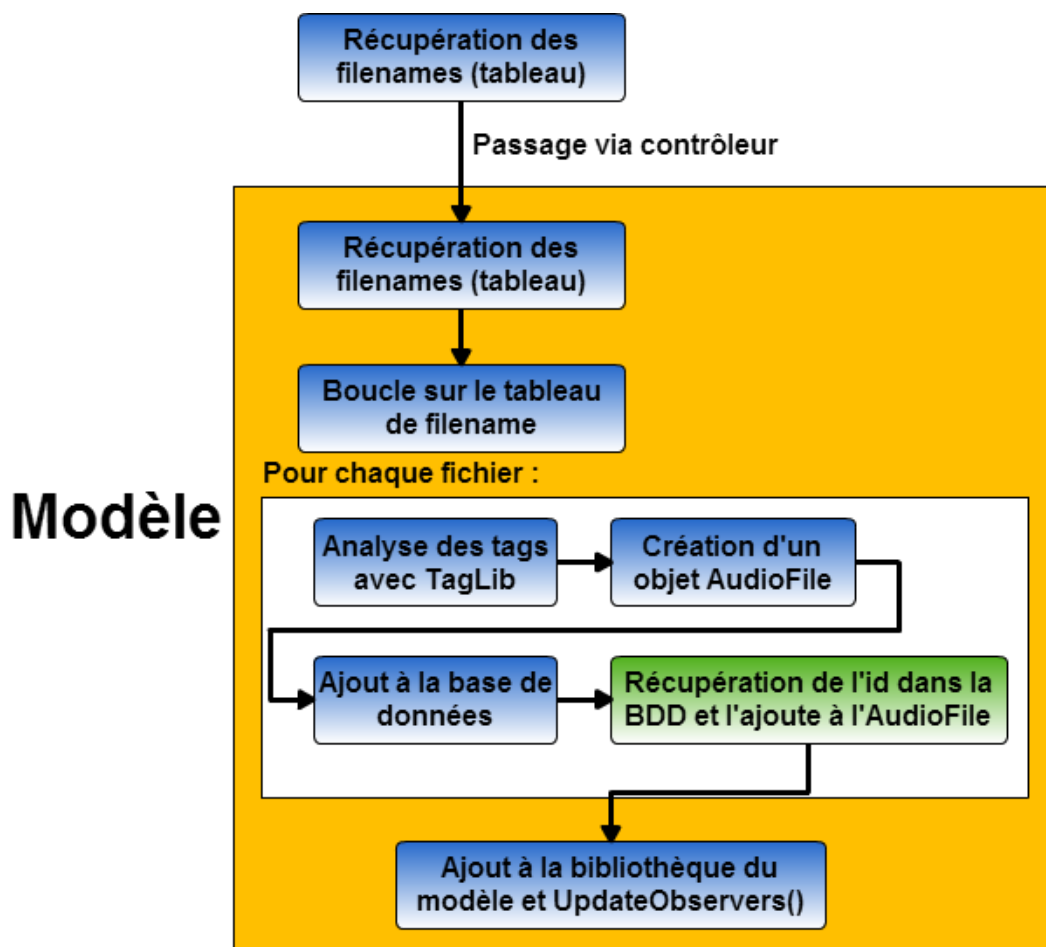


Figure 29 - Schéma importation fichier

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

#### 5.9.4 Tags ID3

ID3 est le nom des métadonnées pouvant être insérées dans un fichier audio comme MP3. Ces métadonnées 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

### 5.9.5 Analyse des tags

La récupération des tags ID3 est effectuée à l'aide de la bibliothèque « TagLib-Sharp » : <https://github.com/mono/taglib-sharp>

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 fourni par .NET) qui propose une méthode statique « 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 parlé.
- La propriété « Properties » qui contient des sous-propriétés comme « Duration », « BitRate » etc. Ce sont les informations concernant le fichier en lui-même.

TODO : la librairie lit les tag MusicBrainz

### 5.9.6 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 est ajouté à cette table 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.

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);
```

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.

### 5.9.7 Modification

TODO : les champs id, duration et path ne sont pas modifiable

TODO : Faire la meme vérification pour le genre que dans l'ajout de musique

TODO

### 5.9.8 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;
}
```

La présence d'un test ternaire 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 modèle et des playlist la concernant. Pour se faire, toutes les playlist 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 modèle puis dans la base de données.

### 5.9.9 Vérification des données

TODO : ajouter vfonctionnalité : vérifié l'intégrité des données (si les fichier référencés dans la bdd existe toujours) pour chaque section via un bouton « check files »

TODO : ajout a analyse fonctionnelle

### 5.9.10 Recherche

La recherche consiste à afficher seulement les lignes aillant 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) 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.

```
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;
}

