Partie 6

Utiliser le son en actionScript 3 (P.O.O).

Du contrôle basique à la création d'un lecteur audio.

La lecture d'un son dans les applications flash est relativement simple. Il suffit pour cela de poser sur la scène l'occurrence d'un son importé dans la bibliothèque (wav, mp3). Très pratique pour des sons courts et légers, comme ceux pouvant accompagner un click, par exemple. En revanche les manipulations plus avancées que la simple lecture d'un son font très vite appel à des connaissances en actionScript.
Afin de poursuivre dans notre apprentissage. Je vous propose de mettre en place un lecteur musical complet en mesure de lire une liste de pistes référencées dans un xml. C'est un travail relativement long mais nécessaire pour avoir conscience d'une mise en place d'application dans une situation de production

Comme la vidéo, l'insertion d'un son d'une durée importante, et d'une qualité décente peut devenir très handicapant quant au poids final de l'animation. Lors de la compilation flash échantillonne le son en mp3 à 96 Kbps ce qui en terme de qualité procède d'une grande dégradation de la qualité sonore. Dès que l'on augmente cette qualité à un niveau plus élevé dans les paramètres, l'animation prend du poids.
Nous allons dans ce chapitre résoudre les questions basiques posées par la manipulation d'un son (positionnement de la tête de lecture, arrêt, pause) et son chargement par le biais d'un fichier externe.

La lecture et le contrôle d'un son sont un peu similaire aux processus en jeu dans la vidéo. L'objectif de cet atelier et de créer un petit lecteur musical en mesure de diffuser un titre, de le pauser, et de le reprendre. Et d'ajouter un contrôle pour le volume.
La difficulté principale de cet atelier sera de créer un système qui offrira une prise en charge complète des contrôles et d'un chargement mp3.

Commençons par l'essentiel à savoir la manipulation du son lui-même. Pour cela nous chargeons un mp3 dans la bibliothèque d'un fichier flash. Et dans le panneau de liaison nous activons l'exportation pour actionScript. (le nom de liaison donné ici est "Death_Watch").

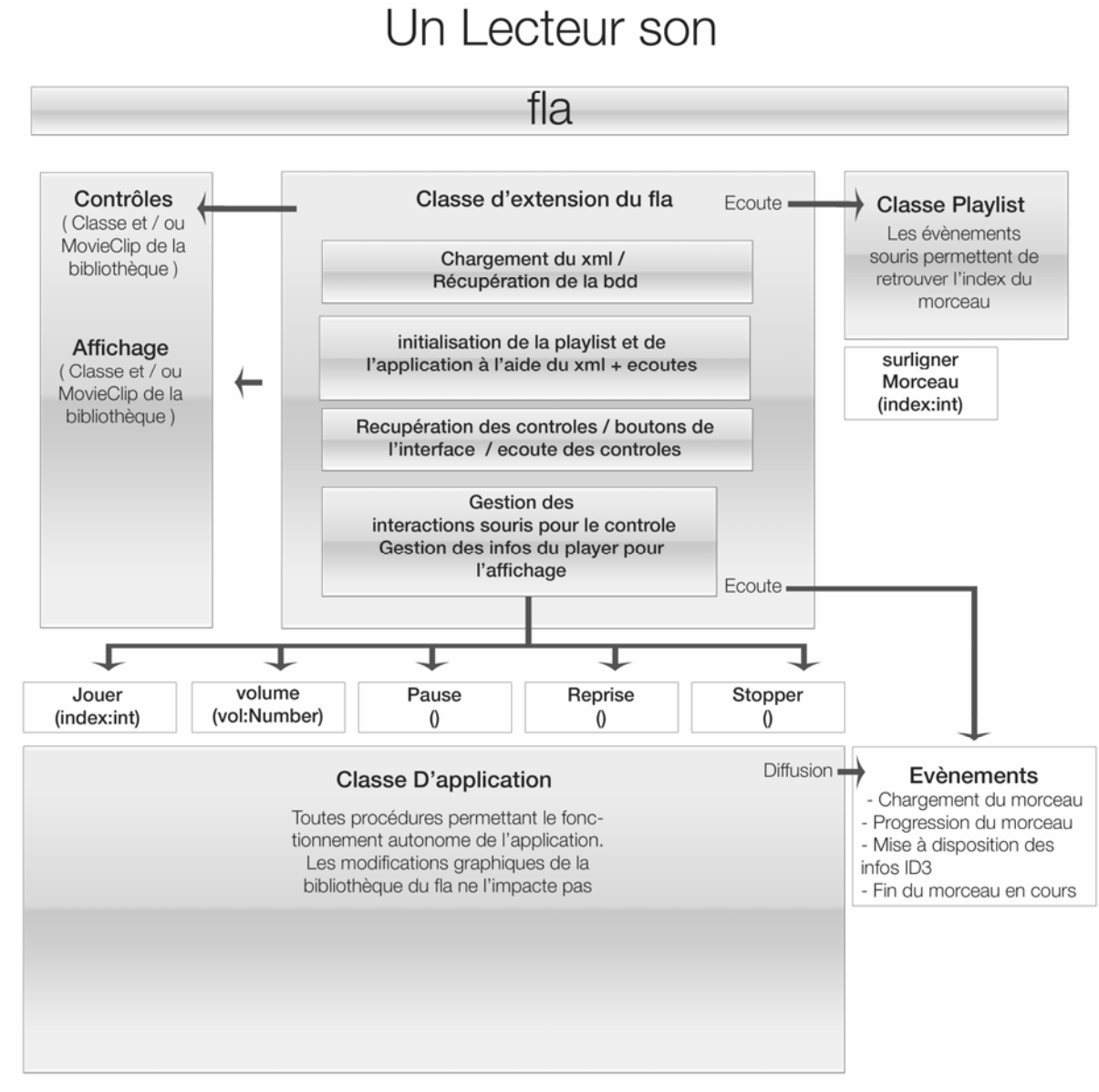
Pour commencer la lecture d'un son en programmation ces deux lignes suffisent :

```
// instanciation du son de la bibliothèque. Le type d'objet est Sound.
var son:Sound=new Death_Watch();
// Nous demandons à l'instance de commencer à jouer à l'aide
// de la methode play() de la classe Sound
var canalson:SoundChannel= son.play();
```

Remarquons que lors de l'appel de la méthode play(), nous avons pris soin de poser une variable recueillant l'objet renvoyé par la méthode. Cette variable est du type SoundChannel, que l'on peut traduire par canal de diffusion. C'est cet objet qui va nous permettre d'agir sur le volume, de stopper la progression du morceau, ou en lui adjoignant un écouteur, d'être averti de la fin de la diffusion. Il est donc extrêmement important de recueillir ce paramètre.

Nous avions vu dans la lecture d'une vidéo, que nous contrôlions la lecture et le volume par l'objet NetStream.
En somme, s'il nous faut transposer, il faut voir l'objet Sound comme le support de l'enregistrement, et l'objet SoundChannel comme l'amplificateur et la platine cd dont il nous reste à construire la façade et ses contrôles.

Organisation de l'application



Cette application doit contenir la gestion du chargement et de la diffusion d'une série de fichiers mp3. Ce type de gestion étant fréquent nous allons nous attacher à créer une classe qui ne sera pas liée aux aspects graphiques ou de fonctionnement général (par quel biais arrivent les infos ? présentation…). Ces aspects seront eux, déployés sur la classe principale.. Ainsi il sera possible d'utiliser cette application en standalone (player)

ou en se servant du cœur au sein d'une application plus complexe. Nous allons déployer trois classes différentes pour son fonctionnement.

1 la classe principale : LecteurSonIDE.

Notre classe principale est attachée à un fichier fla. Cela va nous permettre de nous faciliter la tache pour la gestion des graphiques et des placements.
Nous avons l'opportunité de placer des éléments sur la scène et d'utiliser leur noms d'occurrences comme si il s'agissait de variables de classes.
Pour le bon fonctionnement de l'application nous utiliserons une structure attendant que les éléments soient initialisés pour se déployer. Il est d'usage pour les langages asynchrones d'employer cette technique (cf : javascript). C'est particulièrement important ici car nous avons des éléments déployés sur la scène.
Le point de départ du déploiement sera dont la fonction init() et non la fonction constructeur.

```
package
{
	import flash.display.MovieClip;
	import flash.events.Event;

	public class LecteurSonIDE extends MovieClip
	{
		public function LecteurSonIDE()
		{
			if(stage)
			addEventListener(Event.ADDED_TO_STAGE, init);
			else
			init()
		}

		private function init(e:Event = null):void
		{
			 //point d'entrée de l'application
		}
}}
```

A – Chargement des données xml

Partant de la nous allons charger maintenant la liste XML. Nous allons la structurer de manière à pouvoir ajouter des éléments par la suite, comme la personnalisation des couleurs de l'application.

```
<data>
	<playlist>
		<morceau genre="rock"><titre>Je tuerai la
pianiste</titre><artiste>Bashung</artiste><fichier>medias/Death_Watch.mp3</
fichier></morceau>
		<morceau
genre="classique"><titre>Classique</titre><artiste>Debussy</artiste><fichie
r>medias/Pigs.mp3</fichier></morceau>
		<morceau
genre="jazz"><titre>pianiste</titre><artiste>Bashung</artiste><fichier>medi
as/Mixdown_Nes.mp3</fichier></morceau>
	</playlist>
</data>
```

L'adresse du xml peut être stockées par défaut sur la classe principale. Mais il serait bien plus interessant de la stocker dans la page html. De cette manière le lecteur pourrait diffuser différentes playlist sur un seul et même site ! Nous placerons donc notre adresse dans une flashVar. Nous laisserons malgré cela l'adresse par défaut dans la classe principale pour pouvoir tester l'animation depuis le logiciel.

