# Partie 4 : Sur les classes

**Comment organiser son code ?**

La souplesse de flash, permet l'insertion de codes à de multiples niveaux : Scénario, clips, classes reliées à des clips ou au scénario principal. Il est même possible de se passer entièrement du logiciel est de se servir uniquement du code pour fonctionner en reliant FlashDevelop au « flex sdk » et à la dernière version du player flash en version debugger.
Dès lors il peut apparaître un peu déconcertant de savoir ou l'on devra écrire telle ou telle fonctionnalité de notre application.
Le principe prévaut dans cette nouvelle mouture d'actionScript d'utiliser la programmation orientée objet, c'est-à-dire d'employer des classes pour regrouper chaque fonctionnalité particulière.
C'est un principe qui trouve toute sa vérité dans des applications complexes dotées de nombreuses interactions, mais qui convenons-le au départ peut sembler une gageure, sans avoir pratiqué et en mesuré les différentes implications.

Lorsqu'il s'agit de créer une petite application ou une animation dotée de quelques interactions, l'intérêt des classes peut s'avérer un peu restreinte. L'intérêt d'une classe est sa portabilité, sa capacité d'être réemployée dans un autre contexte sans devenir une usine à gaz.
Pour une animation il est possible aussi avec un peu d'astuce de mélanger éléments créés et interpolés sur la scène et des éléments programmés. Voire de programmer une animation dans flash sans avoir à se servir du scénario principal du logiciel, l'interêt de créer une classe pour cela est que l'animation en question presente un aspect systématique. C'est-à-dire que son comportement soit entièrement lié à une action répétée et ou faisant appel au hasard… sinon le calque actions-image reste le moyen de travailler rapidement.

La seule contrainte reste dans flash la synchronisation des éléments. L'actionscript est exécuté de manière asynchrone, c'est-à-dire qu'il n'attend pas la fin d'une opération pour déclencher la suivante. Une bonne partie du code consiste souvent à indiquer au lecteur flash d'attendre que l'image ou le document soit chargé et quoi faire ou montrer en attendant.
Même pour des petits objets utilisés d'animation en animations il peut s'avérer judicieux de coder les quelques lignes sur la ligne de scénario du clip. Avez-vous remarqué lors d'une exportation pour actionscript que le clip est transformé en une classe qui contiendra toutes les définitions de fonctions ?

Voici par exemple la classe d'un clip contenant une zone sensible programmée sur un calque « actions » vers un lien html créé dans flash :

```
package intro_croisieres_V2_fla
{
	import flash.display.MovieClip;
	import flash.events.MouseEvent;

	public class skip_me_AS3_4 extends MovieClip
	{
		function frame1 () : *;

		public function goHome (param0:MouseEvent) : *;

		public function skip_me_AS3_4 ();
	}
```

```
}
```

En somme, si c'est possible, profitez de cette fonctionnalité offerte par le logiciel… L'important est de bien garder en vue la complexité de votre application : A partir d'un certain point - 150 à 200 lignes - il devient nettement plus pertinent d'organiser son code à l'aide de classes.
Les éléments redondants peuvent être aussi envisagés. Ne serait-ce que par exemple un chargement d'image dont vous aurez l'utilité à maintes reprises : évidemment une classe.

