Partie 1 : Autour du modèle évènementiel

1 Les évènements dans flash.

Asynchronisme et évènements

L'un des atouts de flash est en même temps l'une de ses difficultés. Contrairement au html Actionscript est un langage asynchrone. C'est-à-dire qu'il n'attend pas que certains processus (comme les chargements externes) se terminent pour poursuivre l'exécution du code. Les actions souris pouvant déboucher sur bien plus que le renvoi vers une nouvelle url, elles doivent être entièrement construites également. Certains autres permettent de mettre en oeuvre des intervalles de temps pour créer du mouvement ou des décomptes. Les objets liés aux médias diffusés comme le son et la vidéo émettent eux aussi des signaux permettant de connaître leur état. Ce en effet sont des processus qui ont besoin d'être maîtrisés dans leur déroulement. Pour tout ces processus, l'actionscript propose un système unifié, le modèle évènementiel.

Il met en œuvre :

- un objet diffuseur
- Un objet décrivant un type d'évènement
- un écouteur placé sur ce diffuseur.
- Une fonction qui sera déclenchée par la réception.
-

Différents évènements sont produits par divers objets, s'y retrouver en représente la plus grande difficulté.
Pour s'aider dans cette tâche, la documentation du langage renseigne les différents évènements produits par les différents objets présents dans le langage. Il y a aussi le regroupement logique : tous les éléments pouvant présenter une interactivité diffusent des évènements souris. Dans ces éléments on retrouve : TextField, MovieClip, Sprite, SimpleButton. On sait donc que tous ces objets sont susceptibles d'accueillir un écouteur d'évènement souris. De même on s'imagine bien que les évènements de chargement émettent les éléments nécessaire à leur bon fonctionnement : départ, progression, fin du chargement, erreur d'url…

Le système évènementiel est la clé de voûte qui va permettre d'ajouter de l'interactivité à vos animations, de les alléger en vous servant de chargements externes, etc… Autant dire que ce chapitre est le plus important. Pour ainsi dire même, la maîtrise du langage tient à la maîtrise de ce système.

Utilisation du système évènementiel

Un évènement est un objet composite tout comme peut l'être le type MovieClip ou Sprite, cet objet est automatiquement instancié si l'action qu'il décrit est déclenchée. Puis, il est diffusé au travers d'un diffuseur d'évènement EventDispatcher qui est un autre objet composite au travers de sa méthode dispatchEvent (événement). Pour plus de commodité l'objet EventDispatcher est directement intégré aux objets interactifs que nous avons cités plus haut, nous en verrons la raison par la suite.

Pour utiliser ce système il suffit de placer un écouteur d'évènement sur l'objet diffuseur au travers de la méthode :

objet_a_ecouter.addEventListener(Evt_a_ecouter , fonction_a_declencher)

Pour créer une interaction dans flash, il faut donc effectuer ces trois choses :

- Sélectionner un objet qui diffuse l'événement recherché.
- Abonner un écouteur sur cet objet.
- Créer une fonction qui traitera l'évènement s'il est déclenché.

Tous les systèmes permettant les interactions ou diffusant des informations dans flash sont basés sur cette même formule. Il existe bien une ou deux exceptions basées sur les processus en vigueur dans les précédentes versions (diffusion vidéo, moteurs d'interpolation tierces), que nous évoquerons en temps utiles.

Ce qu'il faut retenir ici c'est simplement le trio : *écouteur, diffuseur, fonction à déclencher.*

Construction d'une fonction évènementielle.

Nous avons sur la scène un MovieClip auquel nous avons donné « mc_lien » comme nom d'occurrence. On souhaite le rendre cliquable et lancer un lien vers une autre page du site.

```
// Cette commande fait apparaître la main lorsque la souris passe sur le
// clip
mc_lien.buttonMode = true ;
// écoute de l'évènement souris « clic » sur le clip
mc_lien.addEventListener(MouseEvent.CLICK,envoiLien)

function envoiLien ( even : MouseEvent) : void
{
      // Mise en place d'une requète url ( chaque processus flash, faisant
      // Appel à une ressources externe utilise cet objet
      var requete : URLRequest = new URLRequest("http://www.google.com");

      // fonction générique pour l'appel d'une nouvelle page.
      navigateToURL(requete, "_blank");
}
```

Exercice : Mettez en place un menu de trois boutons avec un trace dans chaque fonction.