Nous allons utiliser un stockage des informations un peu particulier sous forme d'une classe personnalisée. Chaque info sera stockée dans un objet au sein d'une liste. Nous plaçons cette liste dans nos variables de classes. Nous stockons aussi le xml chargé non pour la listedes morceaux mais pour les paramètres de configuration qu'il est possible de lui adjoindre..

```
// Liste des Objets BadgeSon
private var tab_liste:Array = new Array();
// stockage des infos xml (optionnel)
private var donnees_xml:XML;


private function init(e:Event = null):void
{

	removeEventListener(Event.ADDED_TO_STAGE, init);

	// recherche de la playlist
	if (loaderInfo.parameters.liste != undefined)
	{
		// loaderInfo permet l'accès aux flashvars chargées
		//depuis la page html (identifiant : liste)
		chargementXML(loaderInfo.parameters.liste)
	}
	else
	{
		chargementXML("xml/playlist.xml")
	}
}


public function chargementXML(url:String) : void
		{
	var loader:URLLoader = new URLLoader();
	var requete :URLRequest = new URLRequest(url);
	loader.addEventListener(Event.COMPLETE, XMLCharge);
	loader.load(requete);
}

// Le cœur de l'application peut être déployé ici
private function XMLCharge(even:Event):void
{


	var loader:URLLoader = even.target as URLLoader;

	var donnees_xml:XML = new XML(loader.data);
	XML.prettyIndent = 0;
	XML.ignoreWhitespace = true;
}
```

2 – La classe BadgeSon

Charger un ensemble de données via un xml est certes très pratique mais qui peut singulièrement compliquer chaque modification dès lors ou l'on modifie la structure. L'idée est de standardiser les informations à l'aide d'un objet de telle manière à ce que chaque manipulation du xml n'impacte que la boucle de remplissage de la liste. Tout accès aux informations délivrées par l'objet se fait de manière unique quelle que soit l'application ou il est utilisé. Nous détachons donc le contenu de la manière de le délivrer ce qui est la méthode la plus viable pour conserver un maximum de portabilité.
L'intérêt dans cette méthode est sa souplesse. En effet les fichiers mp3 sont en principes dotés d'informations sur le morceau codés sous forme d'un objet appelé ID3. L'objet aura la capacité de compléter ses informations grâce à la récupération de cet objet durant le chargement. Ce qui permettra d'afficher les informations complètes du morceau si elles sont stockées (artiste, titre, genre, album, année).
Les développements ultérieurs de cette classe pourraient charger les étiquettes id 3 des fichier de manière autonome ce qui pourrait permettre de nous passer des champs concernés dans le xml.
Mieux nous pourrons récupérer ces informations sous forme d'une chaîne de caractère prête à être affichée !

```
package
{
	import flash.media.ID3Info;

	public dynamic class BadgeSon
	{
		private var tagID3 : ID3Info;
		public var artiste:String = "artiste inconnu";
		public var titre:String = "sans titre";
		public var annee:String ="";
		public var genre:String="Pas de genre";
		public var album : String= "pas d'infos";
		public var index:int;
		public var fichier:String;
		private var id3ToString:String;
		public function BadgeSon()
		{}

		public function set ID3 (tag:ID3Info):void
		{
			//trace("annee : ["+tagID3.artist.toString()+"]")
			tagID3 = tag;
			var desc : String = new String();
			if (tagID3.artist != null)
			{
				artiste = tagID3.artist;
				desc += tagID3.artist +"  -  ";
			}
			if (tagID3.songName != null)
			{
				titre = tagID3.songName;
				desc += tagID3.songName+" \n";
			}
			if (tag.album != null)
			{
				album = tagID3.album;
				desc += "Album : " + tagID3.album+" ";
			}
			if (tagID3.year != null)
```

```
                {
                        annee = tagID3.year;
                        desc += "( " + tagID3.year+" ) \n";
                }
                if (tagID3.genre != null)
                {
                        genre = tagID3.genre;
                        desc += "Genre : " + tagID3.genre;
                }

                id3ToString = desc;
        }

        // récuperation d'une chaine affichable.
        public function get description():String {return id3ToString;}

    }

}
```

Nous voyons au passage l'intérêt d'un type de fonction particulier appelé getter/setter. Une variable privée liée à ce type de fonction permet d'une part de faire appel à cette fonction comme si il s'agissait d'un variable. La seconde mais plus intéressante qualité de cette fonction est d'être en mesure d'effectuer des opérations subalternes ce qui n'est bien sur pas possible si l'on met en place une simple variable.
Dans notre classe, son utilisation pour l'étiquette id3 permet au passage de rafraîchir les variables si les infos correspondante sont présentes et aussi de constituer une chaîne préformatée affichable au moyen de la méthode text d'un TextField.
Nous ajoutons alors la boucle de construction a notre fonction de réception du xml.

```
// L'application est deployée ici
private function XMLCharge(even:Event):void
{

    var loader:URLLoader = even.target as URLLoader;
    donnees_xml = new XML(loader.data);
    // ces proprietés semblent être obsoletes dans le fp10
    //XML.prettyIndent = 0;
    //XML.ignoreWhitespace = true;

    //standardisation des infos xml au moyen d'un objet personnalisé
BadgeSon

    for (var i :int; i < donnees_xml.playlist.morceau.length();i++ )
    {
        var badge:BadgeSon = new BadgeSon();
        badge.index = i;
        badge.titre = donnees_xml.playlist.morceau[i].titre;
        badge.artiste = donnees_xml.playlist.morceau[i].artiste;
        badge.fichier = donnees_xml.playlist.morceau[i].fichier;
        tab_liste.push(badge);
    }
}
```

Nous reviendrons sur notre classe principale pour connecter l'ensemble des comportements pour le moment nous allons nous pencher sur l'archistcture de notre classe PisteSon qui va se charger de toute les opérations de chargement et de diffusion des fichiers.

3 - la classe « cœur » : PisteSon.

Voyons dans la documentation actionScript Les possiblités que nous avons de manipuler le son à l'aide d'actionScript.

Documentation (dictionnaire de référence actionscript)

Sound : L'objet Sound contient les méthodes qui vont nous permettre de charger le son à partir d'un fichier externe. Le passer en lecture. L'objet diffuse aussi les évènements de chargement, de réception des informations id3 du fichier.

SoundTransform : L'objet SoundTransform agit sur le volume et la balance du son.

SoundMixer : Objet par le biais duquel il est possible de modifier les propriétés générales du son dans flash. Il contient un objet SoundTransform valant pour le volume et la balance générale du swf. Pour les niveaux plus avancés, cet objet peut nous fournir un objet à partir duquel il est possible de construire un analyseur de spectre et sa représentation graphique.

SoundChannel : L'objet SoundChannel et le plus important en ce qui concerne le comportement d'un son durant la lecture puisque c'est lui qui contient l'objet SoundTransform et qui permet l'arrêt de la lecture. Il envoie également l'évènement qui prévient de la fin de la lecture du fichier.

Nous voyons que plusieurs objets sont susceptibles de délivrer des évènements. L'intérêt de notre classe sera d'unifier la diffusion au travers d'un seul objet, et d'assurer la gestion des différents processus pour que nous puissions l'utiliser de manière simple.
Le son n'est pas un objet d'affichage il n'est donc pas nécessaire qu'elle étende MovieClip ou Sprite. En revanche l'implantation d'un diffuseur qui lui serait attaché nous simplifierait les choses. Nous allons donc faire hériter cette classe d'EventDispatcher. Nous aurons ainsi la possibilité d'émettre ou de rediriger des événements via le diffuseur intégré.

Cahier des charges et choix de la stratégie de programmation.

Le modèle de classe que nous allons utiliser ici sera en mesure de diffuser plusieurs son à la suite. Ainsi nous n'aurons qu'a instancier une seule fois cette clase pour obtenir une piste de lecture en mesure de diffuser tout les morceau de la liste. Cette stratégie pourra ainsi nous permettre de l'utiliser au sein d'une application multipiste. Par exemple il sera possible de ce fait de proposer en option le volume sonore de la musique de fond séparément des bruitages.

Chargement du son et mise en lecture

Le chargement du son en streaming utilise les mêmes évènements de base que celui d'une image. Nous pourrions directement utiliser nos objet badge son pour charger les fichier et stocker les informations de chargement. Mais cela reviendrait à rendre notre classe moins souple pour quelques lignes de fonctionnement. Nous choisissons donc de charger les fichier son grâce à une url et de confier la synchronisation des infos à la classe principale.

On remarquera que les retraits d'écouteurs sont susceptibles d'être employés plusieurs fois. On les place dans un méthode qui va nous faire gagner quelques lignes de code.

```
package
{
	import flash.events.Event;
	import flash.events.EventDispatcher;
	import flash.events.IOErrorEvent;
	import flash.events.ProgressEvent;
	import flash.media.ID3Info;
	import flash.media.Sound;
	import flash.media.SoundChannel;
	import flash.media.SoundLoaderContext;
	import flash.media.SoundTransform;
	import flash.net.URLRequest;
	/**
	public class PisteSon extends EventDispatcher
	{

		private var son:Sound;
		private var canal_son:SoundChannel;
		private var position_lecture : Number = 0;
		private var volume_son : Number = 1;
		public var ID3 :ID3Info;
		public var longueur_morceau:Number=0;
		public var index_piste:int = 0;

		public function PisteSon()
		{
		}

	// ------------- CHARGEMENT SON EXTERNE ------------- //


public function chargerNouveauSon (pfichier : String , tamponLecture : int
= 5000):void
{
	// instanciation d'un objet Sound
	son = new Sound();

	var contexteAudio : SoundLoaderContext = new SoundLoaderContext
(tamponLecture);

	// mise en place des divers écouteurs permettant d'établir les
	// comportements adequats

	son.addEventListener( ProgressEvent.PROGRESS, chargementEnCours );
	son.addEventListener( Event.COMPLETE, retraitEcouteursSon);
	son.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );
	son.addEventListener(Event.ID3, id3Detecte);

	// chargement dynamique du son
	var requete : URLRequest = new URLRequest (pfichier) ;
	son.load (requete, contexteAudio);
}

public function id3Detecte(even:Event):void
{
	trace("Detection ID3");
	son.removeEventListener(Event.ID3, id3Detecte);
	ID3 = son.id3;
```

```
		dispatchEvent(even);
}

public function chargementEnCours ( pEvt:ProgressEvent ):void
{
		dispatchEvent(pEvt);
		// la longueur du morceau correspond au bytes chargés
		longueur_morceau = son.length;
}

private function retraitEcouteursSon(even:Event = null):void
{
		if (son.hasEventListener(Event.COMPLETE))
		{
		son.removeEventListener( ProgressEvent.PROGRESS, chargementEnCours );
		son. removeEventListener ( Event.COMPLETE, retraitEcouteursSon);
		son. removeEventListener ( IOErrorEvent.IO_ERROR, erreurChargement );
		}
		if (son.hasEventListener(Event.ID3))
		{
			son.removeEventListener (Event.ID3, id3Detecte);
		}
}

private function erreurChargement (even : IOErrorEvent):void
{
		retraitEcouteursSon();
		dispatchEvent(even);
}

}

}
```

Nous avons installé les processus de chargement. Il nous reste maintenant à mettre le son qui se charge en lecture.
Nous allons ensuite modifier ces fonctions pour avoir la possibilité de charger plusieurs morceaux à la suite.