```

## 5.10 Listes de lecture

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

### 5.10.1 Classes utilisées

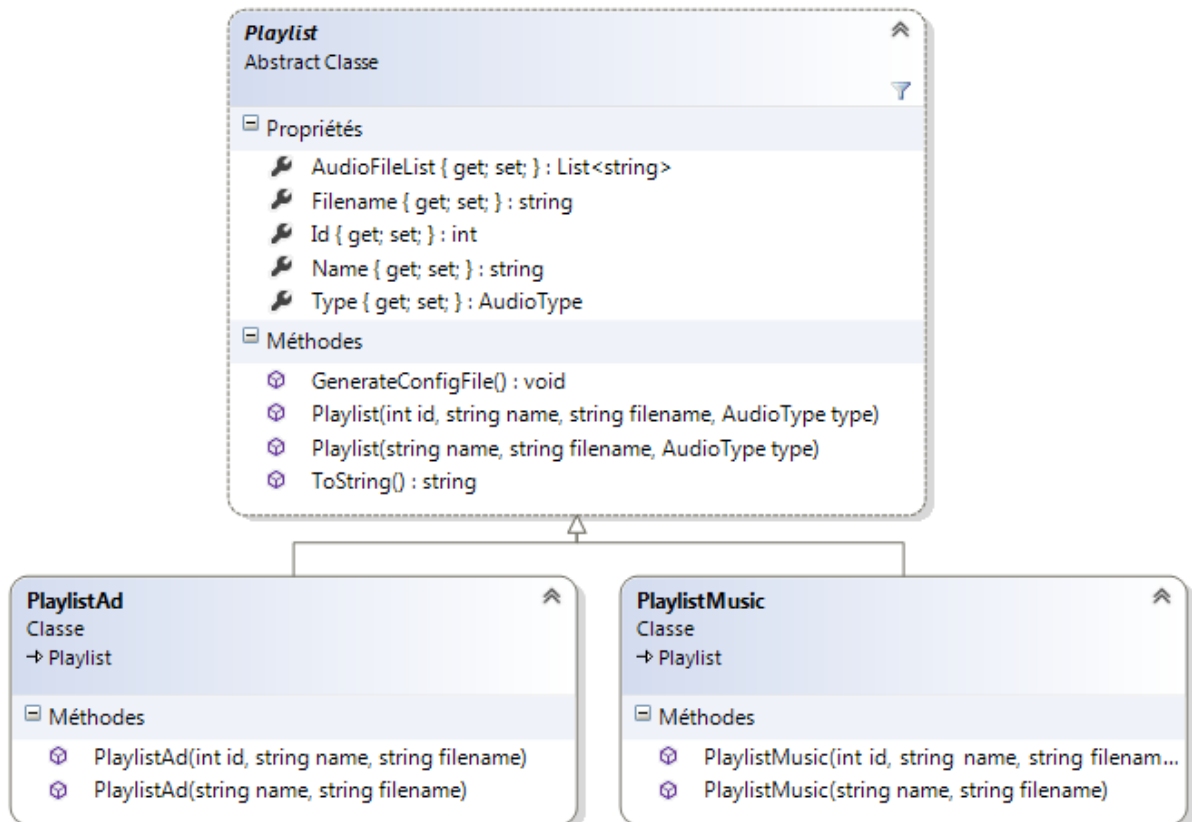


Figure 30 - Classes Playlist

### 5.10.2 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 Endless_Extended_Version-ToffMusic.mp3	Penguinz	-	Wistfulness\Wasted_Penguinz-
---	----------	---	------------------------------

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 diffusées.

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.

### 5.10.3 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 playlist 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 parlé, elle se décompose en plusieurs étapes qui se déroulent dans la méthode « CreatePlaylist » du modèle :

1. Création du futur filename du fichier de configuration de la playlist :  
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 appelé sur l'objet Playlist afin de créer son fichier de configuration (bien que vide pour le moment).
5. L'objet est ajouté au modèle (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.

### 5.10.4 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 modèle 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 demandé sans la dépasser. Il est déterminé que l'algorithme essaye 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.

A 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 :

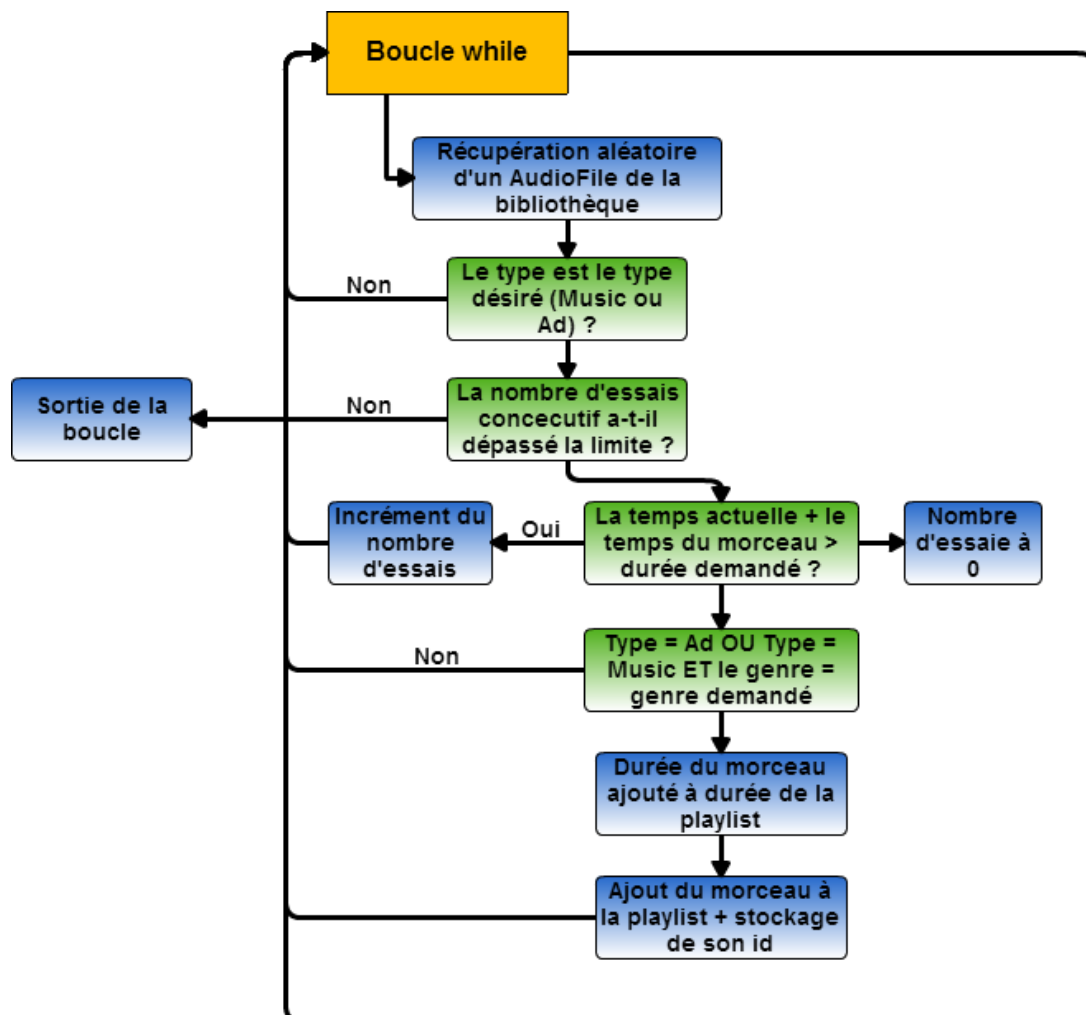


Figure 31 - Algorithme génération playlist

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 modèle. Pour

finir, le fichier de configuration de la playlist est généré et la méthode « UpdateObservers » est appelée.

