## Le composant de base : lecteur MP3

### Objectif

La base de notre projet étant dans un premier temps de permettre à l’utilisateur d’utiliser les fonctionnalités d’un lecteur mp3 classique, il était donc primordial de trouver un composant d’android permettant d’effectuer différentes opération basique telle que la lecture d’un fichier sonore, la possibilité de stopper la lecture, etc …

Après quelques recherches et avec l’aide généreuse de l’étudiant Christophe DUC (Master 2 NTDP), nous avons décidé d’étudier le composant MediaPlayer du framework android.

### Etude du composant MediaPlayer

Dans un premier temps, nous avons consulté de manière approfondie la documentation officielle du composant MediaPlayer.

Dans un second temps, nous souhaitions pouvoir manipuler des exemples de MediPlayer pour approfondir nos connaissances et pouvoir commencer notre propre lecteur mp3.

Pour cela, nous avons étudié et testé le lecteur basique que propose Igor Khrupin sur son blog :

(<http://www.hrupin.com/2010/12/15/simple-android-mp3-media-player>)

[Image : mp3\_player\_khrupin]

Nous avons également étudié le code source que nous a fourni Christophe DUC, représentant une ébauche d’un lecteur mp3 sous android.

### Fonctionnement

Le fonctionnement du MediaPlayer est assez simple, dans le sens ou toutes les fonctionnalités sont accessibles à partir de l’objet MediaPlayer.

En effet, plusieurs méthodes de cette classe nous seront très utiles :

[Image : entete\_function]

[Image : play\_pause] // *Pour lancer la musique et la mettre en pause*

[Image : entete\_function]

[Image : setDataSource] // *Pour préciser le fichier à utiliser*

### Implémentation

1er étape : création du MediaPlayer

Pour utiliser le Media Player, il faut dans un premier temps instancier un objet MediaPlayer de cette manière :

*MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.testsong);*

Ou les paramètres représentent respectivement:

* L’activity c’est-à-dire l’application elle-même (dans notre cas) qui contiendra le MediaPlayer
* Le fichier à utiliser (.mp3 par exemple)

2ème étape : implémentation des fonctionnalités de base

Puis implémenter toutes les fonctions nécessaires via des listeners appropriés.

Un listener est une interface Java permettant de détecter un évènement précis (sur le click, sur le passage d’une souris, etc …).

Par exemple, utiliser un listener sur l’évenement OnClick pour pouvoir faire les traitements appropriés lorsque l’on clique sur le bouton « Play » :

*buttonPlayStop.setOnClickListener(new OnClickListener()*

*{*

*@Override public void onClick(View v)*

*{*

*buttonClick(); // ici les instructions lorsque l’on clique sur //le bouton play*

*}*

*});*

3ème étape : l’interface graphique

Pour terminer il ne reste donc qu’à lier le MediaPlayer à une interface adapté, comme vu précédemment.

## Boucler dans un morceau

### Objectif

Le but de cette fonctionnalité est de pouvoir répéter une partie précise d’une chanson.

Par exemple : de pouvoir écouter la musique de la minute deux, à la minute trois en boucle.

Autrement dit, l’utilisateur sera donc capable de sélectionner un morceau de la chansons à répéter.

### Fonctionnement

Tout d’abord, il faut savoir qu’une boucle est caractérisée par un début, et une fin. C’est-à-dire le moment ou la boucle démarre et où elle termine.

Pour pouvoir boucler dans un morceau, il est naturellement indispensable de pouvoir connaitre la progression de la musique dans le temps. Par exemple de savoir qu’à un temps t, le lecteur a joué deux minutes et trois secondes de la chanson.

Pour cela, la classe MediaPlayer vu précédemment possède les méthodes :

[Image : entete\_function]

[Image : getCurrentPosition] // Permet de connaitre la progession de la musique à un instant précis.

[Image : seekPosition] // Permet de repositionner la lecture à un point précis.

L’algorithme ici est assez trivial.

En effet, si la progression de la musique a dépassé la fin de la boucle. Alors nous ramenons la lecture de la chanson au début de la boucle. Puis on recommence si la fonction boucle est toujours activé.

### Implémentation

Pour détecter si la progression de la chanson a dépassé la fin de la boucle par exemple, il nécessaire de faire cette vérification dans un thread.

Un thread désigne un point d'exécution dans le programme. Et donc faire du multi-thread, revient à exécuter du code à plusieurs endroits de façon indépendante.

Cependant un problème se pose, le thread est bloqué par android toutes les 0,5 secondes par exemple. En effet, android peut bloquer un thread pour éviter qu’il ne soit trop gourmand en ressource.

De ce fait, cela désynchronise notre vérification.

Par exemple, notre algorithme vas détecter que la progression de la musique a dépassé la fin de la boucle 5 secondes trop tard, du fait qu’au moment du dépassement, le thread était bloqué par le système.

[Image : threadBloked] // message d’erreur du thread bloqué

Pour pallier à ce problème, nous avons trouvé un moyen permettant d’ajouter l’exécution du thread dans une file d’attente, et de lancer cette exécution toutes les n millisecondes.

De ce fait, la désynchronisation devient négligeable.

**public** **void** run() // Thread

{

**if**(isLoop)// Si la fonctionnalité boucle est activé

{

//verification, si la progression de la musique a depassé la fin de la boucle

**if**(*mPlayer*.getCurrentPosition() >= seekBar\_fin.getProgress())

{

*mPlayer*.seekTo(seekBar\_debut.getProgress());

// alors la lecture de la musique est ramené au début de la boucle.

}

}

mHandler.postDelayed(**this**, 1000); // lance l’éxécution du thread

// toutes les 1000 milli-secondes

}

Donc pour reprendre notre exemple, maintenant notre algorithme vas détecter que la progression de la musique a dépassé la fin de la boucle quelques milli-secondes trop tard. Ce qui est invisible aux yeux de l’utilisateur.

## Sauvegarde et chargement d’une boucle

### Objectif

Il nous a été également demandé de pouvoir sauvegarder et charger une « boucle ».

C’est-à-dire de pouvoir sauvegarder le début et la fin d’un boucle pour une chanson précise, puis de pouvoir la charger ce qui évite de re-paramétrer une boucle manuellement.

### Fonctionnement

Le moyen que nous avons choisi pour sauvegarder une boucle, est d’enregistrer sa chanson associé, son commencement, ainsi que sa fin dans un fichier XML.

[Image : paramLoopXML]

Ensuite pour charger une boucle, il suffit de parcourir le fichier XML, et de récupérer les informations nécessaires au paramétrage d’une boucle.