La fonction jouerSon

La methode est critique dans notre classe car son fonctionnement est le seul moyen de placer la tête de lecture à un endroit donné de la piste. La fonction Play() de l'objet son renvoie aussi un objet SoundChannel qui lui aussi est important pour le fonctionnement de la classe. Il reseinge sur la lecture en cours, on lui applique le volume, et enfin il émet un évènement lorsque la piste est en fin de lecture. Pour cela cet objet est placé sous une variable de classe. Nous voyons que chaque fois que nous faisons appel à la fonction play(), un nouvel objet SoundChannel est généré, le morceau et le canal de diffusion sont donc réinitialisés, ainsi que le volume.
Ce que nous devons remarquer en premier c'est la vérification de l'existence d'un SoundChannel (canalson). Si nous ne mettons pas en place ce comportement. Chaque fois que cette fonction sera invoquée pendant la lecture d'un son cela génèrera un nouveau canal son et le son se surajoutera au précédent puisque flash à des capacité multipistes. Ainsi nous vérifions que l'objet existe, pour ne pas émettre d'erreur au premier lancement, et nous coupons la diffusion du son en prévision de sa toute prochaine remise en route. Cela permet aussi d'éviter de mauvaise surprise au texte de debbugage : que se passerait-il si l'utilisateur appuie plusieurs fois sur le bouton play ?

Nous commençons à écrire notre méthode en considérant à présent qu'un son est peut être déjà en cours de diffusion. Nous commençons donc par vérifier si un canal de diffusion est ouvert. Si c'est la cas nous vérifions s'il à un écouteur d'évènement de fin de lecture et nous le retirons le cas échéant. Enfin on coupe la lecture du canal.
Nous appliquons ensuite les processus de mise en lecture. Récupération du nouveau canal de diffusion à l'appel de la fonction play(). La fonction prend un paramètre pour le positionnement de la tête de lecture. Nous la relions à une variable de classe qui permettra les réglages depuis la classe principale. Puis nous lui (ré) attribuons le volume sonore en cours (sinon la piste part au volume maximum).

```
public function jouerSon ():void
{
	if(canal_son != null)
	{
	if (canal_son.hasEventListener(Event.SOUND_COMPLETE));
	canal_son.removeEventListener(Event.SOUND_COMPLETE, finLecture);
	canal_son.stop()
	}

	canal_son = son.play(position_lecture);
	canal_son.addEventListener(Event.SOUND_COMPLETE, finLecture);
	volume = volume_son;
}
```

Processus attachés à la fonction jouerSon

- Processus de (re)positionnement de la tête de lecture.

Dans la fonction jouerSon nous utilisons la variable *position_lecture* afin de placer la tête de lecture à l'endroit choisi. Nous allons avoir besoin de manipuler cette variable pour la naviagtion, la pause ou l'arret ou de la consulter pour l'affichage du témoin de progression.
Nous choisissons à nouveau un getter setter pour cette fois-ci s'assurer que la fonction renverra toujours une valeur de type number utilisable pour un processus automatique. Pour cela on verifie l'existence d'un canal de diffusion sinon on renvoie la valeur 0.

```
public function get position():Number
{
	if (canal_son != null)
	var pos:Number = canal_son.position;
	else
	pos = 0;
	//trace("POSITION : " + pos);
	return pos
}

public function set position(pos:Number):void  {position_lecture  = pos;}
```

- Diffusion de l'évènement de fin de lecture

L'évènement SOUND_COMPLETE est relié à une simple fonction de redirection. De cette manière nous n'aurons qu'un écouteur permanent dans la classe principale. Dont la fonction se chargera de lancer la piste suivante. Nous en profitons pour couper la connection au serveur afin de rendre l'objet éligible à la destruction.

```
private function finLecture(even:Event):void
{
	dispatchEvent(even);
}
```

- Adaptation de la function de chargement à la lecture multiple

Nous reprenons maintenant la function de chargement pour lui adjoindre les processus de retraits et de verification pour une diffusion multiple.
:

```
public function chargerNouveauSon (pfichier : String , tamponLecture : int
= 5000):void
{
	if(son != null)
{
	// remise à Zero
	position_lecture = 0;
	canal_son.stop();

// nous devons verifier l'existence des écouteur dans le cas ou
// l'utilisateur a actionné les boutons précédents et suivants

if (canal_son.hasEventListener(Event.SOUND_COMPLETE));
{canal_son.removeEventListener(Event.SOUND_COMPLETE, finLecture);}
retraitEcouteursSon();
}
// instanciation d'un objet Sound
son = new Sound();

var contexteAudio : SoundLoaderContext = new SoundLoaderContext
(tamponLecture);

	// mise en place des divers écouteurs permettant d'établir les
	// comportements adequats

	son.addEventListener( ProgressEvent.PROGRESS, chargementEnCours );
	son.addEventListener( Event.COMPLETE, retraitEcouteursSon);
	son.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );
	son.addEventListener(Event.ID3, id3Detecte);

	// chargement dynamique du son
	var requete : URLRequest = new URLRequest (pfichier) ;
	son.load (requete, contexteAudio);
}
```

Manipulation du son.

volume

Maintenant voyons comment nous pouvons agir sur le volume. Comme la plupart des propriétés exprimant une proportion dans flash, le volume est une valeur décimale comprise entre 0 et 1.

Un processus un peu particulier est à remarquer ici : le fait de modifier l'objet SoundTransform ne modifie pas directement le volume. Il faut, pour que la transformation s'opère, réattribuer l'objet modifié SoundTransform à la propriété idoine de l'objet SoundChannel :

```
public function set volume(pvolume:Number):void
{
	var transformation : SoundTransform = canal_son.soundTransform;
	transformation.volume = pvolume;
	canal_son.soundTransform = transformation;
}
```

Nous pourrons lancer la méthode simplement à l'aide d'un curseur ou un bouton de volume qui indiquerait le niveau entre 0 et 1. Dans n'importe quelle animation ou diffusion de vidéo, elle pourra nous servir de support pour un bouton « mute ». (ndr : Le son d'une vidéo est codé par le même objet SoundTransform.).

Arrêt, Pause, Avancer reculer dans la piste.

Il n'existe pas à proprement parler de fonction de recherche sur la piste ou de pause. Tous ces comportements peuvent malgré cela être programmés en utilisant et en stockant la propriété position de l'objet SoundChannel , puis de relancer la lecture avec la nouvelle position ou de reprendre à l'endroit ou elle à été pausée.
La propriété position transmet le temps écoulé en millisecondes par rapport au début du son. Mis en proportion avec la propriété « length » de l'objet Sound, il devient possible de transmettre une proportion de la position d'un curseur et de la traduire en un point temporel sur lequel caler la tête de lecture.

Pauser / lancer la lecture .

La pause, du fait de l'architecture de flash, consistera à stopper le morceau en diffusion en prenant soin de stocker sa position au moment de l'action. Remarquons que la méthode stop() se trouve sur l'objet SoundChannel et non l'objet Sound() comme on pourrait logiquement le croire.

```
public function pauserSon():void
{
	if(canal_son != null)
	{
		position_lecture = canal_son.position;
		canal_son.stop();
	}
}
```

Il suffit alors de laisser la fonction jouerSon () reprendre la lecture à la position stockée dans notre variable.

Arrêt Complet

Un morceau une fois stoppé avec la méthode stop() de l'objet SoundChannel n'est pas fermé tant que la connexion n'est pas fermée. Ceci se fait en utilisant la méthode close() liée à la

classe Sound. Cette instruction n'est utile que si le fichier est arrêté alors que le chargement n'est pas terminé. Cette methode provoque une erreur à l'exécution si on y fait appel alors que le chargement du morceau à été complété.
De fait, Nous n'allons pas utiliser ce processus dans notre application. De plus le bouton stop sera une simple remise à zéro de la position de le tête de lecture son et l'arrêt de la diffusion. De ce fait l'utilisateur pourra utiliser la fonction play pour reprendre la lecture depuis la dernière plage lue :

```
public function stopperSon():void
{
	if(canal_son != null)
	{
		position_lecture = 0;
		canal_son.stop();
		if (canal_son.hasEventListener(Event.SOUND_COMPLETE));
		canal_son.removeEventListener(Event.SOUND_COMPLETE,
finLecture);
	}
}
```

Notre classe est maintenant fonctionnelle. Nous allons tout de même lui adjoindre un fonction de confort pour l'affichage du temps écoulé et total. L'affichage en millisecondes n'est sans doute pas du meilleur effet et il nous faut trouver un moyen de présenter les temps sous la forme commune MM : SS.

```
public function formaterDuree(nb_millisecondes:Number):String
{
	var tps : Date = new Date();
	tps.setHours(null, null, null, nb_millisecondes);
	var min:String = tps.getMinutes().toString();
	var sec:String = tps.getSeconds().toString();

	if(tps.getMinutes() < 10)
	{
		min = "0"+ tps.getMinutes();
	}
	if(tps.getSeconds() < 10)
	{
		sec = "0"+ tps.getSeconds();
	}
	var temps_formate:String= min+":"+sec;

	return temps_formate;

}
```