#### 5.10.5 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. De ce fait, il est facile de détecter si quel type de morceau l'utilisateur veut ajouter à quel type de playlist.

En premier temps, la vue va générer une liste de type Dictionary<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 modèle d'ajouter les différents éléments dans le modèle 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 modèle via le contrôleur. 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 modèle retourne false. Sinon, la boucle se finit et la playlist génère sa configuration (GenerateConfigFile sur l'objet playlist).

#### 5.10.6 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 modèle) à l'exception que les éléments seront supprimés de la base de données (suppression du lien dans la table « tplaylist\_has\_music ») et du modèle. A la fin, la nouvelle configuration est générée.

#### 5.10.7 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 a 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'objets de type AudioFile en provenance du modèle via son contrôleur. Le modèle 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 additionné à une variable de type TimeSpan afin de calculer le durée totale de la playlist et de l’affichée.

#### 5.10.8 Suppression d’une playlist

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

## 5.11 Grille horaire

### 5.11.1 Classes utilisées

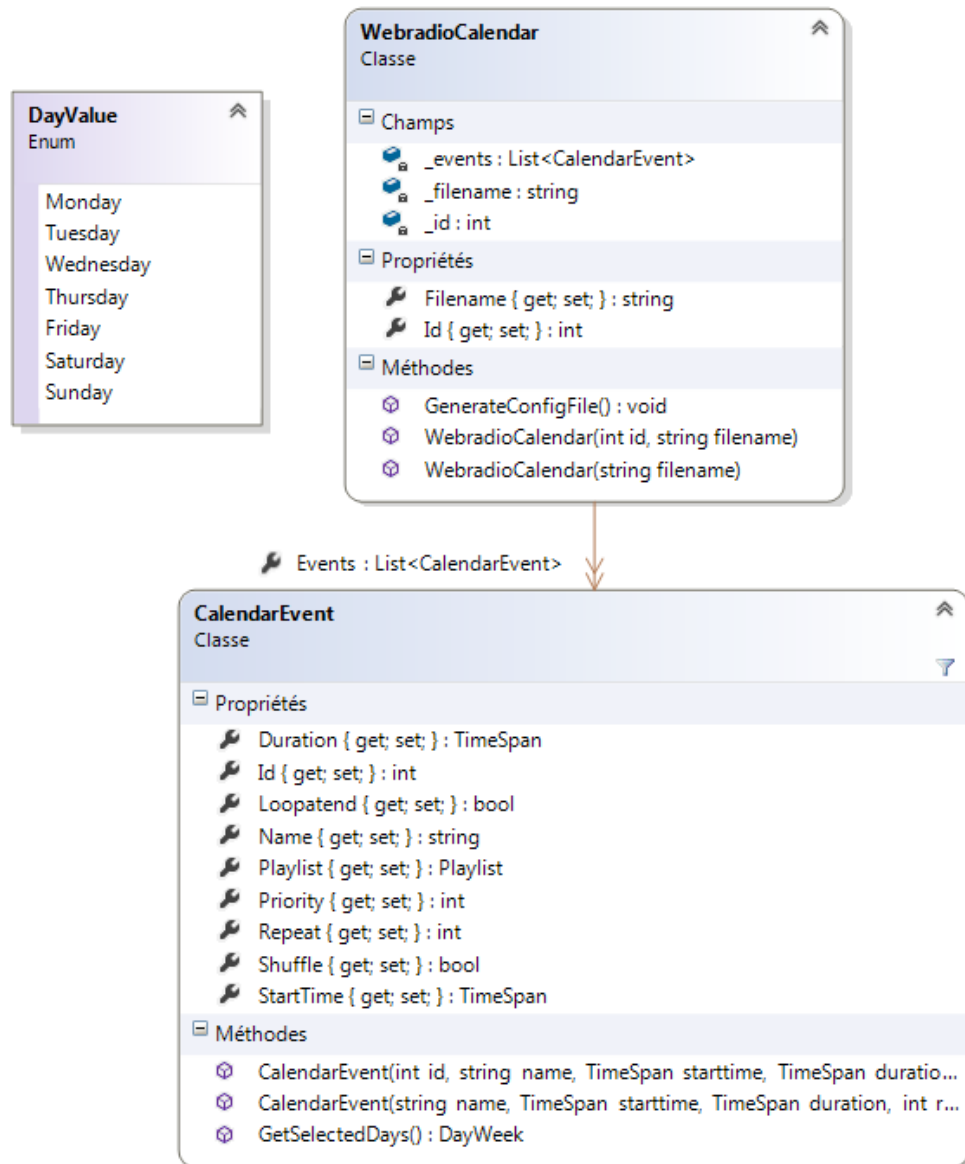


Figure 32 - Classes timetable

### 5.11.2 Outil utilisé

Pour l’affichage et la gestion de la timetable<sup>9</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 :

<sup>9</sup> Table du temps ou grille horaire en anglais.

