



# DOCUMENTATION TECHNIQUE

---

## Contenu

<b>Interface graphique.....</b>	<b>2</b>
<b>Guide d'utilisation de l'application .....</b>	<b>2</b>
Description de l'interface .....	2
<b>Partie technique.....</b>	<b>2</b>
Description du fichier AAP.java .....	3
<b>Fonctionnalités.....</b>	<b>4</b>
<b>Tri des fichiers.....</b>	<b>4</b>
<b>Configuration des répertoires de stockage.....</b>	<b>5</b>
<b>Equalizer .....</b>	<b>5</b>
<b>Lyrics / Paroles de chansons .....</b>	<b>6</b>
<b>Enregistreur .....</b>	<b>7</b>
<b>Fonction boucle .....</b>	<b>7</b>
<b>Sauvegarde d'une boucle .....</b>	<b>8</b>
<b>Chargement d'un boucle .....</b>	<b>9</b>

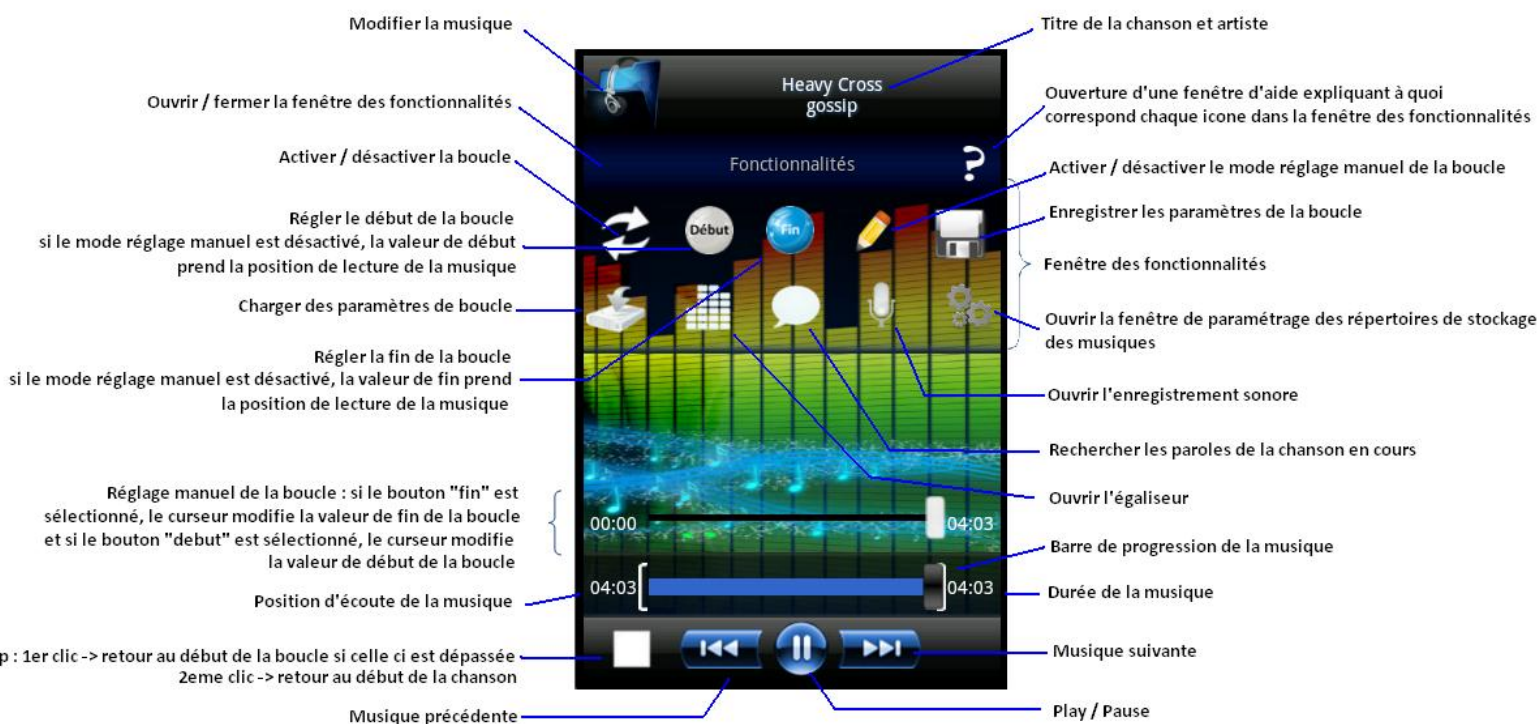


# Interface graphique

## Guide d'utilisation de l'application

Lors du démarrage de l'application, on doit sélectionner une musique parmi celles stockées sur la carte SD du téléphone. Une fois la musique sélectionnée, on arrive sur la fenêtre principale de notre application. La musique se met automatiquement en route.

## Description de l'interface



## Partie technique

Tout ce qui concerne l'interface graphique est stocké dans le répertoire « res » du projet et dans le fichier **AAP.java** (package **fr.unice.aap**).

- Images : drawable-hdpi (pour que la taille en pixel des images soit liée à la densité « high » et que lors d'un changement de densité, la taille des images s'adapte automatiquement)
- Animation de changement d'images pressé/relevé d'un bouton : drawable
- Animation d'apparition et disparition des fenêtres : anim
- Les fichiers de mise en page portrait (fichiers par défaut) : layout
- Les fichiers de mise en page paysage : layout-land



- Chaines de caractères : values et values-es (espagnol)