Regardons maintenant notre fonction évènementielle un peu plus en détail, car elle répond elle aussi à un schéma très strict.
Une fonction liée à un évènement doit TOUJOURS avoir un paramètre en parenthèse : Ce paramètre est l'instance de l'objet évènementiel diffusé par l'élément écouté (ici l'objet est de type MouseEvent). Il contient des propriétés et méthodes qui permettent d'obtenir des informations concernant l'objet écouté et la nature de l'évènement (comme par exemple connaître les coordonnées locales de la souris sur l'objet lors du clic).
Le paramètre évènementiel fait aussi toujours référence à l'objet qui l'a émis au travers de la propriété target.
Une fonction évènementielle ne peut avoir qu'UN SEUL paramètre. En l'occurrence celui que nous venons de voir. Pour utiliser des variables non déclarées dans la fonction il faudra que celles-ci soit déclarée à l'extérieur du corps des fonctions (soit en début de script image, soit dans une classe pourvu de la déclaration « public » ou « private ». Sachez qu'il existe aussi la possibilité de créer ses propres évènements permettant de faire transiter des données lorsque la solution précédente n'est pas applicable. Nous aurons l'occasion de nous pencher sur les classes d'évènements personnalisés dans la deuxième partie du cours.

Une fonction évènementielle ne peut renvoyer aucune valeur. L'utilisation du mot clé « return » en fin de fonction provoquera une erreur. Cela est toutefois logique puisque l'appel de la fonction est effectué automatiquement et qu'il n'existe de ce fait aucune façon de récupérer une éventuelle valeur issue de l'exécution du code.

Si vous vous posiez encore la question de savoir ou et comment placer variables et autres fragments de codes, les quelques règles que nous venons de voir sont déterminantes. Ce sont souvent celles-ci, en fait, qui imposeront de placer une variable localisée dans une fonction ou à la racine du code classe ou scénario. C'est le point d'entrée vers l'architecture de votre application.

Stopper l'écoute d'un évènement.

Nous avons besoin de stopper l'écoute d'un évènement lorsque nous ne voulons plus que la fonction soit déclenchée ou lorsqu'il devient nécessaire de supprimer l'objet qui diffuse l'évènement. Pour ce dernier cas il est nécessaire de connaître un peu mieux le fonctionnement du lecteur flash : Pour ce programme, un élément ne peut être supprimé de la mémoire que si plus aucun évènement n'y est attaché. Sinon, la trace de l'objet reste en mémoire et si il est lourd graphiquement, il peut ralentir voire compromettre le bon fonctionnement de l'application. Dans le principe tout écouteur placé doit pouvoir être retiré avant suppression de l'objet sur lequel il est placé. Nous utilisons pour cela la méthode présente sur l'objet EventDispatcher : diffuseur.removeEventListener(type_evenement, fonction_evenementielle).

```
// nous créons le MovieClip cette fois ci par le code :
var mc_lien :MovieClip=new MovieClip();

mc_lien.graphics.beginFill(1,1);
mc_lien.graphics.drawRect(0,0,200,40);

mc_lien.addEventListener(MouseEvent.CLICK, envoiLien);
mc_lien.buttonMode = true ;
addChild(mc_lien);


function envoiLien (even :MouseEvent):void
{
var requete : URLRequest = new URLRequest("http://www.google.com");
navigateToURL(requete, "_blank");
// retrait de l'écouteur.
mc_lien.removeEventListener( MouseEvent.CLICK , envoiLien);
// retrait du comportement main du pointeur souris
mc_lien.buttonMode = false ;
}
```

L'exemple ici montre comment rendre un bouton cliquable une seule fois. Le clic déclenche la fonction qui renferme l'instruction de désabonnement et la désactivation du pointeur « main » sur le clip. En outre si nous supprimons ce clip il sera effectivement éligible à la suppression totale du cache de l'animation.

Savoir se servir de propriétés et méthodes du paramètre de la fonction évènementielle.