# 1. Création d'applications, en préparer les différents aspects

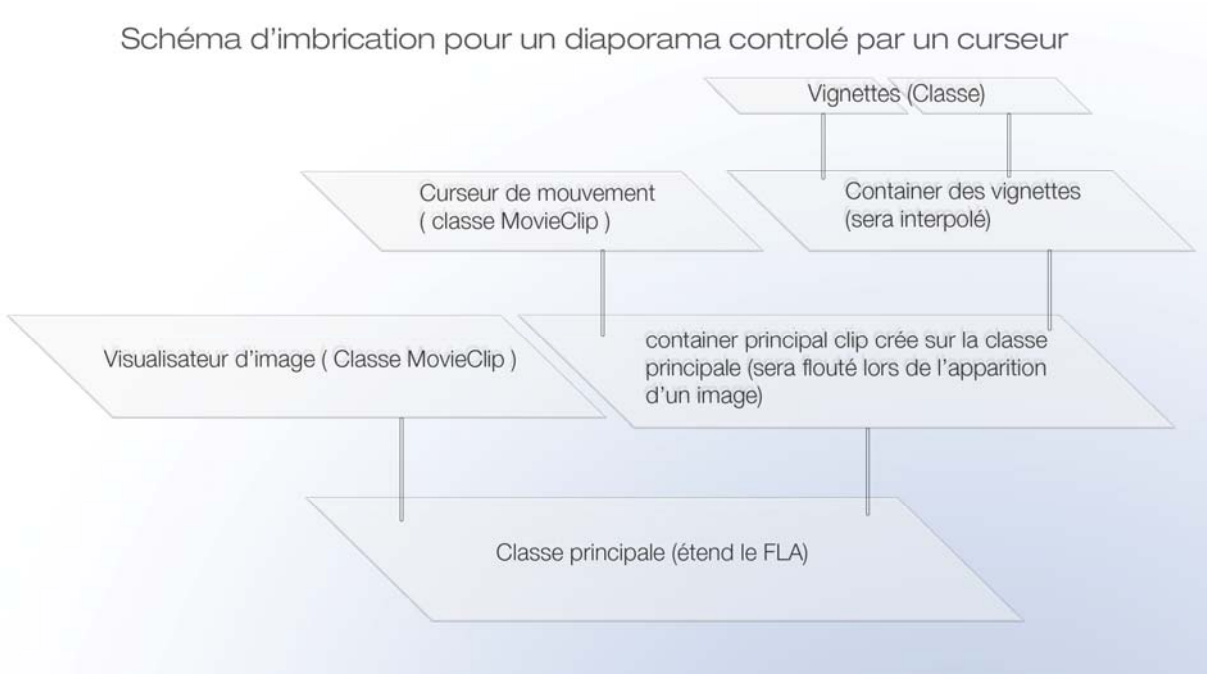
**Le schéma Temporel.**

Les applications flash ont un déroulement dans le temps et c'est un paramètre qu'il faut d'ores et déjà intégrer lors de la conception d'une application.
Cela commence souvent par : « mon application comporte pas mal de graphiques internes, il faudra donc prévoir une barre ou une animation de chargement… ».
On entre dès lors dans la scénarisation de l'application qu'il va falloir s'efforcer d'optimiser en fonction du temps imparti et du budget de l'application. Cette scénarisation s'opère en visitant mentalement l'architecture du site ou de l'application, et en évaluant les temps d'attente en fonction du poids des éléments à charger… etc.
Le poids du code étant négligeable, ce sont bien les éléments graphiques qu'il faudra envisager rigoureusement.
L'intégration de polices de caractères, par exemple, pouvant être très coûteuses en poids surtout si l'on embarque chacune en gras et italique. La question se posera alors de savoir si on la charge à la racine de notre document maître ou par le biais d'une bibliothèque d'un swf externe… Ce qui allègera l'application mais rendra la programmation plus délicate.

Le schéma temporel est donc un véritable story-board, qui commence par la couverture des temps d'attente. Mais, soigné autant qu'il le mérite, il deviendra l'outil qui plongera l'internaute dans un véritable univers… en résolvant des questions si « simples » soit elles que « comment vais-je passer d'une rubrique à l'autre ? ».
Cette charte est aussi indispensable pour la scénarisation éventuelle, que le sont les plans de coupe au cinéma pour dynamiser le montage et immerger un peu plus le spectateur dans le récit. Ne confondez pas cela avec une simple transition par exemple entre deux rubriques d'un site classique qui peuvent être programmées à la fin à partir du moment où ils auront été prévus : Le moteur d'un site  tient souvent en deux ou trois fonctions qui pourront être étoffées par la suite.