### Description du fichier AAP.java

Il s'agit du fichier java lié à notre interface. Il contient déjà la javadoc complète expliquant en détail la fonctionnalité de chaque méthode.

### Événements

Le fichier contient tous les évènements liés à une action sur les boutons de l'interface.

Pour l'action « clic » : la propriété « onClick » de l'objet est remplie par le nom de la méthode et dans le fichier AAP.java elle est déclarée de cette manière :

```
public void nomDeLaMethode(View v){...}
```

### Synchronisation barre de progression / lecture de la musique

Lors du démarrage de la lecture de la musique un thread est lancé ([progressUpdater](#)). Celui-ci modifie la position de la barre de progression en fonction de la position de la lecture de la musique. C'est aussi dans ce thread que le contrôle de la boucle est effectué.



## Fonctionnalités

### Tri des fichiers

Bouton de lancement : **android:id="@+id/btnDossier"**.

Méthode associée au clic du bouton : **public void openDossierMusic(View v)** de la classe **AAP**. Cette méthode se contente de lancer l'activity **MusicListActivity** du package **fr.unice.aap.musics**.

**MusicListActivity** initialise ensuite le menu des options de tri des fichiers audio et récupère les fichiers audio supportés présents dans les répertoires de stockage configurés.

L'activity **MusicListActivity** comporte également toutes les méthodes communes au tri des fichiers audio de manière à centraliser le rafraîchissement de la liste des musiques à afficher selon l'option de tri sélectionnée.

Une fois que l'utilisateur sélectionne une option de tri (« Toutes les chansons », « Artiste », « Album », « Genre »), **MusicListActivity** va lancer l'activity correspondante à l'option de tri sélectionnée (respectivement **AllSongsListActivity**, **ArtistsListActivity**, **AlbumsListActivity**, **GenresListActivity**).