Il reste un élément du système évènementiel que nous n'avons pas abordé : Il s'agit de la variable « even » inscrite entre les parenthèses de notre fonction.
Nous savons donc qu'il s'agit de l'objet instancié lorsque l'action à été déclenché. Cet objet contient des propriétés et méthodes communes à l'ensemble des objets

évènementiels.
En premier lieu faisont connaissance avec target.
target est une propriété partagée par tous les évènements. C'est un raccourci vers l'objet qui à diffusé l'évènement.
Donc lorsque nous utilisons la propriété target du paramètre de notre fonction nous accédons dans l'exemple suivant au clip sur lequel il faut cliquer.

```
function envoiLien (even :MouseEvent):void
{
	var requete : URLRequest = new URLRequest("http://www.google.com");
	navigateToURL(requete, "_blank");
	even.target.removeEventListener( MouseEvent.CLICK , envoiLien);
	even.target.buttonMode = false ;
}
```

D'autres propriétés en revanche sont propres au type d'évènement en cours. L'objet MouseEvent contient de nombreuses informations, telles que les coordonnées de la souris sur l'objet ou la scène au moment du clic. Pour permettre les combinaisons de touches, il est possible encore de savoir si la touche « ctrl »(ou « command »), « alt » ou « maj » sont enfoncées .

Les évènements clavier.

Parmi les évènements utiles dans flash on trouvera aussi les évènements clavier. Moins utilisés. Ils peuvent trouver néanmoins de nombreuse application, de la présentation d'entreprise au jeu de plateforme en passant par les fonctions cachées . On se sert usuellement de deux des propriétés de l'objet transmis à la fonction.

Exemple : gestion clavier d'une progression dans le scénario d'un clip.

On utilise ici la propriété keyCode (code de la touche) de l'évènement. On se sert également de la classe Keyboard qui répertorie les actions claviers sur les touches spéciales de l'ordinateur. Ici nous avons un clip dont le déroulement est ponctué de stop() au long de la timeline. Nous cherchons à la faire progresser si l'on appuie sur la fléche du bas :

```
// on écoute l'objet stage pour les évènements clavier.
stage.addEventListener(KeyboardEvent.KEY_DOWN,gestionGraphique);


function gestionGraphique(even:KeyboardEvent):void
{
	// on compare la propriété keyCode à la touché concernée stockée
	// dans la classe Keyboard
	if(even.keyCode == Keyboard.DOWN)
	{
		graphique.play();
	}
}
```

Exemple reconnaissance d'une combinaison de touches :

Ici, c'est une autre propriété : charCode (code du caractère) qui va nous aider à déterminer précisément quel caractère inscriptible du clavier donne lieu à l'évènement.

Pour être traduit en une chaîne de caractère nous allons avoir recours à une fonction statique de l'objet string dédiés à cela : `String.fromCharCode(even.charCode)`

Nous nous servons également de deux autres propriétés de l'évènement permettant de savoir si la touche « ctrl » ou « maj » sont enfoncées simultanément au moment de l'événement. Pour résultat la condition ne se déclenche que si la combinaison de touche « shift – ctrl – e » est détectée.

```
stage.addEventListener(KeyboardEvent.KEY_DOWN, ecouteClavier);

function ecouteClavier(even:KeyboardEvent):void
{

	var code:String=String.fromCharCode(even.charCode)

	if (even.shiftKey)
	{
		if (even.ctrlKey)
		{
			if (code == "E")
			{
				trace("combinaison de touché détecte")
			}
		}
	}
}
```

Exercice : créez à partir de ces codes un objet avec un déplacement commandé par le clavier

Les évènements redondants : EnterFrame et Timer. Créer un mouvement à l'aide du modèle évènementiel.