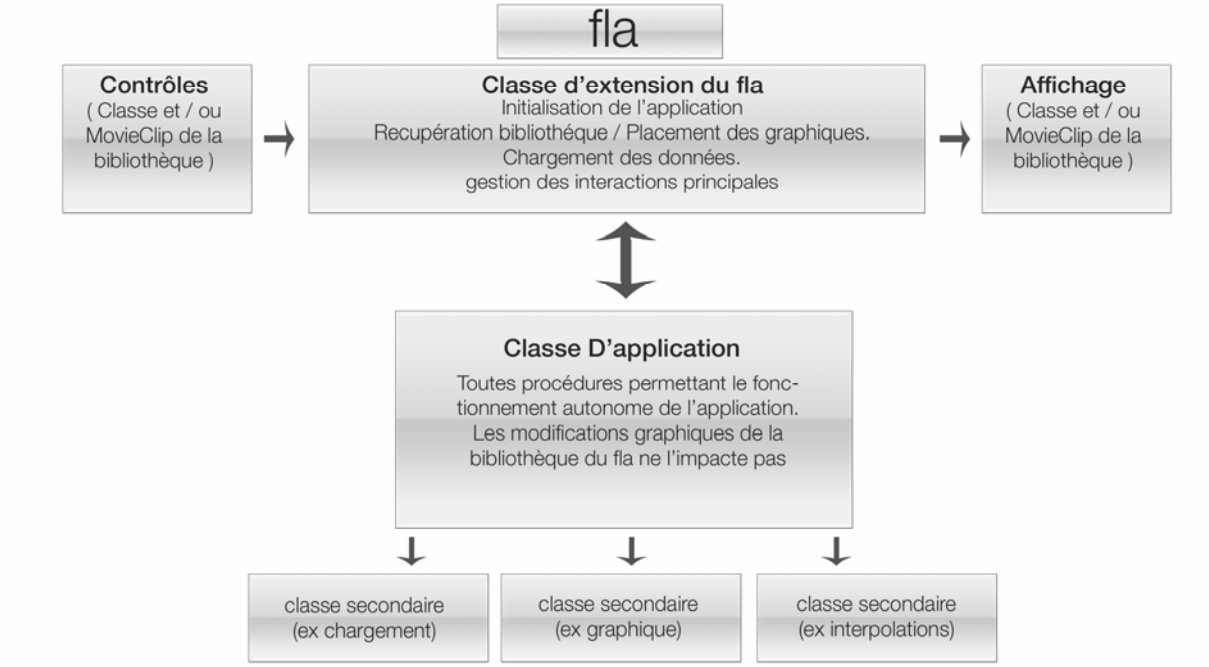
**Le schéma d'imbrication.**

Le schéma d'imbrication consiste à prévisualiser sur le papier comment les éléments de l'application seront placés dans la liste d'affichage afin de garantir une ergonomie maximum pour l'utilisateur  du site et le développeur de l'application.




**Les schémas Objets**

Enfin, la question à se poser lors de la création d'une application c'est qu'est ce que cette application doit pouvoir faire. En faisant la liste des actions possibles, nous pouvons d'ores et déjà nous faire une idée des différentes classes à créer pour lui faire voir le jour.
Pour vous aider à vous faire une idée plus concrète : voici un schéma pour une application générique. Et le découpage des classes tels qu'on pourrait la réaliser en s'aidant de la bibliothèque du fla.