Si l'utilisateur clique sur le bouton « retour » de son téléphone, le résultat de l'activity en cours sera 0 :

```
@Override
public void onBackPressed() {
    this.setResult(0);
    this.finish();
}
```

En revanche, s'il sélectionne une musique, il faut retourner au lecteur et donc arrêter les activity en cours (en dehors de l'activity du lecteur, le résultat de l'activity sera donc 1 :

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    this.setResult(resultCode);
    if(resultCode == 1) {
        this.finish();
    }
}
```

Pour la récupération des métadonnées à afficher (cf Rapport), on utilise un objet de type **android.media.MediaMetadataRetriever**. Cette classe permet de récupérer aisément les métadonnées présentes au sein d'un fichier audio (cf activity **AllSongsListActivity**).



## Configuration des répertoires de stockage

Bouton de lancement : **android:id="@+id/config"**.

Méthode associée au clic du bouton : **public void config(View v)** de la classe **AAP**. Cette méthode affiche une fenêtre de type **AlertDialog** pour proposer une interface affichant la liste des répertoires de stockage sélectionnés et la possibilité d'en ajouter de nouveau.

Si l'utilisateur clique sur le bouton **android:id="@+id/addMusicDirectory"**, la méthode **ajouterDossierMusique(View v)** de la classe **AAP** sera exécutée. Cette dernière se contentera de lancer l'activity **FileExplorerActivity**.

L'activity **FileExplorerActivity**, correspond à une sorte d'explorateur de fichiers présents au sein du téléphone. L'utilisateur partira alors de la racine, et parcourra les dossiers du téléphone, jusqu'à sélectionner le dossier courant en cliquant sur le bouton **android:id="@+id/btnSelectRepCourant"**.

Si l'utilisateur veut parcourir un dossier mais que ce dernier est inaccessible à la lecture, une fenêtre d'erreur sera affichée. Le code correspondant à cet vérification se trouve dans la méthode **protected void onItemClick(ListView l, View v, int position, long id)** de l'activity **FileExplorerActivity**.

Lorsque l'utilisateur sélectionne le répertoire courant, le chemin de ce dernier est récupéré et sauvegardé dans les paramètres d'application sous la clé « **music\_files\_paths** ». La récupération et la sauvegarde de paramètres d'application se réalise à l'aide des interfaces **android.content.SharedPreferences** et **android.content.SharedPreferences.Editor** (**Editor** étant une classe interne de la classe **SharedPreferences**).

## Equalizer

Bouton de lancement : **android:id="@+id/tonalite"**.

Méthode associée au clic du bouton : **public void openEqualizer(View v)** ligne 612 du fichier **AAP.java**. Cette méthode se contente de lancer l'activity **EqualizerActivity** du package **fr.unice.aap**.

**EqualizerActivity** initialise ensuite tout ce qui est nécessaire au fonctionnement de l'equalizer et du visualizer. Comme décrit dans le rapport, l'Equalizer va rechercher le nombre de bandes de fréquence dont est composé le morceau en cours de lecture (ligne 74). Ce morceau est récupéré (ligne 69) via un champ static de la classe **AAP** :