Certains évènements, tels qu'un clic souris, le signal de fin de chargement d'une image externe sont envoyés une seule fois à chaque fois que l'utilisateur actionne le bouton gauche de sa souris. D'autres sont diffusés plusieurs fois et déclenchent à chaque fois la fonction qui lui est associée. Certains d'entre eux sont même diffusés de façon régulière. Ces évènements s'avèrent très utiles pour crées du mouvement, ou un déroulement dans le temps. Nous allons en découvrir un des plus courants : L'événement ENTER_FRAME :
La diffusion de l'évènement ENTER_FRAME est assurée par les MovieClip et les objets héritant de MovieClip à savoir la scène de notre animation. Le comportement de cet évènement est d'envoyer un signal à chaque fois que la tête de lecture du clip entre dans une nouvelle image. Attention, ce comportement n'est pas dépendant de l'état de lecture du clip. Il envoie donc cet évènement même si le clip est stoppé.
L'évènement ENTER_FRAME est dépendant de la cadence de l'animation. Une animation cadencée à 12 images par secondes déclenchera la fonction associée 12 fois en une seconde. Ce comportement nous permet donc de créer un défilement dans le temps à cadence rapide… Donc, du mouvement bien sur, mais aussi des vérifications redondantes pour des jeux ou d'autres applications à haute interactivité. Grâce à cette fonction on entre de plein pied dans ce qu'il n'est pas possible de réaliser en s'appuyant sur la ligne de scénario de flash.

Timer

Si l'on cherche à créer un mouvement qui soit indépendant de la cadence de flash on orientera son choix sur la classe Timer en effet l'objet timer fonctionne selon une horloge interne

configurée pour envoyer un évènement de type TimerEvent.TIMER toute les x millisecondes. La cadence donnée au fichier n'a par conséquent aucune influence sur celui-ci.
L'objet Timer est nettement plus souple qu'un gestion par Event.ENTER_FRAME en effet, il dispose de fonction permettant de démarrer ou stopper le compte, il est aussi possible de modifier la frequence de l'horloge en cours de fonctionnement, permettant par exemple d'augmenter la vitesse de lecture d'un clip en asservissant sa lecture au fonctionnement de l'horloge.

Remplacer le curseur de la souris par un clip

Sur un nouveau fichier nous importons une image que nous insérons dans un clip. Nous le posons sur la scène avec « image » comme nom d'occurrence.
Nous créons un second clip contenant une forme vectorielle créée dans l'environnement auteur un cercle rempli avec un dégradé radial. La couleur est sans importance puisque la forme nous servira de masque. En revanche, il faut que la couleur soit à alpha = 100 en son centre et 0 sur les bords. Nous déposons également ce clip sur la scène avec « lorgnette » comme nom d'occurrence.
Nous créons un calque « actions » dans lequel nous écrivons ce code.

```
// Ces deux lignes sont indispensables pour que le dégradé alpha soit pris
//en compte. cacheAsBitmap sert à ce que le lecteur transforme la forme
//vectorielle en tableau de pixels.
lorgnette.cacheAsBitmap = true;
image.cacheAsBitmap = true;
// Masquage de l'image par la forme
image.mask=lorgnette;
// nous rendons le pointeur de la souris invisible
Mouse.hide();
// le scénario principal étant du type MovieClip on peut placer l'écouteur
// sur celui-ci
this.addEventListener(Event.ENTER_FRAME,pointer)

function pointer (even:Event)
{
      lorgnette.x=mouseX;
      lorgnette.y=mouseY;

      // les deux lignes suivantes peuvent remplacer les deux précédentes.
      // cela permet de créer un effet de retard sur le mouvement du
      masque :
      //lorgnette.x= mouseX*0.4 + lorgnette.x*.6;
      //lorgnette.y= mouseY*0.4 + lorgnette.y*.6;
}
```

Voici le même code commandé par une instance de la classe Timer. Le code devient alors insensible à la cadence de l'animation :

```
lorgnette.cacheAsBitmap = true;
image.cacheAsBitmap = true;
image.mask=lorgnette;
//création d'une instance de la classe timer
var horloge:Timer = new Timer(40);
// Ajout d'un ecouteur d'évènement sur cette classe
horloge.addEventListener(TimerEvent.TIMER,pointer);
// départ du compte
horloge.start();
```

```
Mouse.hide();
// la fonction pointer reçoit mainetant un évènement de type TimerEvent
function pointer (even:TimerEvent)
{
      lorgnette.x= mouseX*0.4 + lorgnette.x*.6;
      lorgnette.y= mouseY*0.4 + lorgnette.y*.6;
}
```

L'ensemble des objets de type évènement auront toujours des propriétés liées à leur nature évènementielle. Les événements diffusés par une interpolation permettent d'en contrôler très finement le fonctionnement et même de les combiner en diffusant des signaux au début, fin, et à chaque changement de valeur de l'objet interpolé.