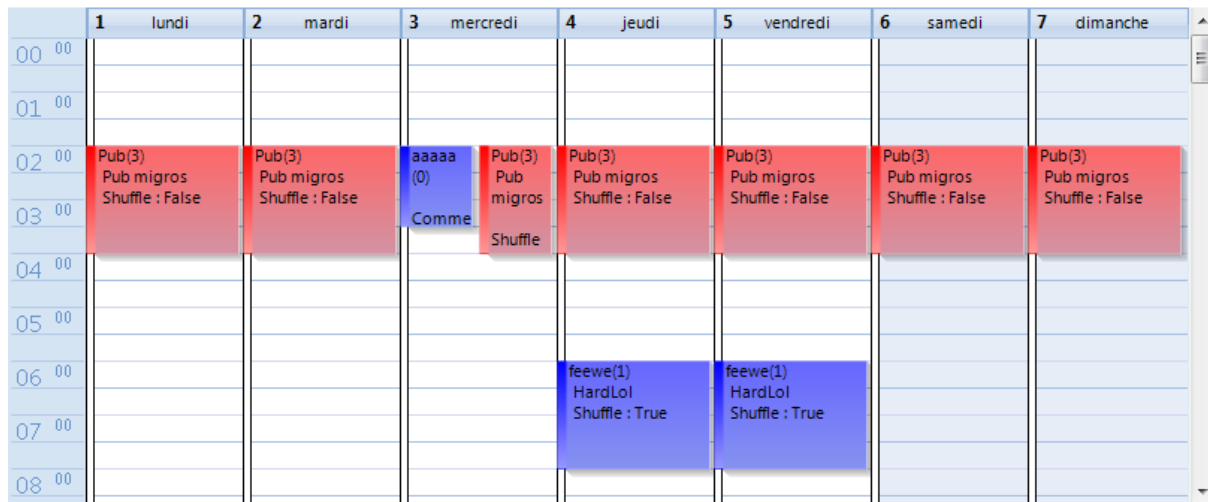


Figure 33 - Day View Calendar

### 5.11.3 Gestion du calendrier par ShoutCAST

C'est le transcoder fourni par ShoutCAST qui se charge de prendre en compte le calendrier et jouer les playlist 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 sur 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.
  - La nom de la playlist jouée (nom du fichier .lst)
- Un horaire : L'horaire définit les paramètres suivants :
  - « 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 données. 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).

#### 5.11.4 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);
```

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 34 - Exemple XML calendrier

#### 5.11.5 Affichage du calendrier

L'affichage du calendrier se fait avec le composant décrit dans le chapitre concernant [l'outil utilisé](#). 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;
```

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 seul 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);
```

La structure ainsi remplie va être récupérée lors d'UpdateView et utilisé 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
    }
}
```

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 test si le jour actuel a une valeur booléenne vraie dans le tableau retournée 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.

#### 5.11.6 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'événement sont automatiquement mise à jour en fonction de la sélection de l'utilisateur.