Voici l'ensemble de la classe dotées des fonctions que nous avons détaillé :

```
package
{
import flash.events.Event;
import flash.events.EventDispatcher;
import flash.events.IOErrorEvent;
import flash.events.ProgressEvent;
import flash.media.ID3Info;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundLoaderContext;
import flash.media.SoundTransform;
```

```
import flash.net.URLRequest;

public class PisteSon extends EventDispatcher
{

private var son:Sound;
private var canal_son:SoundChannel;
private var position_lecture : Number = 0;
private var volume_son : Number = 1;
public var ID3 :ID3Info;
public var longueur_morceau:Number=0;
public var index_piste:int = 0;

public function PisteSon()
{
}

public function pauserSon():void
{
      if(canal_son != null)
      {
            position_lecture = canal_son.position;
            canal_son.stop();
      }
}

public function stopperSon():void
{
      if(canal_son != null)
      {
            position_lecture = 0;
            canal_son.stop();
            if (canal_son.hasEventListener(Event.SOUND_COMPLETE));
            canal_son.removeEventListener(Event.SOUND_COMPLETE,
finLecture);
      }
}

public function get position():Number
{
      if (canal_son != null)
      var pos:Number = canal_son.position;
      else
      pos = 0;
      //trace("POSITION : " + pos);
      return pos
}

public function set position(pos:Number):void
{
      position_lecture  = pos;
}

public function jouerSon ():void
{
      if(canal_son != null)
      {
            if (canal_son.hasEventListener(Event.SOUND_COMPLETE));
            canal_son.removeEventListener(Event.SOUND_COMPLETE,
finLecture);
            canal_son.stop()
```

```
		}

		canal_son = son.play(position_lecture);
		canal_son.addEventListener(Event.SOUND_COMPLETE, finLecture);
		volume = volume_son;
	}

	private function finLecture(even:Event):void
	{
		dispatchEvent(even);
	}

	public function set volume( pvolume:Number):void
	{
		// mise en mémoire du volume pour l'appliquer de piste en piste.
		volume_son = pvolume;

		if (canal_son != null)
		{
			var transfo_son: SoundTransform = canal_son.soundTransform;
			transfo_son.volume = pvolume;
			canal_son.soundTransform = transfo_son;
		}

	}

	// -------------------- CHARGEMENT SON EXTERNE ---------------------- //

	public function chargerNouveauSon (pfichier : String , tamponLecture : int
	= 5000):void
	{
		if(son != null)
		{
			// remise à Zero
			position_lecture = 0;
			canal_son.stop();

	// nous devons verifier l'existence des écouteur dans le cas ou
	// l'utilisateur a actionné les boutons précédents et suivants
		if (canal_son.hasEventListener(Event.SOUND_COMPLETE));
			canal_son.removeEventListener(Event.SOUND_COMPLETE,
	finLecture);
			retraitEcouteursSon();
		}

		// instanciation d'un objet Sound
		son = new Sound();

		var contexteAudio : SoundLoaderContext = new SoundLoaderContext
	(tamponLecture);

		// mise en place des divers écouteurs permettant d'établir les
		// comportements adequats

		son.addEventListener( ProgressEvent.PROGRESS, chargementEnCours );
		son.addEventListener( Event.COMPLETE, retraitEcouteursSon);
		son.addEventListener( IOErrorEvent.IO_ERROR, erreurChargement );
		son.addEventListener(Event.ID3, id3Detecte);


		// chargement dynamique du son
```

```
		var requete : URLRequest = new URLRequest (pfichier) ;

		son.load (requete, contexteAudio);
		jouerSon();
}

public function id3Detecte(even:Event):void
{
		trace("Detection ID3");
		son.removeEventListener(Event.ID3, id3Detecte);
		ID3 = son.id3;
		dispatchEvent(even);
}

public function chargementEnCours ( pEvt:ProgressEvent ):void
{
		dispatchEvent(pEvt);
		longueur_morceau = son.length;
}

private function retraitEcouteursSon(even:Event = null):void
{
		if (son.hasEventListener(Event.COMPLETE))
		{
		son.removeEventListener( ProgressEvent.PROGRESS, chargementEnCours );
		son.removeEventListener( Event.COMPLETE, retraitEcouteursSon);
		son.removeEventListener( IOErrorEvent.IO_ERROR, erreurChargement );
		}
		if (son.hasEventListener(Event.ID3))
		{
			son. removeEventListener(Event.ID3, id3Detecte);
		}
}


private function erreurChargement (even : IOErrorEvent):void
{
		retraitEcouteursSon();
		dispatchEvent(even);
}


public function formaterDuree(nb_millisecondes:Number):String
{
		var tps : Date = new Date();
		tps.setHours(null, null, null, nb_millisecondes);
		var min:String = tps.getMinutes().toString();
		var sec:String = tps.getSeconds().toString();

		if(tps.getMinutes() < 10)
		{
			min = "0"+ tps.getMinutes();
		}
		if(tps.getSeconds() < 10)
		{
			sec = "0"+ tps.getSeconds();
		}
		var temps_formate:String= min+":"+sec;
		return temps_formate;

}
```

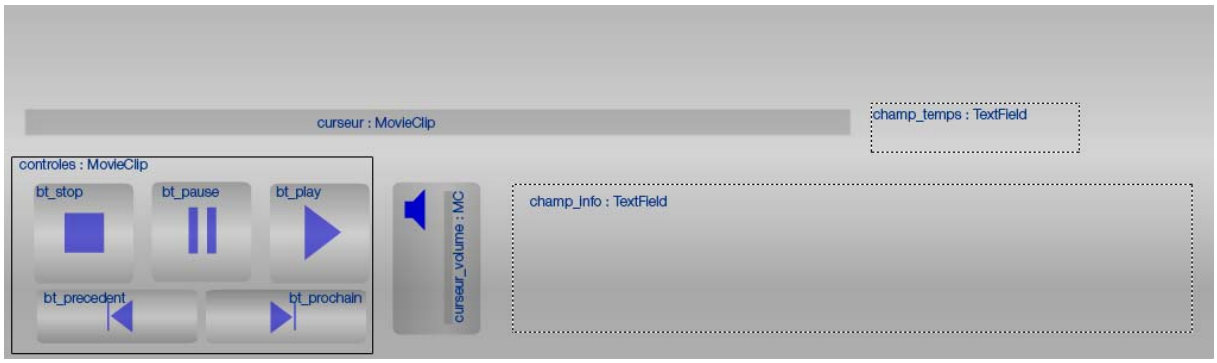
```
}

}
```

Nous disposons d'une classe de lecture de son personnalisée et suffisamment souple et simple pour être utilisé dans quasiment tous les contextes d'application. Il nous reste maintenant à gérer le déroulement de la lecture des pistes et l'utilisation des boutons de contrôles crées dans le fla.
Nous retournons donc dans notre classe principale pour relier notre classe aux divers contrôles que l'on souhaite mettre en place.

Ajout et écoute du lecteur sur la classe principale.

Voici les différents noms d'occurrence donnés aux éléments de la scène de mon application. Je peux y accéder directement dans ma classe principale. Ce qui me permet de ne pas me préoccuper du placement des différents élément. La seule contrainte de ce système est de veiller à ne modifier les tailles en passant par l'intérieur des clips concernés. De cette manière les paramètres d'échelle restent à 1.



Nous allons placer maintenant notre lecteur dans la classe principale ainsi que deux écouteurs pour le rafraîchissement des informations du morceau et celui du signal de fin de lecture. Nous aurons à l'issue de cela un départ de la lecture du premier morceau suivi des autres en boucle
Le point d'insertion de notre application s'est reporté de la fonction init() à la fonction de reception du xml. Logiquement puisque nous devons disposer des adresses pour lancer la lecture. C'est par conséquent la fonction xmlCharge qui va recevoir nos instructions de départ.
Le code qui suit ne reprend que les fonctions modifiées par ce comportement et les variables à ajouter.

```
package
{
…

public class LecteurSonIDE extends MovieClip
{
// Coeur de l'application contient toutes les fonction pour la manipulation
des pistes jouées
private var lecteur:PisteSon

// Liste des Objets BadgeSon
private var tab_liste:Array = new Array();
```

```
// stockage des infos xml (optionnel)
private var donnees_xml:XML;

// Premet la gestion de plages
private var morceau_en_cours:int = 0;

public function LecteurSonIDE()
{…}

private function init(e:Event = null):void
{
	removeEventListener(Event.ADDED_TO_STAGE, init);
	// l'objet est instancé dès la fonction init
	lecteur = new PisteSon();

	//reception des infos du fichier
	lecteur.addEventListener(Event.ID3, messagesPiste);
	lecteur.addEventListener(Event.SOUND_COMPLETE, jouerSuivant);

	…

}


private function jouerSuivant(even:Event = null):void
{
	trace(tab_liste[morceau_en_cours].fichier);
	if (morceau_en_cours < tab_liste.length -1)
	{
		morceau_en_cours++;

	lecteur.chargerNouveauSon(tab_liste[morceau_en_cours].fichier);
	}
	else if(morceau_en_cours == tab_liste.length -1)
	{
		morceau_en_cours = 0;

	lecteur.chargerNouveauSon(tab_liste[morceau_en_cours].fichier);
	}

}

// reception des infos ID3 et rafraichissement des badges
private function messagesPiste(even:Event):void
{
	tab_liste[morceau_en_cours].ID3 = lecteur.ID3;
	champ_info.text = tab_liste[morceau_en_cours].description;
	for (var v : * in lecteur.ID3)
		{
		trace(v+" : "+lecteur.ID3[v]);
		}
}

// ---------------------CHARGEMENT XML -------------------------//

public function chargementXML(url:String) : void
{…}

	// L'application est deployée ici
	private function XMLCharge(even:Event):void
	{
```

```
        …
        //standardisation des infos
        // xml au moyen d'un objet personnalisé BadgeSon

        for (var i :int; i < donnees_xml.playlist.morceau.length();i++ )
              {…}

              // Chargement du premier morceau
              lecteur.chargerNouveauSon(tab_liste[0].fichier);

              lecteur.jouerSon();
              // mise en route de l'horloge.
              horloge.start()
        }
}
}
```

Liaisons fla – classe principale – Classe PisteSon.

Parmi les éléments présents sur la scène de mon fichier fla, il en existe deux catégories :

- Les éléments finaux (boutons ply , pause, précédent…)

- Ceux dont nous allons nous servir pour créer des comportements supplémentaires et manipuler pendant l'execution du lecteur. Parmi ceux là on discerne les Textfield des éléments de type curseur. (« curseur » et curseur_volume) .

Nous allons créer l'affichage et l'interactivité de ces derniers dans notre classe principale. Leur présence sur la scène nous permet de gérer leur taille et emplacement, éventuellement de permettre quelques traitements graphiques accessoires tels que l'ajout de filtre sur ces occurrences.

Traitement graphique de la navigation et du volume

En créant un curseur qui renvoie une proportion de sa position par rapport à sa longueur totale, soit un decimal compris entre 0 et 1, nous pouvons facilement faire des modifications en ce servant de ces chiffres comme opérateur, on obtient alors une proportion du morceau en cours, et donc d'afficher la position de al tête de lecture et inversement de recaler notre morceau à un autre endroit : La fonction de navigation consistera simplement à appeler la fonction jouer Son du lecteur en modifiant les données du curseur en un ratio entre 0 et 1 de la durée du morceau.

Le code suivant va dessiner un rectangle représentant une barre de progression. Chaque clic de souris sur cette barre va dessiner une forme graphique en prenant la valeur de l'abscisse local de la souris dans le rectangle. (dessinerTemoin())
Cette fonction pourra être également utilisée pour la visualisation du curseur de volume

Je vous propose ici deux variantes de cette même fonction, la seconde desine une barre à l'aide de l'outil de dégradé :

```
function dessinerTemoin(largeur:Number, hauteur : Number, couleur : Number
= 0x0066FF,palpha : Number = 0.7):Sprite
{
      // création d'un objet doté des capacités de graphics
      var dessin:Sprite = new Sprite();
      // Création du remplissage selon les paramètres de la fonction
      dessin.graphics.beginFill(couleur,palpha);
      // création de la forme
      dessin.graphics.drawRoundRect(0,0,largeur,hauteur,3,3);
      // desactivation des actions souris
      dessin.mouseEnabled = false;
      return dessin;
}
```

La seconde crée un graphique doté d'un dégradé

```
private function dessinerTemoin(largeur:Number, hauteur:Number,
couleur_1:Number = 0x0066FF, couleur_2:Number = 0x0033CC):Sprite
{
      //on utilise un sprite pour l'ecriture de la forme
      var objet:Sprite = new Sprite();
      // creation d'une matrice de transformation pour le dégradé
      var matrix:Matrix = new Matrix();
      // Equivaut à l'outil de transformation de dégadé
      // La rotation se donne en radians
      matrix.createGradientBox(largeur, hauteur, 1.575);

      // création d'un remplissage en dégradé
      objet.graphics.beginGradientFill(GradientType.LINEAR,
                                       [couleur_1, couleur_2],
                                       [.8, .8],
                                       [65, 220],
                                       matrix,
                                       "pad",
                                       "rgb",
                                        1);
      // dessin de la forme
      objet.graphics.drawRoundRect(0, 0, largeur, hauteur,5,5);
      // retrait de la sensibilité souris
      objet.mouseEnabled=false;
      //renvoi dde la forme
      return objet;
}
```

Nous conservons la première version afin de ne pas alourdir par trop la compréhension du code. Cette même fonction va donc remplir trois rôles : Illustrer le chargement de la piste, celui de la progression du morceau et celui du curseur de volume. On voit ici tout l'intérêt de créer un fonction la plus générique possible.
Nous allons employer la même technique de fonctionnement de l'affichage pour ces comportements : Nous plaçons dans les clips réceptacles, un container de type sprite qui contiendra les formes crées par l'une des méthodes ci-dessus. La forme sera ôtée à chaque renouvellement. En ce qui concerne celui-ci nous allons relier la barre de chargement à l'écoute de celui-ci sur le lecteur. Nous allons avoir besoin également d'un timer et de son système évènementiel. pour assurer le rafraîchissement du témoin de progression.
Enfin nous allons écouter les click sur le témoin de progression pour la navigation dans la piste et sur le clip curseur_volume pour la gestion de celui-ci.