Charger une image ou un swf externe.

Le chargement d'une image ou d'un fichier swf externe est une opération très courante. Grâce à ce système vous pourrez mettre en place des sites entiers en alliant un système de chargement externes aux menus que vous devriez savoir mettre en place arrivé à ce point du cours.
Tout d'abord il est important de souligner que le chargement d'un media est susceptible de diffuser un nombre important d'évènements tels que le début, la progression, le signalement d'une erreur de chargement, la fin du chargement et quelques autres encore. Nous allons appréhender la base du chargement externe au travers du chargement simple d'une image et d'un swf.
Ce premier code utilise uniquement la classe d'évènement Event. Sachez toutefois que le signalement d'erreur de chargement est géré par le type d'objet IOErrorEvent, et que les signaux de progression qui nous permettront de mettre en place une barre de chargement est assuré par la classe ProgressEvent.

Nous faisons la connaissance de l'objet Loader: C'est l'objet qui nous sert à charger le media. Il fait partie des objets affichables dans flash. Cela à plusieurs implications on peut directement l'utiliser pour l'affichage du graphique chargé. Il peut être réutilisé pour charger d'autres images. Il contient en outre une propriété content qui permet d'extraire le contenu chargé.
celui-ci est équipé d'un diffuseur d'évènement (loader.contentLoaderInfo) qui va servir à émettre tout les signaux du téléchargement. Donc ici le diffuseur et l'objet ne sont pas une seule et même chose.
En nous intéressant à la documentation nous nous apercevons que deux évènements peuvent correspondre à notre attente :

Event.complete
Dispatched when data has loaded successfully. (diffusé une fois le chargement terminé)

Event.init
Dispatched when the properties and methods of a loaded SWF file are accessible. (diffusé une fois que le swf chargé est disponible).

La différence entre les deux réside dans le fait que l'évènement init assure qu'un swf ou une image permettront toutes les propriétés et méthodes dont dispose un movieClip ou un bitmap. Nous choisirons donc d'écouter cet évènement.

Nous allons abonner un écouteur qui réagira lorsque l'évènement INIT sera émis par l'objet contentLoaderInfo. Il sera alors possible de le redimensionner, si c'est un bitmap de le lisser en cas de redimensionnement, de le centrer dans un container puis in fine de l'afficher sur la scène.

```
var container_principal:MovieClip=new container();
addChild(container_principal);
var requete:URLRequest=new URLRequest("VOTRE_IMAGE_OU_SWF");
var chargeur:Loader=new Loader();
chargeur.contentLoaderInfo.addEventListener(Event.INIT,chargementTermine);
function chargementTermine(even:Event)
{
container_principal.addChild(chargeur);
//centrage du media
chargeur.x= container_principal.width / 2 – chargeur.width / 2;
chargeur.y= container_principal.height / 2 – chargeur.height / 2;
chargeur.contentLoaderInfo.removeEventListener(Event.INIT,chargementTermine
);
}
```

Gérer le chargement une image ou un swf externe.

Comme je l'ai évoqué, le chargement d'une image émet plus d'évènements que ceux que nous avons vus. Ces évènement peuvent nous permettre en particulier d'éviter des erreurs qui bloqueraient notre application si le chargement ne peut s'effectuer ou d'afficher une animation de chargement. Tous les évènements sont diffusés au travers de l'objet contentLoaderInfo contenu par l'instance de notre loader. C'est donc lui qu'il faut cibler avec des écouteur approprié)

A / Gestion des erreurs.