#### 5.11.7 Création d'un événement

Le formulaire de création est composé de 2 MaskedTextBox pour les champs « start time » et « duration ». Celle 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;
}
```

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 fourni les valeurs de chaque jours (définies 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 modèle 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 modèle. 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.

#### 5.11.8 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, seule 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 :

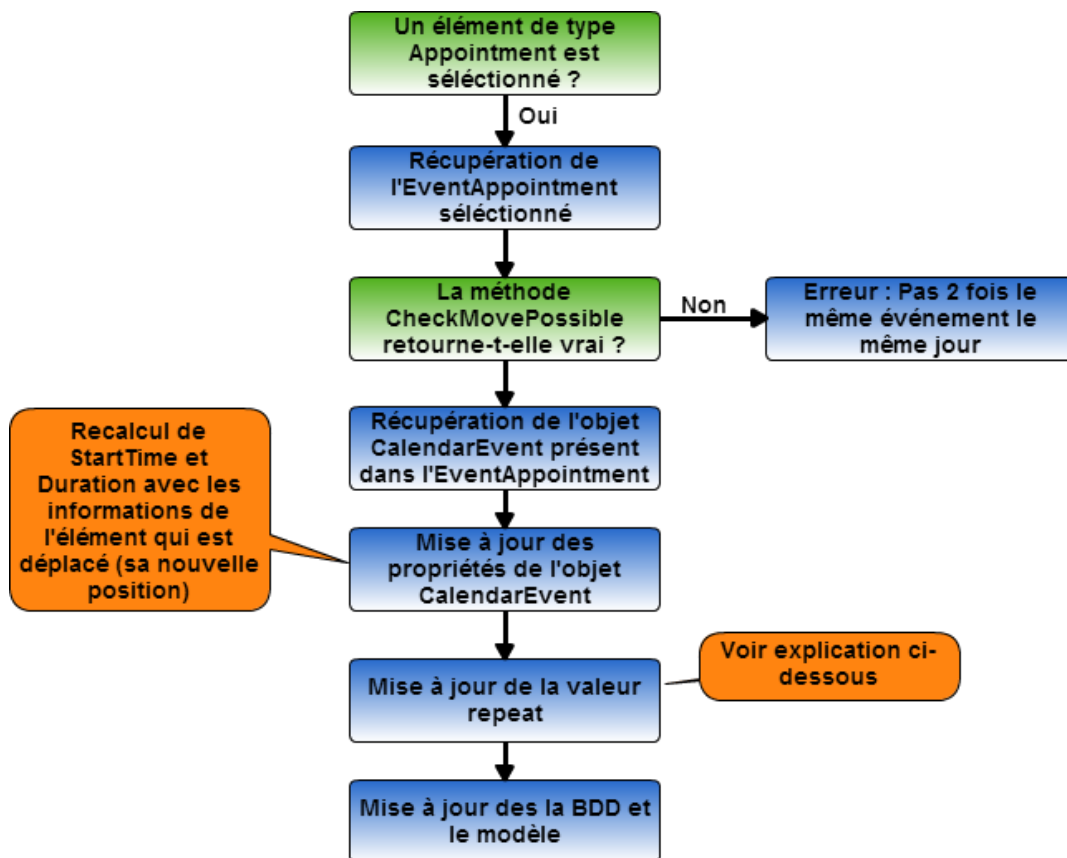


Figure 35 - Modification d'un événement

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 concernée 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 égale au jour en question. De cette façon :  
`repeat += (ev.StartDate.DayOfWeek==DayOfWeek.Monday)?(int)DayValue.Monday : 0;`

Pour finir, les configurations sont régénérées.

#### 5.11.9 Suppression d'un événement

Un clic droit sur un élément appelle l'événement « MouseClick » 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 contrôleur est appelée. L'objet CalendarEvent stocké dans l'EventAppointment est donné en paramètre. Cette méthode ira appeler la même méthode dans le modèle qui s'occupera, lui, de supprimer l'événement dans la BDD et ses propriétés. UpdateObservers est ensuite appelé ainsi que la régénération de configuration.

## 5.12 Transcoders

### 5.12.1 Classes utilisées

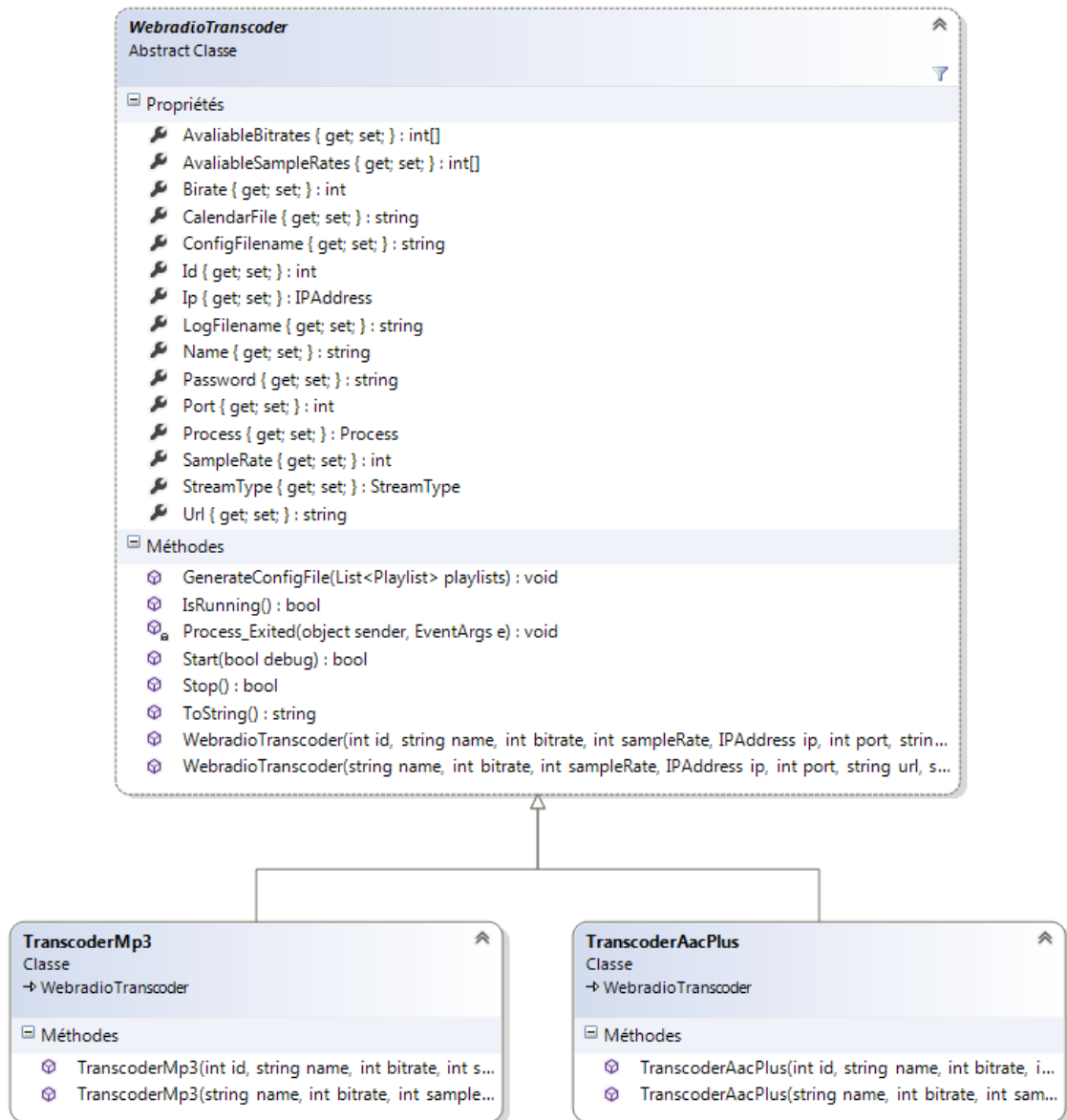


Figure 36 - Classes transcoders

### 5.12.2 Outil utilisé

ShoutCAST propose un outil en lignes de commandes nommé « 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 compresser 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

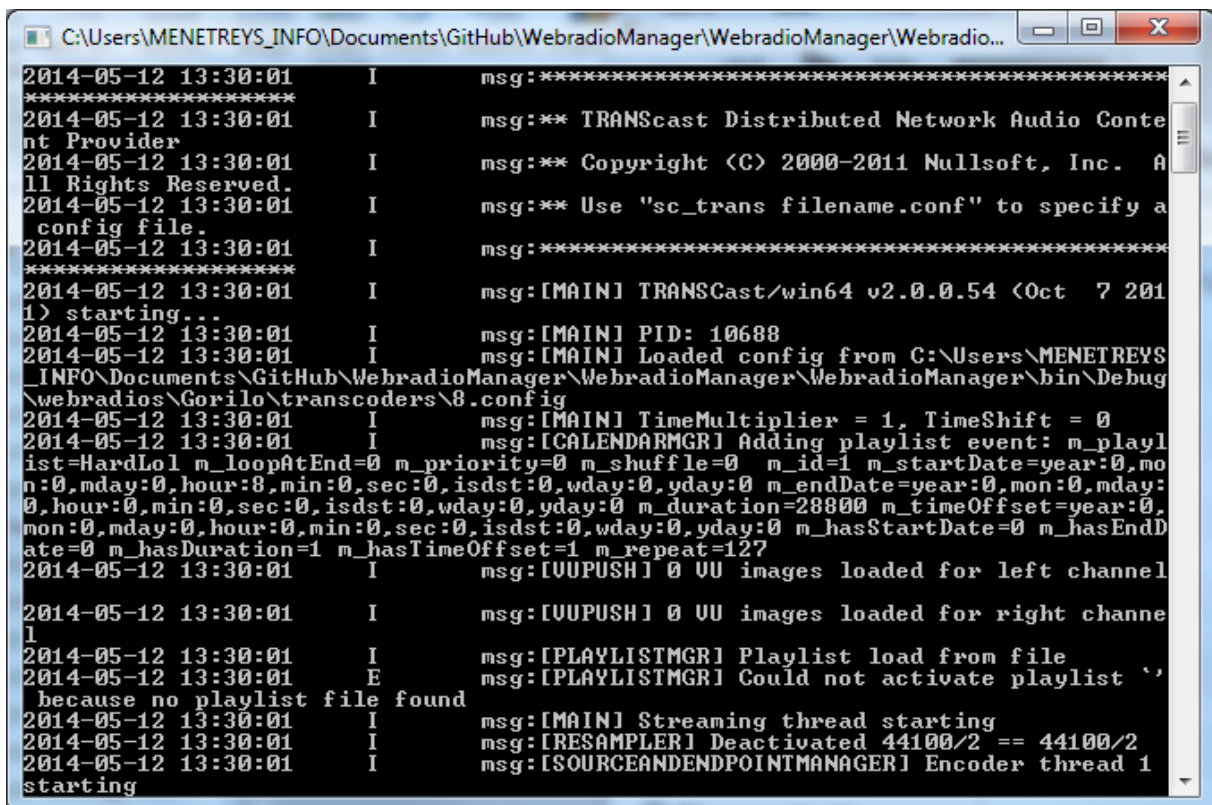
Dans le cas d'un webradio, le transcodeur sert à créer un flux à partir de fichiers audio afin de l'envoyer sur un serveur (serveur de diffusion) qui s'occupera de la diffusion aux clients/auditeurs.

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 le 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\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 VU images loaded for left channel
2014-05-12 13:30:01 I msg:[VUPUSH] 0 VU 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 37 - ShoutCAST transcoder console

Cet outil propose une API<sup>10</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 a été sélectionné. 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)