```
mEqualizer = new Equalizer(0, AAP.mPlayer.getAudioSessionId());
```

Les Seekbar qui permettent la variation de la valeur des bandes de fréquence sont alors générées « en dur » et non pas via un fichier xml.

L'equalizer est réinitialisé à chaque changement de chanson dans le fichier **AAP.java**, pour cela :

1. L'**EqualizerActivity** est stockée dans **AAP.java** grâce à la ligne suivante de **EqualizerActivity.java** (ligne 51)  
**AAP.equalizerActivity = this;**



2. Lors de chaque changement de piste (appel des méthodes `musiqueSuivante` et `musiquePrécédente`) la méthode réinitialisation est appelée :  
**`equalizerActivity.resetEqualizer()`**;

**Bug connu :** Les modifications apportées à une gamme de fréquence sont effectives sur le morceau en cours de lecture mais les valeurs ne sont pas sauvegardées. Concrètement si on ferme la fenêtre après modifications et qu'on l'ouvre une nouvelle fois, les valeurs par défaut seront affichées.

## Lyrics / Paroles de chansons

Bouton de lancement : **`android:id="@+id/paroles"`**.

Méthode associée au clic du bouton : **`public void findLyrics(View v)`** ligne 630 du fichier `AAP.java`. Comme toutes les méthodes issues d'un clic de notre interface, le passage en paramètre d'une vue est obligatoire, cependant la vraie méthode `findLyrics` sera déclenchée autrement que via le bouton c'est pourquoi la surcharge ayant pour paramètre une vue est créée.

**`public void findLyrics()`** ligne 634 réalise donc le vrai travail :

- Elle déclenche l'animation pour le retrait du menu de fonctionnalités,
- Elle rend visible la `ScrollView` qui contiendra toutes les informations,
- Elle appelle ensuite la méthode **`setLyricsTextResult(String artist, String song, boolean estRetour)`** avec comme paramètres la chanson en cours de lecture, ainsi une première recherche est effectuée automatiquement sans que l'utilisateur ne rentre d'informations.

La méthode **`setLyricsTextResult(...)`** effectue alors la requête sur la base de donnée de **LyricsWiki** de la sorte :

- Si tous les champs `artist` et `song` sont renseignés, on récupère le résultat sous format `xml`, on le parse pour afficher l'échantillon de paroles ainsi qu'un bouton permettant d'ouvrir un navigateur web pour afficher l'intégralité des paroles.
- Si seul l'artiste est renseigné et que l'appel de la méthode n'est pas un retour (c'est un retour lorsque l'utilisateur quitte le navigateur web ouvert à l'aide des boutons dédiés, ce booléen est nécessaire car la fermeture du navigateur appelle la méthode `setLyricsTextResult`) un navigateur internet est alors ouvert sur la page de cet artiste affichant ainsi la liste complète de ses chansons et albums.
- Si la recherche n'a donné aucun résultat, un simple « Not Found » est affiché.

Lorsque le navigateur web a besoin d'être ouvert, la méthode **`openBrowser()`** est appelée. Cette méthode rend le conteneur de la page de recherche invisible et celui du navigateur visible.

Deux lignes importantes de cette méthode :

- **`lyricsView.getSettings().setJavaScriptEnabled(true)`**  
Active le JavaScript (les paroles sont affichées en JavaScript sur le site LyricsWiki).



- **lyricsView.setWebViewClient(new WebViewClient())**

Force l'ouverture de nouvelles pages dans une WebView. Par défaut, lorsqu'une nouvelle page doit être ouverte (click d'un lien) le navigateur internet par défaut du téléphone est lancé, sortant ainsi du cadre de notre application. En forçant l'ouverture d'une WebView on permet alors à l'utilisateur de quitter à tout moment.

## Enregistreur

Bouton de lancement : **android:id="@+id/rec"**.

Méthode associée au clic du bouton : **public void openRecordLayout(View v)**. Cette méthode déclenche l'animation pour le retrait du menu de fonctionnalités et rend visible la fenêtre d'enregistrement : **android:id="@+id/RecordLayout"**.

Ce conteneur est composé d'un champ modifiable correspondant au nom de l'enregistrement, d'un bouton qui déclenchera et arrêtera l'enregistrement et enfin d'un bouton pour minimiser la fonctionnalité.

La méthode principale liée à cette fonctionnalité est donc **recordSound(View v)** ligne 751. Cette méthode démarre un nouveau **Recorder** (fr.unice.aap.recorder.Recorder.java) dont le paramètre est le chemin du fichier qui sera enregistré. Ce chemin sera formaté pour assurer le bon fonctionnement de l'enregistrement.

Quelques lignes importantes :

- `recorder.setAudioSource(MediaRecorder.AudioSource.MIC);`  
Défini le microphone du téléphone comme source d'enregistrement.
- `recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);`  
Défini le format d'enregistrement en .3gp.

## Fonction boucle



Activation de la fonction boucle à l'aide du bouton :

### Classe AAP

Lors de l'évènement click sur ce bouton, la variable « `isLoop` » passe à « True » et permet donc de boucler dans un morceau de cette manière (ligne 970 à 991) :