Nous ajoutons donc quatre variables de classe à celles déjà présentes dans lecteurSonIDE :

```
private var layer_lecture : Sprite = new Sprite();
private var layer_chargement : Sprite = new Sprite();
// Implanté dans le curseur_volume
private var layer_volume : Sprite = new Sprite();

// L'horloge premet de raffraichir la barre de progression
private var horloge:Timer = new Timer(100);
```

puis à la fin de la fonction init nous assemblons les éléments et les mettons en écoute :

```
private function init(e:Event = null):void
{
      // ajout d'une ecouteur de progression de chargement
      lecteur.addEventListener(ProgressEvent.PROGRESS, dessinerChargement);


                  ///… (code inchangé) ///

      // ajout des éléments au clip "curseur" de la scène
      curseur.addChild(layer_chargement);
      curseur.addChild(layer_lecture);
      // ajout du sprite d'affichage à l'élément curseur volume
      curseur_volume.addChild(layer_volume);
      curseur.buttonMode = true;
      curseur_volume.buttonMode = true;

      // Mise en ecoute de la navigation via la barre de progression
      curseur.addEventListener(MouseEvent.CLICK, allerA);
      curseur_volume.addEventListener(MouseEvent.CLICK, gestionVolume);

      // préremplissage du curseur de volume ( A fond par defaut)
var forme_volume :Sprite  = dessinerTemoin(curseur_volume.width,
curseur_volume.height);
forme_volume.y = curseur_volume.height - forme_volume.height;
layer_volume.addChild(forme_volume);

}

private function gestionVolume(even:MouseEvent):void
{
      // conversion de l'emplacement du clic en une fraction de 1
      var ratio = Math.abs(curseur_volume.mouseY - curseur_volume.height) /
curseur_volume.height;
      //retrait de le forme si presente
      while(layer_volume.numChildren >0)
      {
            layer_volume.removeChildAt(0);
      }
      // Commande de volume sur le lecteur.
      lecteur.volume = ratio

// On place une nouvelle barre de la taille correspondante à l'emplacement
// du clic
var forme_volume :Sprite = dessinerTemoin(curseur_volume.width,
curseur_volume.height * ratio);
forme_volume.y = curseur_volume.height - forme_volume.height;
layer_volume.addChild(forme_volume);
}
```

```
// Function de navigation dans la piste et de raffraichissement de la
// progression

private function allerA(even : MouseEvent) :void
{
	lecteur.stopperSon();
	// conversion de l'emplacement du clic en une fraction de 1
	var ratio = curseur.mouseX / curseur.width;
	// reprise du son par rapport à la nouvelle position de la tête de
lecture
	lecteur.position = lecteur.longueur_morceau * ratio;
	lecteur.jouerSon();

	// Si un forme est déjà présente , on l'enlève
	if(layer_lecture.numChildren > 0)
	{
		layer_lecture.removeChildAt(0);
	}
// On place une nouvelle barre de la taille correspondante à
//l'emplacement du clic
layer_lecture.addChild( dessinerTemoin(curseur.mouseX, curseur.height));
}

// Fonctions de mise à jour du temoin de progression

private function rafraichirTemoin (even:Event):void
{
	var nouvelle_pos : Number = (lecteur.position /
lecteur.longueur_morceau)* curseur.width;
	while(layer_lecture.numChildren > 0)
	{
		layer_lecture.removeChildAt(0);
	}
	var forme_curseur : Sprite = dessinerTemoin(nouvelle_pos,
curseur.height);
	layer_lecture.addChild(forme_curseur);

	champ_temps.text =
lecteur.formaterDuree(lecteur.position)+"/"+lecteur.formaterDuree(lecteur.l
ongueur_morceau);
	//trace(champ_temps.text)
}

private function dessinerChargement(pEvt:ProgressEvent):void
{
	while(layer_chargement.numChildren > 0)
	{
		layer_chargement.removeChildAt(0);
	}
	var ratio:Number = pEvt.bytesLoaded / pEvt.bytesTotal ;
	var taille_temoin:Number = curseur.width * ratio;
	var temoin_chargement:Sprite= dessinerTemoin(taille_temoin,
curseur.height, 0x99FF00, 0x00CC33)
	layer_chargement.addChild(temoin_chargement);
}
```

A partir de ce moment nous pouvons naviguer dans la piste modifier son volume et voir la progression de son chargement . Il nous reste à poser l'écouteur et la gestion des clics sur le clip « controles ».
Nous la donnons ici dans la classe maintenant au complet (en gras) :

```
package
{
import flash.display.MovieClip;
import flash.events.Event;
import flash.events.MouseEvent;
import flash.events.ProgressEvent;
import flash.events.TimerEvent;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.display.Sprite
import flash.utils.Timer;
import flash.display.SimpleButton;
import flash.text.TextField;

public class LecteurSonIDE extends MovieClip
{
// Switch pour le statut du lecteur
private var statut_silence : Boolean = false;

// Coeur de l'application contient toutes les fonction pour la manipulation
// des pistes jouées
private var lecteur:PisteSon

// Liste des Objets BadgeSon
private var tab_liste:Array = new Array();
// stockage des infos xml (optionnel)
private var donnees_xml:XML;
// ces objets sont implantés dans le clip "curseur" pour afficher
// les infos de progression de chargement de lecture .
private var layer_lecture : Sprite = new Sprite();
private var layer_chargement : Sprite = new Sprite();
// Implanté dans le curseur_volume
private var layer_volume : Sprite = new Sprite();

// L'horloge premet de raffraichir la barre de progression
private var horloge:Timer = new Timer(100);
// Premet la gestion de plages
private var morceau_en_cours:int = 0;

public function LecteurSonIDE()
{
	if(stage)
	addEventListener(Event.ADDED_TO_STAGE, init);
	else
	init()
}

private function init(e:Event = null):void
{
	lecteur = new PisteSon();
	removeEventListener(Event.ADDED_TO_STAGE, init);

	// Mise en place des écouteurs
	horloge.addEventListener(TimerEvent.TIMER, rafraichirTemoin);
	//reception des infos du fichier
	lecteur.addEventListener(Event.ID3, messagesPiste);
	lecteur.addEventListener(ProgressEvent.PROGRESS, dessinerChargement);
	lecteur.addEventListener(Event.SOUND_COMPLETE, jouerSuivant) ;

	//Mise en ecoute du clip contrôle en activant « useCapture »
	controles.addEventListener(MouseEvent.CLICK, gestionControles, true);

	// recherche de l'adresse du xml
	if (loaderInfo.parameters.liste != undefined)
	{
		chargementXML(loaderInfo.parameters.liste)
	}
```

```
		else
		{
			chargementXML("xml/playlist.xml")
		}

		// ajout des éléments au clip "curseur" de la scène
		curseur.addChild(layer_chargement);
		curseur.addChild(layer_lecture);
		// ajout du sprite d'affichage à l'élément curseur volume
		curseur_volume.addChild(layer_volume);
		curseur.buttonMode = true;
		curseur_volume.buttonMode = true;

		// Mise en ecoute de la navigation via la barre de progression
		curseur.addEventListener(MouseEvent.CLICK, allerA);
		curseur_volume.addEventListener(MouseEvent.CLICK, gestionVolume);

		// préremplissage du curseur de volume ( A fond par defaut)
		var forme_volume :Sprite  = dessinerTemoin(curseur_volume.width,
curseur_volume.height);
		forme_volume.y = curseur_volume.height - forme_volume.height;
		layer_volume.addChild(forme_volume);
}

private function gestionVolume(even:MouseEvent):void
{
// conversion de l'emplacement du clic en une fraction de 1
		var ratio = Math.abs(curseur_volume.mouseY - curseur_volume.height) /
curseur_volume.height;
// reprise du son par rapport à la nouvelle position de la tête de lecture

while(layer_volume.numChildren >0)
		{
			layer_volume.removeChildAt(0);
		}
		lecteur.volume = ratio
// On place une nouvelle barre de la taille correspondante à l'emplacement
// du clic
				var forme_volume :Sprite = dessinerTemoin(curseur_volume.width,
curseur_volume.height * ratio);
				forme_volume.y = curseur_volume.height - forme_volume.height;
				layer_volume.addChild(forme_volume);
}

private function gestionControles(even:MouseEvent):void
{
		if (even.target.name == "bt_prochain")
		{
			jouerSuivant();
		}

		if (even.target.name == "bt_precedent")
		{
			if (morceau_en_cours > 0)
			{
				morceau_en_cours--;
				lecteur.index_piste = morceau_en_cours;
			lecteur.chargerNouveauSon(tab_liste[morceau_en_cours].fichier);
			}
			else if(morceau_en_cours == 0)
			{
				morceau_en_cours = tab_liste.length -1;
		lecteur.index_piste = morceau_en_cours;
		lecteur.chargerNouveauSon(tab_liste[morceau_en_cours].fichier);
			}
		}

		if (even.target.name == "bt_stop")
```

```
		{
			horloge.stop();
			lecteur.stopperSon();
			champ_temps.text =
		lecteur.formaterDuree(0)+"/"+lecteur.formaterDuree(0000);
			layer_lecture.removeChildAt(0);

		}
		if (even.target.name == "bt_play")
		{
			horloge.start()
			lecteur.jouerSon();
		}
		if (even.target.name == "bt_pause")
		{
			horloge.stop();
			lecteur.pauserSon()
		}
}

// Connexion des éléments de la scène et du lecteur


private function allerA(even : MouseEvent) :void
{
		// conversion de l'emplacement du clic en une fraction de 1
		var ratio = curseur.mouseX / curseur.width;
		// reprise du son par rapport à la nouvelle position de la tête de lecture
		lecteur.position = lecteur.longueur_morceau * ratio;
		lecteur.jouerSon();

		// Si un forme est déjà présente , on l'enlève
		if(layer_lecture.numChildren > 0)
		{
			layer_lecture.removeChildAt(0);
		}
		// On place une nouvelle barre de la taille correspondante à l'emplacement du
clic
layer_lecture.addChild( dessinerTemoin(curseur.mouseX, curseur.height));
}

private function jouerSuivant(even:Event = null):void
{
		trace(tab_liste[morceau_en_cours].fichier);
		if (morceau_en_cours < tab_liste.length -1)
		{
			morceau_en_cours++;
			lecteur.chargerNouveauSon(tab_liste[morceau_en_cours].fichier);
		}
		else if(morceau_en_cours == tab_liste.length -1)
		{
			morceau_en_cours = 0 ;
			lecteur.chargerNouveauSon(tab_liste[morceau_en_cours].fichier);
		}

			}


private function messagesPiste(even:Event):void
{
			tab_liste[morceau_en_cours].ID3 = lecteur.ID3;
			champ_info.text = tab_liste[morceau_en_cours].description;
			for (var v : * in lecteur.ID3)
			{
				trace(v+" : "+lecteur.ID3[v]);
			}
}
```

```
// ----------------------CHARGEMENT XML -------------------------//

public function chargementXML(url:String) : void
{
	var loader:URLLoader = new URLLoader();
	var requete :URLRequest = new URLRequest(url);
	loader.addEventListener(Event.COMPLETE, XMLCharge);
	loader.load(requete);
}

// L'application est deployée ici
private function XMLCharge(even:Event):void
{
	var loader:URLLoader = even.target as URLLoader;
	donnees_xml = new XML(loader.data);
	//XML.prettyIndent = 0;
	//XML.ignoreWhitespace = true;

	//standardisation des ifos xml au moyen d'un objet personnalisé BadgeSon

	for (var i :int; i < donnees_xml.playlist.morceau.length();i++ )
	{
		var badge:BadgeSon = new BadgeSon();
		badge.index = i;
		badge.titre = donnees_xml.playlist.morceau[i].titre;
		badge.artiste = donnees_xml.playlist.morceau[i].artiste;
		badge.fichier = donnees_xml.playlist.morceau[i].fichier;
		tab_liste.push(badge);
	}


			// Chargement du premier morceau
			lecteur.chargerNouveauSon(tab_liste[0].fichier);

			lecteur.jouerSon();
			// mise en route de l'horloge.
			horloge.start()
}

// Fonctions de mise à jour du temoin de progression

private function rafraichirTemoin (even:Event):void
{
	var nouvelle_pos : Number = (lecteur.position / lecteur.longueur_morceau)*
curseur.width;
	while(layer_lecture.numChildren > 0)
	{
		layer_lecture.removeChildAt(0);
	}	var forme_curseur : Sprite = dessinerTemoin(nouvelle_pos,
curseur.height);
	layer_lecture.addChild(forme_curseur);

champ_temps.text =
lecteur.formaterDuree(lecteur.position)+"/"+lecteur.formaterDuree(lecteur.longueur_
morceau);
	//trace(champ_temps.text)
}


private function dessinerChargement(pEvt:ProgressEvent):void
{
	while(layer_chargement.numChildren > 0)
	{
		layer_chargement.removeChildAt(0);
	}
	var ratio:Number = pEvt.bytesLoaded / pEvt.bytesTotal ;
	var taille_temoin:Number = curseur.width * ratio;
```

```
		var temoin_chargement:Sprite= dessinerTemoin(taille_temoin, curseur.height,
0x99FF00, 0x00CC33)
		layer_chargement.addChild(temoin_chargement);
}


private function dessinerTemoin(largeur:Number, hauteur : Number, couleur : Number
= 0x0066FF,palpha : Number = 0.7):Sprite
{
		var dessin:Sprite = new Sprite();
		dessin.graphics.beginFill(couleur,palpha);
		dessin.graphics.drawRoundRect(0,0,largeur,hauteur,3,3);
		dessin.mouseEnabled = false;
		return dessin;
}

}
}
.
```