Un problème de chargement (connexion coupée, mauvais lien) peut mettre en péril la stabilité et le bon déroulement d'une image. Aussi prévoir ces types de comportement peut nous permettre d'afficher quand même un symbole par défaut, de la même manière qu'en html. Pour ce faire il est utile de prévoir dans son animation un clip que nous pourrons mettre en place par le script en l'exportant au travers de son panneau de liaison. Nous le ferons passer pour l'image chargée au cas où une erreur est émise. Dans ce cas deux objet peuvent apparemment nous servir. Le premier « flash.events.HTTPStatusEvent » informe le lecteur flash sur la présence ou l'absence de connexion internet. Il est émis dans l'un ou l'autre cas. Il contient la propriété status qui envoie le code de connexion http correspondant. Nous n'avons pas un réel besoin d'écouter cette évènement dans le cadre de notre application.
Le lecteur en cas de lien brise ou d'image corrompue envoie également un évènement qui spécifie l'erreur : flash.events.IOErrorEvent.IO_ERROR
En écoutant cet évènement il nous est possible d'ajouter une gestion des erreurs :
Nous créons un clip dans la bibliothèque avec « container » comme nom de liaison. Il contient éventuellement un décor.

```
var container_principal:MovieClip=new container();
addChild(container_principal);
var requete:URLRequest=new URLRequest("images/wild.jpg");
var chargeur:Loader=new Loader();
chargeur.contentLoaderInfo.addEventListener(Event.INIT,chargementTermine);
chargeur.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR,
erreurChargement);
function erreurChargement (even: IOErrorEvent):void
{
// retrait des ecouteurs
chargeur.contentLoaderInfo.removeEventListener(Event.INIT,chargementTermine
);
chargeur.contentLoaderInfo.removeEventListener(IOErrorEvent.IO_ERROR,erreur
Chargement);
```

```
// instanciation d'un clip destiné à remplacer l'image manquante
var img_defaut : MovieClip = new ImageDefaut();
container_principal.addChild(img_defaut);
// j'envoie un signal de fin de chargement
chargeur.contentLoaderInfo.dispatchEvent(new Event(Event.INIT));
}
function chargementTermine(even:Event):void
{
container_principal.addChild(chargeur);
//centrage du media
chargeur.x= container_principal.width / 2 – chargeur.width / 2;
chargeur.y= container_principal.height / 2 – chargeur.height / 2;
chargeur.contentLoaderInfo.removeEventListener(Event.INIT,chargementTermine
);
chargeur.contentLoaderInfo.removeEventListener(IOErrorEvent.IO_ERROR,erreur
);
}
```

Note : Dans la fonction « erreurChargement » nous diffusons un évènement INIT destiné à enclencher un nouveau chargement dans le cas d'un chargement par lot, comme celle d'une galerie. Dans ce cas nous pourrions écouter l'évènement init afin de déclencher le chargement suivant.

A Création et gestion d'une animation de chargement.