```

<sup>10</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.

```
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)

### 5.12.3 Définition des bitrates, taux d'échantillonnage et type d'encoder

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

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

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 };
```

### 5.12.4 Fichier de configuration et log

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'encoder (mp3 ou aacp)
- Bitrate\_1 : Le bitrate de l'encoder
- Adminport : Le port pour accéder au transcoder (pour envoi de requêtes ajax). 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 protocole utilisé 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 transcoders d'une même webradio pointent sur le même calendrier)

A 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 encoders 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 doivent 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](#)).

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\webradios\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\webradios\Gorilo\calendar.xml  
playlistfilename_1=Pub migros  
playlistfilepath_1=C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioMa  
nager\WebradioManager\bin\Debug\webradios\Gorilo\playlists\Pub migros.lst  
playlistfilename_2=Commercial  
playlistfilepath_2=C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioMa  
nager\WebradioManager\bin\Debug\webradios\Gorilo\playlists\Commercial.lst  
playlistfilename_3=HardLol  
playlistfilepath_3=C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioMa  
nager\WebradioManager\bin\Debug\webradios\Gorilo\playlists\HardLol.lst
```

```
playlistfilename_4=Test  
playlistfilepath_4=C:\Users\MENETREYS_INFO\Documents\GitHub\WebradioManager\WebradioMa  
nager\WebradioManager\bin\Debug\webradios\Gorilo\playlists\Test.lst
```

#### 5.12.5 Licence MP3

Pour diffuser en mp3, l'utilisateur doit obtenir une licence comme expliqué dans la documentation du transcoder 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.



### 5.12.6 Création d'un transcoder

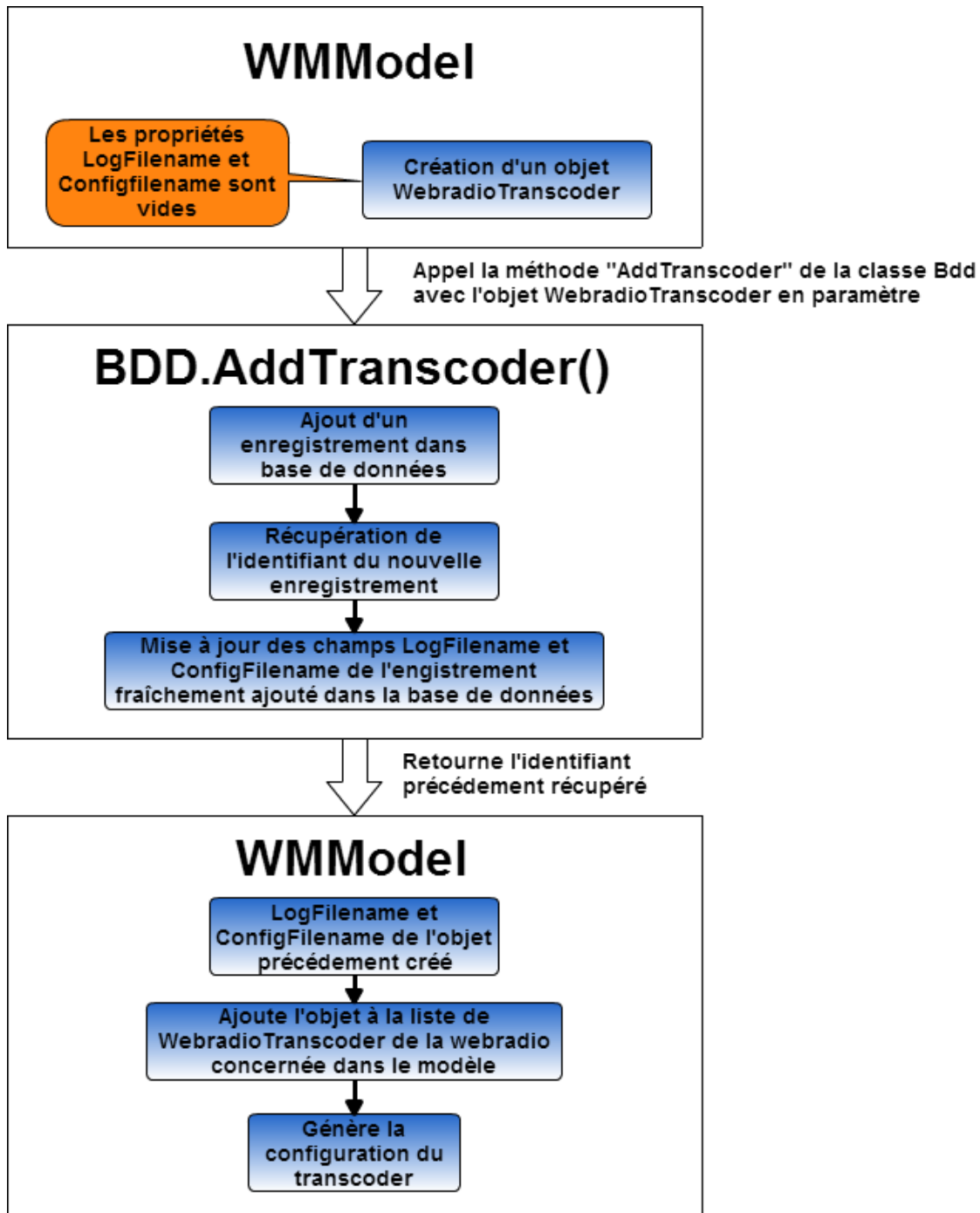


Figure 38 - Schéma création d'un transcoder

Le schéma ci-dessus explique le fonctionnement globale de l'ajout d'un nouveau transcoder à une webradio au niveau du modèle. 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 statique « 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 transcoder 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 modèle 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 modèle retourne « false » et la vue, dans ce cas, affiche un message d'erreur à l'utilisateur.

#### 5.12.7 Résolution de nom

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

#### 5.12.8 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 transcoder (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;
```