Mise en place de la playlist

Nous finirons avec notre lecteur en abordant un objet optionnel : l'affichage d'une liste de lecture interactive.
Nous allons en profiter pour faire la rencontre d'une fonction fort interessante dans le langage actionscript qui va nous permettre de copier l'aspect d'un clip et de ce qu'il contient sur un objet bitmap et cela à l'exécution.
Prenons d'abord le temps de réfléchir aux besoins d'un tel objet :
J'ai besoin :

- D'une série de bouton sur lesquels j'inscrirai ces informations. Il faudra donc que je les crée de façon dynamique.
- d'une liste constituée des données à afficher.
- Je vais avoir besoin de relier ces boutons au fonctionnement de notre lecteur.
- Je vais avoir besoin d'habiller les boutons et l'objet lui-même.
- Je voudrai pouvoir rafraîchir les titres à l'aide des infos id3.

Prenons ces différents point et considérons les solutions que nous pouvons appliquer à ces questions.
Pour cela gardons en tête que la construction d'une class prend du temps et essayons d'appliquer des solution qui soient transposables dans d'autres situations.

1 les boutons.

Les boutons ici auront un fonctionnement très simple. Inutile de créer une sous-classe pour cela. Une seule fonction créant l'objet à l'aide des données de la classe et des informations des objets BadgeSon devrai suffire. Nous utiliserons une boucle de remplissage qui appellera la fonction de création de boutons.

```
/* La fonction de création montre un procédé intéressant
Comment faire une copie visuelle de type Bitmap à partir un objet
d'affichage ou un groupe d'objet :  On obtient alors une "photo" de l'objet
(plus d'interactivité). Manupulable à volonté. */
```

```
/* Pour les différents états du bouton je modifie les propriété d'un même
objet. Comme si, en photo je demandais au modèle de changer de pose pour
chaque nouvelle image... */
private function creerEtiquette(badge:BadgeSon):MovieClip
{
// création d'un container pour pouvoir utiliser les outils de dessin
(surbrillance)
var cont:MovieClip = new MovieClip();
// création du bouton
var bt : SimpleButton = new SimpleButton();
// je donne au bouton comme nom l'index transformé en chaine de caractères.
bt.name = String(badge.index);
// Je crée un sprite
var fond:Sprite = new Sprite();
// Création de l'état normal à partir de données du xml
fond.graphics.beginFill(skin.normal.@couleur, skin.normal.@alpha)
fond.graphics.drawRect(0, 0, limites.width, 20);

// Création du champ texte à l'état normal
var tf:TextField = new TextField();
// définition du format
var format : TextFormat = new TextFormat(fonte, skin.@taille_txt,
skin.normal.@c_texte);
// Attribution d'un format par défaut
tf.defaultTextFormat = format;
var str:String = (badge.index + 1) + ": " + badge.artiste + " : " +
badge.titre;
// limitation du texte à 30 catactères (pour qu'il ne depasse pas du
bouton).
tf.text = str.substr(0, 30);
tf.height = tf.textHeight + 5;
tf.autoSize = TextFieldAutoSize.LEFT;
tf.x = 10;
// placement du champ texte dans le sprite "fond"
fond.addChild(tf);

// configuration pour l'état enfoncé et cliqué.
/* L'objet bitmapData est l'objet qui contient effectivement les données
d'image dans un bitmap ("fichier" image). Je l'initialise en lui donnant
ses dimensions.*/
// ici je prépare l'image de l'état enfoncé du bouton.
var dataDown:BitmapData = new BitmapData(fond.width, fond.height, true, 0);
// Puis je le remplis avec la copie de pixels du clip "fond" à l'aide de sa
méthode draw.
dataDown.draw(fond);

//Copie pour l'état cliqué
var dataHit:BitmapData = new BitmapData(fond.width, fond.height, true, 0);
dataHit.draw(fond);

//modification du fond et du champ texte avant copie pour l'état normal
fond.graphics.clear();
fond.graphics.beginFill(skin.normal.@couleur, skin.normal.@alpha);
fond.graphics.drawRect(0, 0, limites.width, skin.@h_etiquette);
tf.setTextFormat(new TextFormat(fonte, skin.@taille_txt,
skin.normal.@c_texte));
var dataUp:BitmapData = new BitmapData(fond.width, fond.height, true, 0);
dataUp.draw(fond);

// puis même opération pour l'état survolé
fond.graphics.clear();
```

```
fond.graphics.beginFill(skin.dessus.@couleur, skin.dessus.@alpha);
fond.graphics.drawRect(0, 0, limites.width, 20);
tf.setTextFormat(new TextFormat(fonte, skin.@taille_txt,
skin.dessus.@c_texte));
var dataOver:BitmapData = new BitmapData(fond.width, fond.height, true, 0);
dataOver.draw(fond);

// J'instancie un bitamp vide qui va contenir les données des objets
bitmapData

bt.downState = new Bitmap(dataDown);
bt.hitTestState = new Bitmap(dataHit);
bt.upState = new Bitmap(dataUp);
bt.overState = new Bitmap(dataOver);

// placement dynamique de l'objet badge sur le clip pour d'autres modes de
fonctionnement que celui présentement employé
cont["badge"] = badge;
cont.addChild(bt);
return cont;


}
```

2 la liste.

Nous allons nous servir de la liste d'objets que nous avons constitué au chargement du xml. Nous fournirons cette liste dès le constructeur de la classe. Elle nous servira à établir la liste graphique et a retrouver le morceau joué lors de la sélection d'un nouveau morceau par l'emploi de la playlist ou d'un autre système de navigation du lecteur.
La création d'une liste graphique en actionScript passe presque toujours par le parcours d'un tableau d'objet ou d'une liste xml à l'intérieur d'une boucle chargé d'interpréter les informations de leur donner une apparence et de les afficher dans l'ordre et l'axe souhaité.
Nous plaçons les boutons dans un container pour pouvoir modifier l'emplacement et de masquer les boutons autres que celui en lecture, le précédent et le suivant.

```
private var tab_morceaux : Array;
// container des boutons
private var layer_liste:MovieClip = new MovieClip();


public function contruirePlayList(): void
{
      for each(var musique:BadgeSon in tab_morceaux)
      {
            var etiquette:MovieClip = creerEtiquette(musique);
// on place le bouton
            etiquette.y = this.height;
// on crée une propriété à la volée dans le badge faisant référence au
bouton
            musique["etiquette"] = etiquette
            layer_liste.addChild(etiquette);
      }

      creerEtiquetteéation et activation du masque
      var forme:Shape = new Shape()
      forme.graphics.beginFill(0, 0);
forme.graphics.drawRoundRect(0, 0, limites.width, limites.height, 2, 2);
```

```
        addChild(forme);
        layer_liste.mask = forme;
}
```

2 Liaison au fonctionnement du lecteur musical.

D'ores et déjà nous savons que nous allons avoir dans cette objet des interactions entrantes (utilisation de la console, passage automatique au nouveau morceau) et des interactions sortantes (clic sur l'un des morceau de la liste.)
Dans ce cas une bonne solution pourrait être de mettre au point un système de sélection passive. Une fonction publique pourrait être mise en place demandant en paramètre l'index du morceau à sélectionner. De cette manière nous pourrions externaliser également la sélection souris et la leguer à la classe principale qui à déjà en charge l'ensemble des interactions du lecteur. Le problème sera ensuite de pouvoir différencier les boutons entre eux pour retrouver le bon morceau. Bien des solutions s'offrent à nous pour résoudre cette question :

- Nous pouvons rendre la classe BadgeSon dynamique afin de lui faire accepter des paramètres non encore écrits sous forme de variables publique. Ainsi nous pourrions y stocker quel est le bouton qui lui est attribué et le retrouver aisément dans la fonction évènementiel et comparant la cible du clic et la référence du bouton dans une boucle.
- Nous pouvons référencer le numéro d'index , voire le l'objet badge dans une instance de MovieClip englobant le bouton (le MovieClip est une classe dynamique).
- Nous pouvons tricher avec actionScript et nommer nos boutons avec leur numéro d'index transformé en chaine (le paramètre name ne peut être un objet de type Number).