# 2. Organiser son travail, savoir circonscrire ses classes.

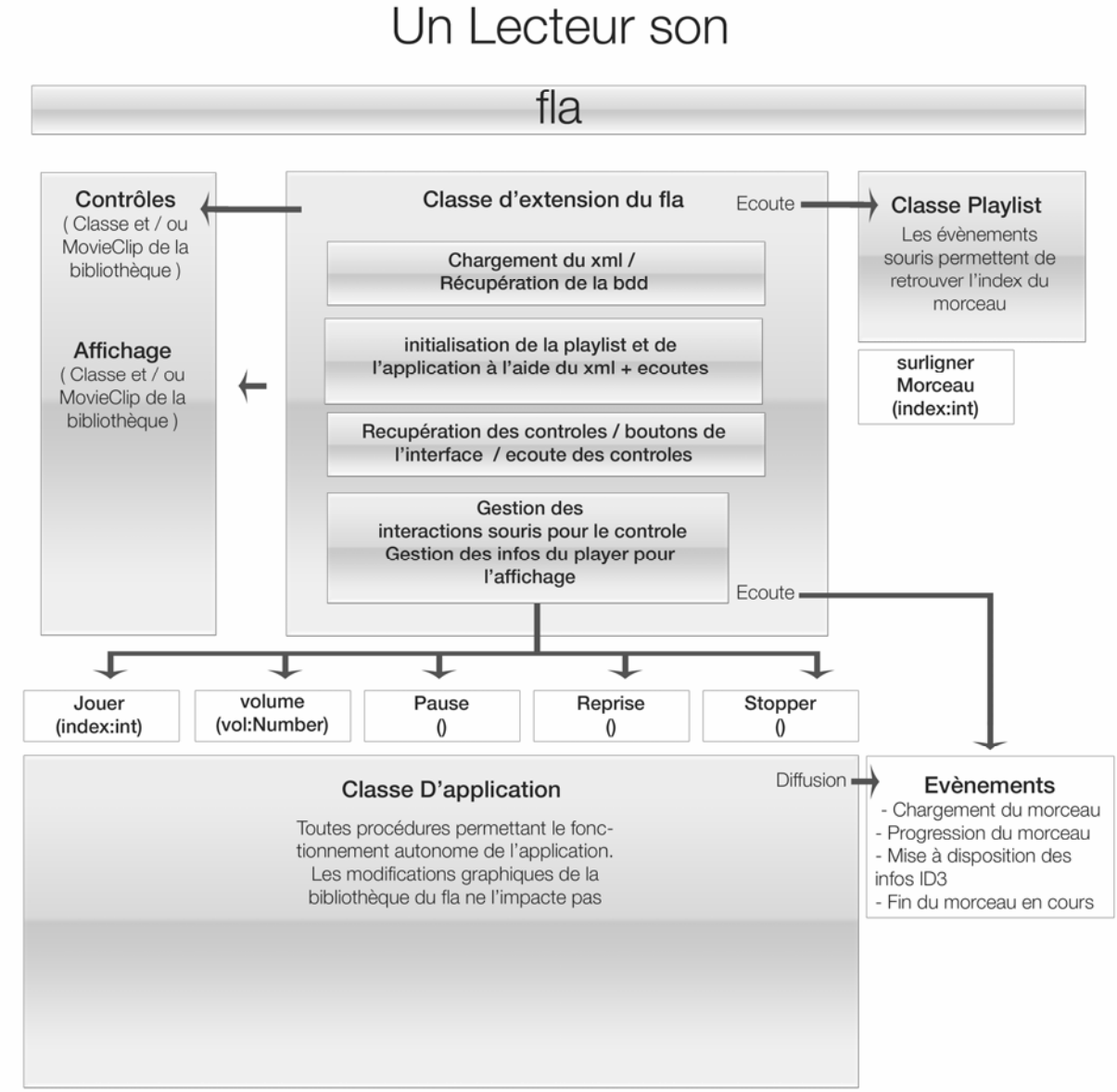
De nombreuses questions concernant la création de classes se posent lorsque l'on commence à s'aventurer dans la programmation orientée objet.
La première d'entre elle est celle de savoir si l'on doit faire une classe pour l'objet ou si il est juste nécessaire d'inclure l'objet dans la classe qui est censé gérer ce processus entre autres.
Plusieurs critères peuvent justifier l'utilisation d'une classe :

La principale règle est de lisser au maximum la longueur de chacune en isolant dans une classe chaque comportement complexe d'une animation. Songez que plus vous isolerez les mécanismes concernant un même processus plus vous aurez de facilité à vous relire. De plus dans FlashDevelop, votre travail sera rendu bien plus efficace grâce à son excellente aide à la programmation.
L'autre idée directrice en direction de l'architecture de classe entre elles est qu'une modification d'un paramètre dans un xml par exemple ne donne pas lieu à des changements dans toutes les classes concernées. Dans ce cadre j'aurai intérêt à standardiser mes informations au travers d'une classe de distribution des infos ou d'une classe permettant d'instancier un objet disposant des infos.