#### 5.12.9 Modification d'un transcoder

Pour la mise à jour des informations d'un transcoders, 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 mise à jour avec celles du formulaire de modification. L'objet modifié est ensuite envoyé au modèle via le contrôleur afin d'apporter les modifications dans la base de données et les données du modèle.

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.

#### 5.12.10 Suppression d'un transcoder

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

#### 5.12.11 Exécution et fermeture

La classe WebradioTranscoder propose 3 méthodes concernant la gestion du processus du transcoder 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('/', '\\')
};
```

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 « tué ». 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 transcoder est encore dans les processus système et le programme n'arrive pas à l'arrêter. Dans ce cas, l'utilisateur doit aller tuer le processus lui-même.
- 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](#) ».

### 5.12.12 Administration web (weblet)

La fonctionnalité « next track » permet de passer à la musique suivante dans la playlist qui est jouée par le transcoder 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:"+transcoder.AdminPort+"/api",
"POST", data);
```

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 transcoder (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.

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

### 5.12.13 Capture live

TODO : expliquer listing des périphérique audio

TODO : explication utilisation de l'API pour activer et désactivé

TODO : utilisation de la boucle de surveillance des stgatus (pour l'historique) pour mettre à jour la valeur « capture » de chaque classe WebradioTranscoder

TODO : fonctionnalité en beta, instable

### 5.12.14 Historique

L'historique de chaque transcoder est enregistré dans la base de données (table thistory). L'historique référence quelle musique a été jouée par quel transcoder à quelle date.

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 transcoders 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](#)) 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>
  2  <activesource source="playlist|capture|dj|relay">
  3    <currenttrack/>
  4    <nexttrack/>
  5    <name/>
  6    <file/>
  7    <ip/>
  8    <port/>
  9    <url/>
 10    <sourcetype/>
 11    <bitrate/>
 12    <device/>
 13    <input/>
 14    <samplerate/>
 15    <channels/>
 16  </activesource>
 17 </endpointlist>
```

```
18 <endpoint>
19 <name/>
20 <bytessent/>
21 <status/>
22 </endpoint>
23 <endpoint>
24 <name/>
25 <bytessent/>
26 <status/>
27 </endpoint>
28.
29. </endpointlist>
30. </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.

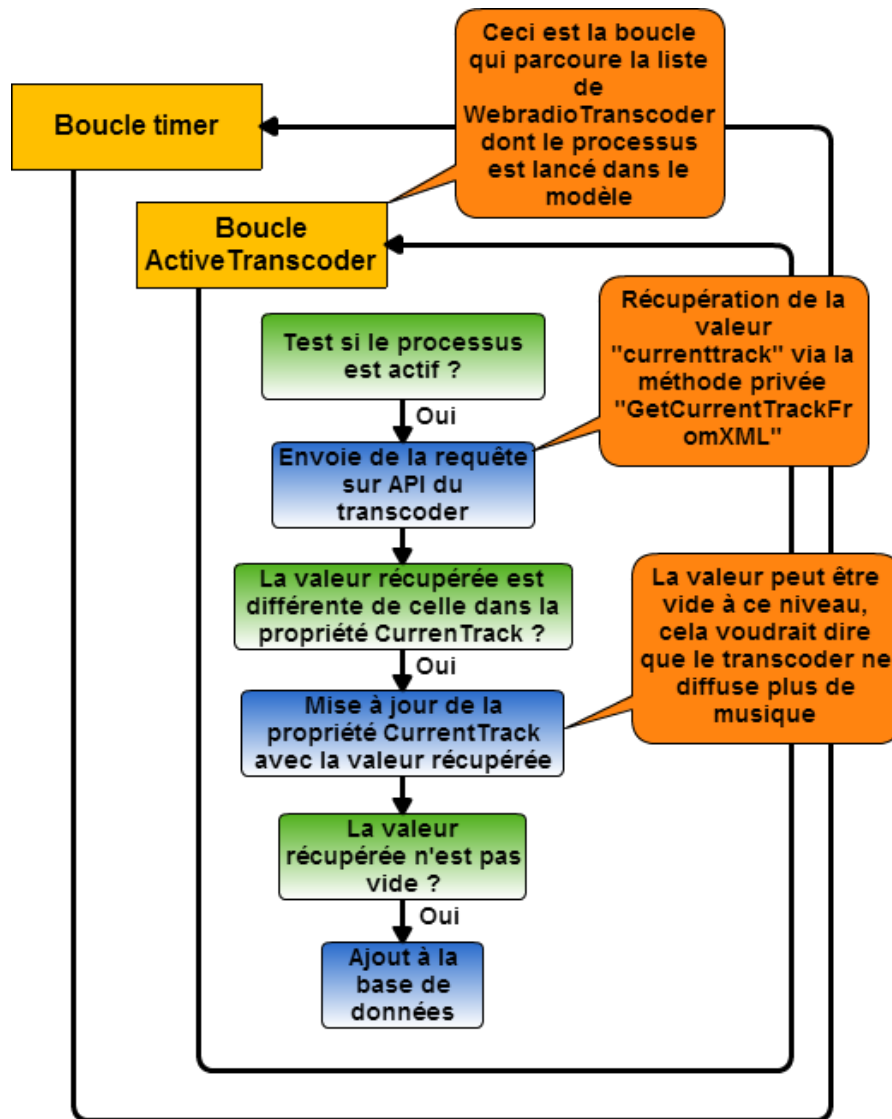


Figure 39 - Schéma gestion de l'historique

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 modèle 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.