Nous choisirons cette dernière pour sa simplicité d'utilisation même si ce n'est pas une méthode très orthodoxe ! Cependant elle fonctionne nous allons le voire sans produire de comportement non attendu.

```
public function selection(index:int):void
{
        var badge_sel:BadgeSon;


        for each(var badge in tab_morceaux)
        {
                // on recupère le bouton.
                var etiquette : MovieClip = badge.etiquette;
                if (index != badge.index)
                {
        // si ce n'est pas la selection on efface la surbrillance eventuelle
                        etiquette.graphics.clear();
                }
                else
                {
        // stockage du badge selectionné
                        badge_sel = badge;
        // mise en surbrillance
                        etiquette.graphics.beginFill(0xffffff, 1);
                        etiquette.graphics.drawRect(1, 1, etiquette.width -2,
etiquette.height-2);
                }
        }
        // Placement du bouton au centre de la partie visible de la liste
```

```
            layer_liste.y = - (badge_sel.etiquette.y -
badge_sel.etiquette.height);

            if (badge_sel.index == tab_morceaux.length -1)
            {// si c'est la dernier on le recale en bas

                  layer_liste.y += badge_sel.etiquette.height;
            }
            if (badge_sel.index == 0 )
            {
                  // si c'est le bouton du haut on le laisse au dessus
                  layer_liste.y = 0;
            }
}
```

4 l'habillage

L'habillage de notre objet concerne deux choses différentes l'aspect des boutons et l'habillage de la liste. :

- Pour les boutons nous allons stocker les données de couleur pour les différents aspect au sein d'un fragment de xml. Nous disposerons ainsi d'un comportement par defaut que nous pourrons aisement réécrire en fournissant à notre classe un habillage alternatif que nous pourrions faire transiter par les xml externe. Cet aspect implique un éléement important dans la structure de notre classe. Elle ne doit se déclencher qu'une fois les éventuels paramètres alternatifs attribués. Nous choissons pour cela de créer une fonctions qui mettra effectivement en place la liste graphiquement.

```
private var _skin :XML =
      <skin_playlist h_etiquette = "20" taille_txt="12">
            <select couleur="0xCFCFCF" alpha=".6" />
            <normal couleur="0x12337E" alpha="0.2" c_texte = "0x333333"/>
            <dessus couleur="0x4575E4" alpha="0.2" c_texte = "0xFFFFFF"/>
            <click couleur="0xFFFFFF" alpha="0.4" c_texte = "0x000000"/>
      </skin_playlist>

public function get skin():XML { return _skin; }
public function set skin(value:XML):void { _skin = value; }
```

- Nous nous laissons également la possiblité de donner à la playlist une typo différente en stockant la chaine l'identifiant (police.fontName) et en placant un setter

```
private var fonte:String = "arial";
public function set police (_fonte:String) : void {fonte = _fonte; }
```

- L'habillage de notre liste elle pourrait être gérée par un clip créé dans le logiciel et rendu exportable dans la bibliothèque. Nous pourrons ainsi en modifier l'aspect pour le conformer aux différentes chartes graphiques dans lesquelles ils devra s'insérer.
  Le clip comportement un avant plan : un clip imbriqué portant le nom d'occurrence « coque » et un clip de fond qui sera automatiquement placé sous la liste.

```
public function set habillage (hab : MovieClip):void
{
      addChildAt(hab, 0)
      var coque:MovieClip = hab.getChildByName("coque") as MovieClip;
      addChild(coque)
}
```

5 rafraîchissement

Nous classe principale comporte déjà une fonction chargé du rafraîchissement des objets lors de la receptions des étiquettes ID3. Nous pourrions l'exploiter en créant ici une fonction publique prenant en paramètre le numéro d'index pour retrouver le bouton concerné le recréer et le replacer à ses coordonnées initiales.

```
public function raffraichirTitre(index:int)
{
	for each(var badge in tab_morceaux)
	{
		var etiquette : MovieClip = badge.etiquette;
		if (index == tab_morceaux.indexOf(badge))
		{
			etiquette.graphics.clear();
			var limites:Rectangle =
etiquette.getBounds(layer_liste).clone();
			etiquette.parent.removeChild(etiquette)
			etiquette = creerEtiquette(badge);
			badge["etiquette"] = etiquette
			etiquette.y = limites.y;
			etiquette.x = limites.x;
			layer_liste.addChild(badge.etiquette)
		}
	}
	selection(index)
}
```

6 Classe playlist au complet :

```
package
{
	import flash.display.Bitmap;
	import flash.display.BitmapData;
	import flash.display.DisplayObject;
	import flash.display.MovieClip;
	import flash.display.Shape;
	import flash.display.SimpleButton;
	import flash.display.Sprite;
	import flash.events.MouseEvent;
	import flash.geom.Rectangle;
	import flash.text.TextField;
	import flash.text.TextFieldAutoSize;
	import flash.text.TextFormat;
	/**

	 *  - La classe playlist receuille un tableau d'objet badge son .
	 * - Crée des occurences de boutons à l'aide de la liste et des
données du xml (_skin) pour definir couleurs et tailles
	 * - On deploie la playlist à partir de la fonction
construirePlaylist(). Cela laisse la possibilité de fournir
	 * d'habillage alternatif :
	 * - Il est possible de definir un clip contenant le fond de la liste
et son habillage. On lui envoie
	 * un clip créé dans le logiciel composé de deux autre clips portant
comme nom d'occurence "fond" et "coque"
```

```
	 * - Il est possible aussi de fournir un xml alternatif.
	 * - la classe ne contient pas d'écouteur. Elle fonctionne
mécaniquement en lui fournissant le numéro d'index
	 * à mettre en surbrillance.
	 */


public class PlayList extends MovieClip
{
	// receuille le tableau composé d'objets BadgeSon (funct. const.)
private var tab_morceaux : Array;
	// stocke une police alternative (setter : police)
private var fonte:String = "arial";
	// container des boutons
private var layer_liste:MovieClip = new MovieClip();
// Taille maximale de l'affichage de la liste (funct. const.). Definit la
taille du masque
private var limites : Rectangle = new Rectangle(0, 0, 300, 60);
// Les paramètres de couleur des bouttons sont paramétrables via un xml
structuré de cette manière : (get. set. skin)
private var _skin :XML =
	<skin_playlist h_etiquette = "20" taille_txt="12">
		<select couleur="0xCFCFCF" alpha=".6" />
		<normal couleur="0x12337E" alpha="0.2" c_texte = "0x333333"/>
		<dessus couleur="0x4575E4" alpha="0.2" c_texte = "0xFFFFFF"/>
		<click couleur="0xFFFFFF" alpha="0.4" c_texte = "0x000000"/>
	</skin_playlist>

public function PlayList(morceaux:Array, plimites : Rectangle = null)
{
	// on definit les nouvelles limites si le paramètre n'est pas nul
	if (plimites != null )
	{
		limites = plimites;
	}
	// on associe la liste d'objet BadgeSon au tableau tab_morceau.
	tab_morceaux = morceaux;
	// placement de la liste à x = 0 et Y = 0;
	addChild(layer_liste)

}
	// On fournit le nom de la police alternative
public function set police (_fonte:String) : void {fonte = _fonte; }

// On peu fournir un xml sur le modele de skin afin de modifier les
couleurs des boutons
public function get skin():XML { return _skin; }
public function set skin(value:XML):void { _skin = value; }

// On peut habiller la playlist avec un clip crée dans flash
// celui ci contient deux clips on récupère celui nommé "coque"
// pour le placer au dessus de la liste
public function set habillage (hab : MovieClip):void
{
	addChildAt(hab, 0)
	var coque:MovieClip = hab.getChildByName("coque") as MovieClip;
	addChild(coque)
}

// Boucle à l'intérieur du tableau de badge afin de créer les boutons et de
les placer dans la liste
```

```
public function contruirePlayList(): void
{
	for each(var musique:BadgeSon in tab_morceaux)
	{
		var etiquette:MovieClip = creerEtiquette(musique);
// on place le bouton
		etiquette.y = this.height;
// on crée une propriété à la volée dans le badge faisant référence au
bouton
		musique["etiquette"] = etiquette
		layer_liste.addChild(etiquette);
	}

	creerEtiquetteéation et activation du masque
	var forme:Shape = new Shape()
	forme.graphics.beginFill(0, 0);
forme.graphics.drawRoundRect(0, 0, limites.width, limites.height, 2, 2);
	addChild(forme);
	layer_liste.mask = forme;
}

// On trouve le badge séléctionné pour le mettre en surbrillance.
public function selection(index:int):void
{
	var badge_sel:BadgeSon;


	for each(var badge in tab_morceaux)
	{
		// on recupère le bouton.
		var etiquette : MovieClip = badge.etiquette;
		if (index != badge.index)
		{
	// si ce n'est pas la selection on efface la surbrillance eventuelle
			etiquette.graphics.clear();
		}
		else
		{
	// stockage du badge selectionné
			badge_sel = badge;
	// mise en surbrillance
			etiquette.graphics.beginFill(0xffffff, 1);
			etiquette.graphics.drawRect(1, 1, etiquette.width -2,
etiquette.height-2);
		}
	}
	// Placement du bouton au centre de la partie visible de la liste
		layer_liste.y = - (badge_sel.etiquette.y -
badge_sel.etiquette.height);

		if (badge_sel.index == tab_morceaux.length -1)
		{// si c'est la dernier on le recale en bas

			layer_liste.y += badge_sel.etiquette.height;
		}
		if (badge_sel.index == 0 )
		{
			// si c'est le bouton du haut on le laisse au dessus
			layer_liste.y = 0;
		}
}
```

```
// cette fonction permet de raffraichir l'affichage du bouton Pour prendre
en compte les infos ID3
public function raffraichirTitre(index:int)
{
	for each(var badge in tab_morceaux)
	{
		var etiquette : MovieClip = badge.etiquette;
		if (index == tab_morceaux.indexOf(badge))
		{
			etiquette.graphics.clear();
			var limites:Rectangle =
etiquette.getBounds(layer_liste).clone();
			etiquette.parent.removeChild(etiquette)
			etiquette = creerEtiquette(badge);
			badge["etiquette"] = etiquette
			etiquette.y = limites.y;
			etiquette.x = limites.x;
			layer_liste.addChild(badge.etiquette)
		}
	}
	selection(index)
}

/* La fonction de création montre un procédé intéressant
Comment faire une copie visuelle de type Bitmap à partir un objet
d'affichage ou un groupe d'objet :  On obtient alors une "photo" de l'objet
(plus d'interactivité). Manupulable à volonté. */
/* Pour les différents états du bouton je modifie les propriété d'un même
objet. Comme si, en photo je demandais au modèle de changer de pose pour
chaque nouvelle image... */
private function creerEtiquette(badge:BadgeSon):MovieClip
{
// création d'un container pour pouvoir utiliser les outils de dessin
(surbrillance)
var cont:MovieClip = new MovieClip();
// création du bouton
var bt : SimpleButton = new SimpleButton();
// je donne au bouton comme nom l'index transformé en chaine de caractères.
bt.name = String(badge.index);
// Je crée un sprite
var fond:Sprite = new Sprite();
// Création de l'état normal à partir de données du xml
fond.graphics.beginFill(skin.normal.@couleur, skin.normal.@alpha)
fond.graphics.drawRect(0, 0, limites.width, 20);

// Création du champ texte à l'état normal
var tf:TextField = new TextField();
// définition du format
var format : TextFormat = new TextFormat(fonte, skin.@taille_txt,
skin.normal.@c_texte);
// Attribution d'un format par défaut
tf.defaultTextFormat = format;
var str:String = (badge.index + 1) + ": " + badge.artiste + " : " +
badge.titre;
// limitation du texte à 30 catactères (pour qu'il ne depasse pas du
bouton).
tf.text = str.substr(0, 30);
tf.height = tf.textHeight + 5;
tf.autoSize = TextFieldAutoSize.LEFT;
tf.x = 10;
```

```
// placement du champ texte dans le sprite "fond"
fond.addChild(tf);

// configuration pour l'état enfoncé et cliqué.
/* L'objet bitmapData est l'objet qui contient effectivement les données
d'image dans un bitmap ("fichier" image). Je l'initialise en lui donnant
ses dimensions.*/
// ici je prépare l'image de l'état enfoncé du bouton.
var dataDown:BitmapData = new BitmapData(fond.width, fond.height, true, 0);
// Puis je le remplis avec la copie de pixels du clip "fond" à l'aide de sa
méthode draw.
dataDown.draw(fond);

//Copie pour l'état cliqué
var dataHit:BitmapData = new BitmapData(fond.width, fond.height, true, 0);
dataHit.draw(fond);

//modification du fond et du champ texte avant copie pour l'état normal
fond.graphics.clear();
fond.graphics.beginFill(skin.normal.@couleur, skin.normal.@alpha);
fond.graphics.drawRect(0, 0, limites.width, skin.@h_etiquette);
tf.setTextFormat(new TextFormat(fonte, skin.@taille_txt,
skin.normal.@c_texte));
var dataUp:BitmapData = new BitmapData(fond.width, fond.height, true, 0);
dataUp.draw(fond);

// puis même opération pour l'état survolé
fond.graphics.clear();
fond.graphics.beginFill(skin.dessus.@couleur, skin.dessus.@alpha);
fond.graphics.drawRect(0, 0, limites.width, 20);
tf.setTextFormat(new TextFormat(fonte, skin.@taille_txt,
skin.dessus.@c_texte));
var dataOver:BitmapData = new BitmapData(fond.width, fond.height, true, 0);
dataOver.draw(fond);

// J'instancie un bitamp vide qui va contenir les données des objets
bitmapData

bt.downState = new Bitmap(dataDown);
bt.hitTestState = new Bitmap(dataHit);
bt.upState = new Bitmap(dataUp);
bt.overState = new Bitmap(dataOver);

// placement dynamique de l'objet badge sur le clip pour d'autres modes de
fonctionnement que celui présentement employé
cont["badge"] = badge;
cont.addChild(bt);
return cont;


}

}
}
```