**Architecture des classes**

Ensuite il est utile de savoir ou s'arrête la classe principale et où commencent les classes spécialisées de l'application. La règle ici est relativement simple : Eviter le plus possible la propagation des paramètres optionnels de l'application dans le fonctionnement des classes spécialisées. De cette manière il vous sera beaucoup plus aisé d'utiliser ces classes dans les configurations les plus diverses. Dans le cas d'un lecteur son par exemple. On peut le trouver sous de multiples formes et présentation, malgré cela, ils ont tous en commun la même base. (Jouer, arrêter, modifier le volume…). Il est alors intéressant de prévoir cela afin de ne pas avoir à réecrire une classe à chaque fois.

Par exemple, je veux créer un petit lecteur musical géré par xml externe. Voici à quoi pourrait ressembler le schéma d'application. La classe de gestion du son est bien isolée et pourra être réutilisée en de multiples occasions :




# 3. Organiser son travail, les classes de l'application, les classes permanentes.

On distinguera deux différentes catégories d'utilisation. On pourrait les appeler classes locales et classes globales.

**L'enveloppe et le cœur**

Les premières ne nous seront utiles que dans le cadre de l'application. Il s'agit souvent des classes graphiques qui nous servent à habiller l'animation. Les autres sont des classes usuelles qui pourraient être susceptibles d'être réemployées. Chargeurs et diffuseurs de medias, objets usuels (curseurs, barre de chargement, movieclips étendus permettant la création instantanée de d'objet graphiques longs à programmer)
En effet qu'il s'agisse du logiciel flash ou de FlashDevelop deux possibilités d'importation des classes sont offertes :

- Soit l'on importe une série de classes de manière définitive pour l'utiliser dans le logiciel au fur et à mesure des applications. Nous l'avons déjà employée lors du chapitre sur les interpolations en important la classe tweenMax. Ces classes doivent être stables ou doivent être améliorées pour un compatibilité maximum. Avantage et inconvénient chaque modification à la classe est appliquée à chaque compilation de tout fichier qui l'utilise.

- Soit on importe un ou plusieurs paquetages qui seront accessibles uniquement depuis le fichier en cours.

Ne vous créez pas un problème à propos de savoir dans quelle catégorie ranger la classe que vous créez.
Si vous vous organisez de manière à ce que vos noms de paquetages soient identiques au fil de vos travaux, vous pourrez améliorer une classe d'application en application, en les copiant dans le dossier jusqu'à ce que vous la jugiez éligible à faire partie des classes globales (importées de manière permanente dans le logiciel). C'est ainsi qu'au fur et à mesure vous pourrez vous constituer une bibliothèque d'objets qui pourront vous aider à réduire le temps passé à la programmation.
J'ai par exemple dans mes classes globales :
Des objets graphiques complexes : Curseur générique, générateurs de bouton, Effets de planètes tournantes, formulaires, effets de glissement sur une image…
Des classes procédurales et de stockages : Objets regroupant les informations d'une image, chargeurs (dowloads, uploads) divers, classes statiques regroupant des fonction usuelles (placement, copie d'objet).

Ces manières de faire procèdent de deux stratégies de programmation différentes :

- Soit l'on crée des classes attachées directement au logiciel que l'on modifie de travail en travail. Dans la partie suivante, suivre cette logique fera adopter au programmateur la première voie.

- Soit on crée des classes génériques permettant une utilisation sans modification. Cette voie concerne assez peu d'éléments en regard de l'ensemble du travail d'un programmateur. Si vous travaillez en équipe il est nécessaire de rendre disponible vos outils aux autres pour permettre la compilation. Par ailleurs leur mise en commentaire leur permettra leur utilisation par des tiers en facilitant leur portabilité, mais aussi pour permettra de rentrer dedans après de long mois sans pour autant passer une demi journée à comprendre ce que vous avez voulu faire !

La partie qui suit va essayer de vous présenter les deux logiques de la manière la plus globale possible afin que vous puissiez vous faire une idée, voire de savoir prendre une direction lors de l'organisation des classes d'une application. Nous allons prendre l'exemple de la manipulation d'un XML de données à travers l'ensemble d'une application.

# 4. Les classes non graphiques dans une application Flash

Les classes non graphiques regroupent une partie non négligeable de la programmation AS3. Il s'agit de classe n'étendant pas d'objets graphiques. Donc ne pouvant directement être ajoutée à la liste d'affichage.