```
/**
 * Thread qui synchronise la musique et la barre de musique + boucle
 */
private Runnable progressUpdater = new Runnable() {
    @Override
    public void run() {
        if(isPlay && !onTouchSeekBarMusic)
        {
            seekBar_Music.setProgress(mPlayer.getCurrentPosition());
        }
    }
}
```





```

        ((TextView)findViewById(R.id.position)).setText(heureToString(mPlayer.getCurrentPosition()));

        if(isLoop)//Si la fonction boucle est activé
        {
            // si la progression de la musique a dépassé la fin de la boucle
            if(mPlayer.getCurrentPosition() >= seekBar_fin.getProgress())
            {
                // on replace la lecture au début de la boucle
                mPlayer.seekTo(seekBar_debut.getProgress());
            }
        }

        mHandler.postDelayed(this, 1000);
    }
}
};

```

## Sauvegarde d'une boucle



Sauvegarde de la boucle courante en appuyant sur le bouton :

Ensuite l'utilisateur rentre un nom de boucle et en validant, cette nouvelle boucle va être sauvegardée dans le fichier XML **loopConfig.xml** à la **racine de la carte SD**.

Appel de la méthode **public static void addLoop(Loop loop)** ligne 594 (AAP .java).

Structure du fichier XML :

```

<root>
  <chanson titre="when_the_rain">
    <loop>
      <nom>Loop_WhentheRain</nom>
      <debut>10000</debut>
      <fin>20000</fin>
    </loop>
  </chanson>
  <chanson titre="cosmic_girl">
    <loop>
      <nom>Loop_Cosmicgirl_1</nom>
      <debut>30000</debut>
      <fin>50000</fin>
    </loop>
    <loop>
      <nom>Loop_Cosmicgirl_2</nom>
      <debut>35000</debut>
      <fin>64600</fin>
    </loop>
  </chanson>
  <chanson titre="aerodynamique">
    <loop>
      <nom>Loop-aerodynamique</nom>
      <debut>15000</debut>
      <fin>28000</fin>
    </loop>
  </chanson>
</root>

```





- <chanson titre= « when\_the\_rain »> correspond à la chanson titrée « when\_the\_rain »
- <loop> correspond à la boucle
- <nom> correspond au nom de la boucle
- <debut> correspond au début de la boucle en milliseconde
- <fin> correspond à la fin de la boucle en milliseconde

### Classe ExtractLoopConfig

La méthode **public static void** addLoop(Loop loop) se charge de parcourir le fichier XML pour sauvegarder la boucle « loop ».

Premièrement, la méthode vérifie si le nœud <chanson> correspondant a déjà été créé.

Si oui, la méthode rajoute un nœud <loop> au nœud <chanson> correspondant.


Si non, la méthode crée un nœud <chanson> correspondant à la chanson de la boucle, et crée à l'intérieur de ce nœud <chanson> un nœud <loop>.

### Chargement d'une boucle



Chargement de la boucle en appuyant sur le bouton :

Ensuite l'utilisateur choisit une boucle déjà enregistrée pour la chanson en cours de lecture.

 Chargement
annuler
00:00:40-00:00:50
00:01:48-00:02:38
00:02:01-00:03:01

Appel de la méthode **public static** ArrayList<Loop> getLoops(String titre) ligne 549 (AAP.java).

Puis appel de la méthode **public static** Loop getLoop(String nomLoop, String titre) ligne 566 (AAP.java) pour récupérer dans le fichier XML la boucle choisie dans la boîte de dialogue ci-dessus.

### Classe ExtractLoopConfig

La méthode **public static** ArrayList<Loop> getLoops(String titre) se charge de chercher les boucles de la chanson en cours de lecture pour remplir la boîte de dialogue ci-dessus

La méthode **public static** Loop getLoop(String nomLoop, String titre) se charge de récupérer la boucle choisie dans la boîte de dialogue.