7 implatation dans la classe principale

Revenus dans notre classe principale nous placons maintenant une variable de classe pour pouvoir accéder à l'objet depuis l'ensemble de nos fonctions :

```
private var playlist : PlayList;
```

puis dans la fonction XMLChargé nous instancions et affichons la playlist au dessus panneau infos. Nous plaçons la propriété visible de celui-ci sur false. Nous pouvons placer dans l'animation un clip chargé de rendre visible soit l'un soit l'autre ainsi l'utilisateur en cliquant accède soit à la playlist soit aux infos ID3 du morceau en cours.

```
private function XMLCharge(even:Event):void
{
	var loader:URLLoader = even.target as URLLoader;
	donnees_xml = new XML(loader.data);
	//XML.prettyIndent = 0;
	//XML.ignoreWhitespace = true;

	//standardisation des ifos xml au moyen d'un objet personnalisé BadgeSon

	for (var i :int; i < donnees_xml.playlist.morceau.length();i++ )
	{
		var badge:BadgeSon = new BadgeSon();
		badge.index = i;
		badge.titre = donnees_xml.playlist.morceau[i].titre;
		badge.artiste = donnees_xml.playlist.morceau[i].artiste;
		badge.fichier = donnees_xml.playlist.morceau[i].fichier;
		tab_liste.push(badge);
	}

		// Installation de la playlist

	playlist = new PlayList(tab_liste);
	playlist.police = Font.enumerateFonts()[0].fontName;
	playlist.contruirePlayList();
	addChild(playlist);
	playlist.habillage = new HabillagePlaylist();
	playlist.x = champ_info.x;
	playlist.y = champ_info.y;
	playlist.addEventListener(MouseEvent.CLICK, selectionPL, true);

	champ_info.visible = false;
	fond_infos.visible = false;
	// Chargement du premier morceau
	lecteur.chargerNouveauSon(tab_liste[0].fichier);

	//mise en surbrillance du morceau en cours
	playlist.selection(0)
	lecteur.jouerSon();
	// mise en route de l'horloge.
	horloge.start()
}


private function selectionPL(even:MouseEvent):void
{
	lecteur.chargerNouveauSon(tab_liste[int(even.target.name)].fichier);
	lecteur.index_piste = int(even.target.name)
	playlist.selection(int(even.target.name));
	morceau_en_cours = int(even.target.name)
}
```

Nous ajoutons les comportements adaptés dans les différentes fonctions :

Dans gestionControles :

```
…
if (even.target.name == "bt_precedent")
	{
		if (morceau_en_cours > 0)
		{
			morceau_en_cours--;
			lecteur.index_piste = morceau_en_cours;
		lecteur.chargerNouveauSon(tab_liste[morceau_en_cours].fichier);
		}
		else if(morceau_en_cours == 0)
		{
			morceau_en_cours = tab_liste.length -1;
			lecteur.index_piste = morceau_en_cours;
	lecteur.chargerNouveauSon(tab_liste[morceau_en_cours].fichier);
		}
		playlist.selection(morceau_en_cours);
	}
…
```

Nous plaçons également la même instruction à la fin de la fonction jouerSuivant :

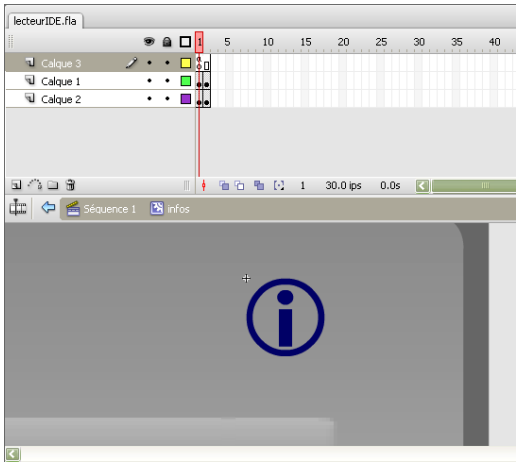
```
playlist.selection(morceau_en_cours);
```

enfin dans la fonction messagesPistes nous assurons le rafraîchissement des boutons par les étiquettes id3 :

```
private function messagesPiste(even:Event):void
{
	tab_liste[morceau_en_cours].ID3 = lecteur.ID3;
	playlist.raffraichirTitre(morceau_en_cours);
	champ_info.text = tab_liste[morceau_en_cours].description;
}
```

Mise en place du bouton infos

Le bouton infos permettra de faire apparaître alternativement la playlist ou bien les informations id3 du morceau en cours . Nous mettons d'abord en place les éléments supplementaires sur la scène de notre application :



La deuxième image du clip est un changement de couleur. L'action est un stop pour éviter la mise en lecture du clip.

Voici les éléments ajoutés à notre application et les noms d'occurrence attribués :

Enfin nous ajoutons un écouteur de clics à l'élément « infos » dans la fonction xmlCharge (peu importe l'endroit dans la fonction :

```
bt_infos = infos;
bt_infos.addEventListener(MouseEvent.CLICK,affichageInfos)
```

voici la fonction de gestion du bouton

```
private function affichageInfos(e:MouseEvent):void
{
	if(playlist.visible)
	{
		playlist.visible = false;
		champ_info.visible = true;
		fond_infos.visible = true;
		bt_infos.gotoAndStop(2);
	}
	else
	{
		playlist.visible = true;
		champ_info.visible = false;
		fond_infos.visible = false;
		bt_infos.gotoAndStop(1);
	}

}
```

Conclusion

Nous venons de mettre en place un lecteur musical complet et en très grande partie dynamique. Nous pouvons voir que la mise en place d'une application pourtant d'aspect simple peut être longue si l'on se préoccupe de l'ensemble de opérations et de interactions indispensables à sa bonne ergonomie.
Nous voyons ici tout l'intérêt qu'il y a à préparer son travail par les étapes que nous avons évoqué au chapitre précédent.
Ce chapitre nous aura aussi permis de bien mettre en évidence l'importance des évènement dans la bonne synchronisation de ce type d'application. Nous aurons eu également l'occasion d'aborder la notion de classe au plus proches des véritables intérêts qu'elle peut offrir. En effet, en créant un application spécialisée nous avons également mis en place une classe de gestion

de sons qui pourra sans problème être réutilisable dans la plupart des cas où un son lourd doit être diffusé par un chargement externe et éventuellement contrôlé à divers degré. Cette façon d'opérer nous permettra à l'avenir d'économiser le temps dédié à l'écriture d'une classe semblable.
Optionnellement nous aurons enfin pu rencontrer l'objet BitmapData et de la possibilité de copie d'objet qu'il recèle.