## 5.13 Serveur de diffusion interne

### 5.13.1 Classe utilisée

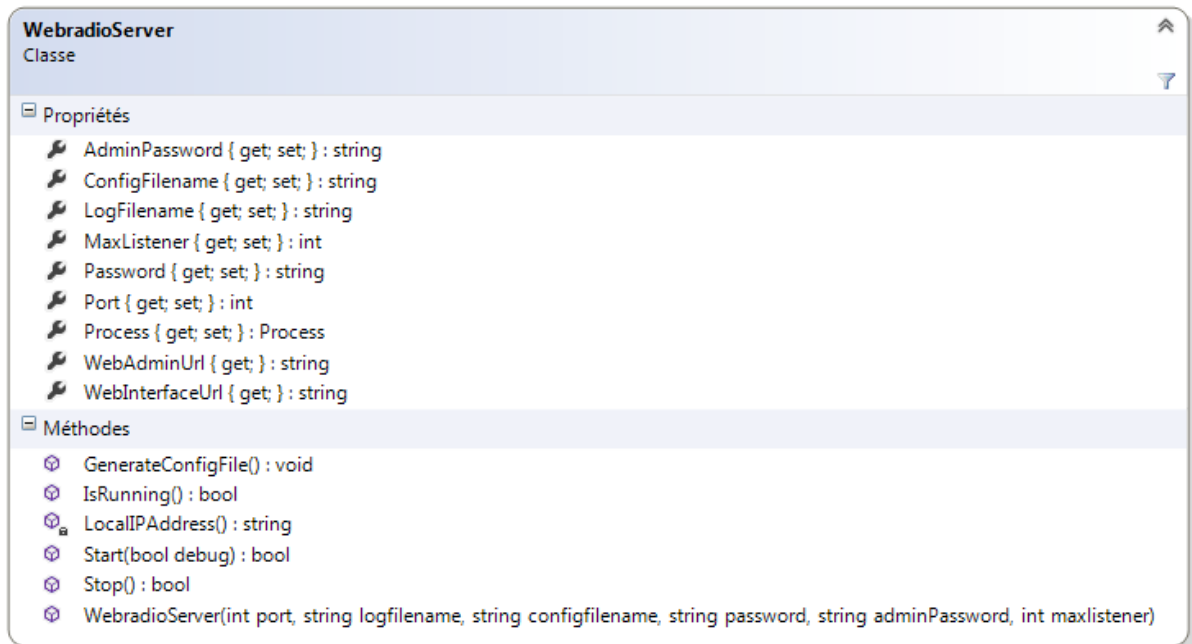


Figure 40 - Classe WebradioServer

TODO : ajouter Listener et WebradioServerStats

### 5.13.2 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 utilisé dans ce projet. Le serveur fonctionne avec un fichier de configuration qui lui est donné 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\Webrad
ioManager\bin\Debug\webradios\Gorilo\server\config.config -> change autodumpsour
cetime_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
radios\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 41 - 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](#).

### 5.13.3 Configuration

Voici la liste des paramètres configuré 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 seconde 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\WebradioMa
nager\WebradioManager\bin\Debug\webradios\Gorilo\server\log.txt
portbase=8000
password=lol
adminpassword=admin
publicserver=always
maxuser=32
autodumpsourcetime=0
```

#### 5.13.4 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 modèle via le contrôleur 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 modèle. En fin de traitement, la méthode « UpdateObservers » est appelée.

#### 5.13.5 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('/', '\\')
};
```

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 fermé manuellement par l'utilisateur.

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

### 5.13.6 Statistiques et auditeurs

TODO : UpdateStats et propriété WebradioServerStats

TODO : Il y a 2 types d'update effectué avec le bouton update :

- UpdateListeners : la vue recois une liste de Listener en provenance du modèle
- UpdateServerStats : depuis le modèle, la propriété Stats du server est mise à jour, puis UpdateObservers est appelé (c'est dans UpdateView que les stats serveur seront affichés)

TODO : bouton « kick » va kicker via l'api et l'événement utilisé

### 5.13.7 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 modèle. 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;
    }
}
```

## 5.14 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>11</sup>

### 5.14.1 IsRunning

Chaque classe comportant une priorité de type Process 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;
```

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 fermée 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.

### 5.14.2 Détection des crashes/fermetures externes

Il est possible que les processus lancé par le logiciel soit fermés par une application externe ou qu'ils soient « tués » par l'utilisateur en passant par le gestionnaire de tâche 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.

Pour se faire, le modèle dispose de 2 propriété 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 sensé est en fonctionnement. Ensuite, un timer (une minuterie à laquelle est attaché une méthode qui est appelé 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.

### 5.14.3 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.

<sup>11</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)

## **6 Tests**

## **7 Plannings**

### **7.1 Prévu**

### **7.2 Final**

## **8 Apports personnels**

## **9 Conclusion**

## 10 Améliorations possibles

- Multi-serveur de diffusion par transcoder
- Evènements périodiques (+ top horaire)
- DJ
- Gestion de semaine « type » (gestion multi semaine)

## 11 Références

- <https://cacoo.com> : Création de diagrammes en ligne
- <http://balsamiq.com/> : Création de « mokuup »
- <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...)

## 12 Annexes