L'utilisation d'une classe non graphique s'avère très utile, dès qu'il s'agit de manipuler des données, de les stocker et les distribuer dans une application dynamiques ou encore de gérer des processus complexes, de diffuser des évènements personnalisés.
C'est également un excellent moyen de subdiviser les instructions de la classe principales qui à dejà en charge la gestion de nombreux processus. De cette façon on ôte un nombre conséquent de lignes à celle-ci, ce qui la rendra plus digeste à travailler. Au-delà de 300 lignes la logique et les attributions d'une classe deviennent plus difficiles à comprendre même pour le créateur qui l'ouvre six mois plus tard pour une nouvelle utilisation.

Elles peuvent être uniquement un confort de programmation, comme une collection de fonctions destinés à alléger le codage du placement d'un objet : FlashDevelop est capable d'afficher les méthodes et propriétés de vos classes personnalisées vous gagnez ainsi un temps précieux à utilisés l'aide à la programmation. Une classe clairement identifiée sera d'un accès aisé, plutôt que de transmettre un xml de classe en classe et de risquer de rigidifier son code selon les variations d'identifiants ou de structure.
Elles peuvent aussi s'avérer obligatoires en de multiples occasions pour la gestion de processus internes comme un historique, un gestionnaire de scores, un système de socket pour l'échange des données etc…

**Gestion de données xml dans une application.**

Dès qu'il s'agit de gérer une galerie dynamique ou de gérer la playlist d'un lecteur musical ou vidéo on pourra utiliser un xml. La souplesse de ce media en fait un avantage autant qu'il peut s'avérer problématique si on le propage dans l'ensemble des classes de l'application.
Il est courant alors de centraliser les informations afin que les classes de l'application n'ait pas à utiliser le xml en question.
On peut utiliser pour ce faire deux méthodes :
La première permet de normaliser les infos xml en un objet que l'on instancie, confortable à utiliser au fil des classes.
La seconde est un peu plus pointue mais permet d'avoir comme objet de transmission uniquement un index, objet standard par excellence rendant ainsi possible une parfaite séparation entre les différentes classes de l'application. C'est une solution à privilégier pour un travail en équipe.
Nous évoquons les deux, mais je vous conseille dans un premier temps d'utiliser la première, bien qu'un peu moins souple, mais qui vous permettra de vous familiariser un peu plus avec les classes et la notion d'objet.

**Utiliser une classe de stockage de données.**

Deux stratégies peuvent être employées et même conjuguées en fonctions du cahier des charges concernant les interactions utilisateurs de l'application.

La première - utilisée dans les applications ne nécessitant pas de stockage de données d'interaction. ( Ex : diaporamas ) - consiste à créer une classe capable d'aller chercher chacune des informations d'un xml interne ou externe, Puis de distribuer chacune des propriétés grâce à un système de méthodes statiques.

Voici la structure d'un xml chargeant un diaporama et donc chaque image est cliquable et renvoi vers un lien.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<main>
	<plages path="musique/">
		<piste url="01.mp3" artiste="Jaques Brel" titre="Les Bourgeois" />
		< piste url="02.mp3" artiste ="Arthur h " titre ="Negresse blanche" />
		< piste url="03.mp3" artiste ="Massive Attack" titre ="Sly" />
		< piste url="04.mp3" artiste ="John coltrane" titre ="wise one" />
	</plages>
</main>
```

### Stockage dans un objet

Stocker un objet auquel on attribue les propriétés du xml est d'une part la manière la plus simple d'envisager ce type de fonctionnement. En effet ici chaque info xml devient un objet facile à manipuler
Et une façon de voir la p.o.o. dans son fonctionnement le plus basique. C'est donc un bon moyen de débuter.

Cette méthode consiste à créer un classe d'objet dont chaque occurrence stockera les propriétés et nœuds textes de chacune des lignes du xml.
Dans certains cas, elle peut être utilisée conjointement avec la classe xml… Cela permet de profiter d'une assistance au code maximum mais aussi de créer des méthodes de gestion de données intégrées…
Une telle classe pourra avoir cet aspect :

```
package
{
	public var index : int = -1;
	public var lien_fichier : String;
	public var titre : String;
	public var artiste : String;

	// dynamic permet de pouvoir ajouter une propriété à l'objet à la volée sans
	// qui n'existe à l'origine.
	public dynamic class BadgeSon
	{

		public function BadgeSon()
		{

		}

	}

}
```

Ainsi si un changement doit intervenir dans ce xml il sera juste nécessaire de modifier l'adressage dans cette boucle. Au lieu d'aller dans chacune des classes ou les infos xml interviennent. Nous allons constituer un tableau avec un objet typé qui aura pour avantage de proposer toujours la même variable pour la même valeur.

```
for each(var donnee : * in donnees_xml.plages.*)
{
	var badge :BadgeSon = new BadgeSon() ;
	badge.titre = donnee.@titre;
	badge.fichier = donnees_xml.plages.@path+donnee.@url;
	badge.artiste = donnee.@artiste;
```

```
                        tab_pistes.push(badge) ;
                }
```

Le parcours du xml aura pour effet de classer les données dans le même ordre. Ainsi la première valeur du tableau, à l'index 0  correspondra au premier nœud xml stockant votre galerie ou votre playlist.

**Stockage et gestion depuis une classe de parcours du xml :**

C'est une autre manière d'envisager la centralisation et la normalisation des données. Ici nous créeons une classe statique s'occupant de charger le xml, puis de le parcourir au travers de ses fonctions en lui fournissant un numéro d'index correspondant à l'ordre du xml.
Etant une classe statique on peu aussi l'utiliser dans toute l'application. Il peut ainsi être utilisé sans se contraindre à isoler les classes les unes des autres

/*Classe Statique de chargement et gestion du XML */

```
package {

	import flash.events.Event
	import flash.geom.Rectangle;
	import flash.net.URLLoader;
	import flash.net.URLRequest;

	public final class XMLManager {

		//XML dans lequel va être stocké le contenu XML du fichier chargé
		public static var dataXML:XML;
		 //Loader qui va charger le XML
		public static var loader:URLLoader;
		private static var diffuseur:EventDispatcher;
		/*-- Fonctions de chargement des données --*/

		//Fonction de chargement du XML de base
		public static function load(purl:String, pdiffuseur
:EventDispatcher):void
		 {
			Diffuseur = pdiffuseur ;
			loader = new URLLoader(new URLRequest(purl));
			//création du loader et chargement des données
			loader.addEventListener(Event.COMPLETE, loadComplete);
			//déclenché à la fin du chargement du XML
		}

		//Fonction déclenchée à la fin du chargement du XML
		private static function loadComplete(even:Event):void
		{
			dataXML = new XML(even.target.data);
			diffuseur.dispatchEvent(even)
		}
		/*-- Fonctions Publiques de Données de musique--*/

		/*-- fonction renvoyant le nombre d'images contenu dans le XML--*/
		public static function get imgs():int {
			return dataXML.plages.piste.length();
		}

		//fonction renvoyant le chemin des images
		public static function get chemin():String {
			return dataXML.plages.@path
```

```
		}

		//fonction renvoyant l'url complète (path+url) de l'image
		public static function getURL(nb:int):String {
			return chemin + dataXML.plages.piste[nb].@url
		}

		public static function getTitre(nb:int):String
		{
			return dataXML.plages.piste [nb].@titre;
		}

		public static function getArtiste(nb:int):String
		{
			return dataXML.plages.piste [nb].@artiste;
		}


	}
}
```

Nous voyons donc que ces deux méthodes peuvent produire les mêmes effets sans toutefois procéder de la même méthode. C'est ensuite une question de pratique assidue de souhaiter appliquer l'une ou l'autre solution.
Toutefois l'objet contient pour moi un avantage primordial pour certaines utilisations, celui de pouvoir embarquer beaucoup plus que les données du xml. Par exemple, pour un chargement d'image mon objet peut également stocker les données d'image. Ainsi le chargement du swf ou du bitmap associé à l'objet n'aura à être chargé qu'une fois. Si il est déjà chargé on évite le lourd processus du loader.