Le chargement d'une image lourde (le lecteur flash permet la manipulation d'un bitmap de 2880 pixels de cotés) ou d'un swf contenant des élément bitmap peut prendre un certains temps. Il peut être bienvenu dans ce cas d'afficher une animation de chargement ou d'attente afin de ménager la patience du visiteur.
Pour ce faire nous aurons besoin d'écouter deux évènements supplémentaires, et d'ajouter une instruction (disparition de l'animation) à notre fonction de fin.
Event.OPEN nous informe du debut du chargement de l'image. Nous insererons donc un ecouteur qui se chargera dans la fonction d'afficher et de centrer notre animation.
ProgressEvent.PROGRESS quand à lui est une classe d'évènement qui permet en consultant ses propriétés de connaître le nombre de bytes chargés. Nous connaissons également à travers lui le nombre de bytes à charger. En réalisant une simple opération nous pouvons obtenir un decimal entre 0 et 1 que nous pouvons appliquer à l'échelle horizontale (scaleX) d'un clip contenant un graphique en forme de barre. L'évènement est déclenché à chaque chargement d'un nouveau paquet de donnée. Il peut donc créer l'illusion d'une barre animée.
// un clip contenant un rectangle montrant la barre de chargement dans son état final est créé dans la bibliothèque (contient un champ texte : « champ_texte » et le clip du graphique : « barre ».
On lui donne « BarreChargement » comme identifiant de liaison.

```
var barre_load : MovieClip= new BarreChargement();
var container_principal:MovieClip=new container();
addChild(container_principal);
var requete:URLRequest=new URLRequest("images/wizld.jpg");
var chargeur:Loader=new Loader();
chargeur.contentLoaderInfo.addEventListener(Event.INIT,chargementTermine);
chargeur.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR,erreurCha
rgement);
// ajout d'une ecoute sur le commencement et la progression du chargement
chargeur.contentLoaderInfo.addEventListener(Event.OPEN,departChargement);
chargeur.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,chargeme
ntEnCours);
function departChargement(even:Event):void
{
```

```
	//centrage de la barre de chargement par rapport à son état final
	barre_load.x= container_principal.width / 2 – barre_load.width / 2;
	barre_load.y= container_principal.height / 2 – barre_load.height / 2;
	// reduction de l'echelle de la barre à son état minimal
	barre_load.barre.scaleX=0;
	//ajout à la scène
	container_principal.addChild(barre_load);
}
function chargementEnCours(even:ProgressEvent):void
{
	var ratio_chargement:Number= even.bytesLoaded / even.bytesTotal;
	var pourcent:Number= Math.round(ratio_chargement * 100);
	var texte:String = pourcent +" %";
	barre_load.champ_texte.text = texte;
	barre_load.barre.scaleX = ratio_chargement;
}
function erreurChargement (even: IOErrorEvent):void
{
	// retrait des ecouteurs
	chargeur.contentLoaderInfo.removeEventListener(Event.INIT,chargementT
	ermine);
	chargeur.contentLoaderInfo.removeEventListener(IOErrorEvent.IO_ERROR,
	erreurChargement);
	chargeur.contentLoaderInfo.removeEventListener
	(Event.OPEN,departChargement);
	chargeur.contentLoaderInfo.removeEventListener
	(ProgressEvent.PROGRESS,chargementEnCours);
	// instanciation d'un clip destiné à remplacer l'image manquante
	var img_defaut : MovieClip = new ImageDefaut();

	container_principal.addChild(img_defaut);

	// j'envoie un signal de fin de chargement
	chargeur.contentLoaderInfo.dispatchEvent(new Event(Event.INIT));

}

function chargementTermine(even:Event):void
{
	// retrait de la barre de chargement :
	container_principal.removeChild(barre_load);
	//centrage du media
	chargeur.x= container_principal.width / 2 – chargeur.width / 2;
	chargeur.y= container_principal.height / 2 – chargeur.height / 2;
	container_principal.addChild(chargeur);
	chargeur.contentLoaderInfo.removeEventListener(Event.INIT,chargementT
	ermine);
	chargeur.contentLoaderInfo.removeEventListener(IOErrorEvent.IO_ERROR,
	erreur);
	chargeur.contentLoaderInfo.removeEventListener
	(Event.OPEN,departChargement);
	chargeur.contentLoaderInfo.removeEventListener
	(ProgressEvent.PROGRESS,chargementEnCours);
}
```

C/ optimisation du swf chargé (scaling)ou de l'image (lissage).

Nous allons compléter notre code par un processus qui va redimensionner notre média et lui appliquer un lissage si nous détectons qu'il s'agit d'un bitmap.

Le redimensionnement doit tenir compte du format vertical ou horizontal du media chargé. Nous devrons donc tenir compte de la hauteur et largeur.
Le redimensionnement ne s'applique que si le medias est PLUS grand que le conteneur. Aussi nous aurons besoin de savoir si la hauteur de l'image depasse la hauteur du cadre et ainsi de suite.
Nous nous servons pour cela de la classe math qui contient plusieurs méthodes de comparaison de valeur numérique. L'une Math.max(valeur_1, valeur_2) renvoie la valeur la plus grande. Math.min(valeur_1, valeur_2) renvoie la valeur la plus petite. C'est celle-ci dont nous avons besoin pour notre comportement de réduction. Nous emploierons des ratios de manière à modifier l'échelle plus simplement.

```
var ratio_largeur:Number= (container_principal.width-30 )/ chargeur.width;
var ratio_hauteur:Number= (container_principal.height-30) /
chargeur.height;
var ratio_final:Number= Math.min(ratio_largeur,ratio_hauteur);
if(ratio_final<1)
chargeur.scaleX=chargeur.scaleY=ratio_final;
```

Le lissage est uniquement utilisable sur les objets de type Bitmap. Aussi, si nous voulons garder la capacité de charger aussi des swf, nous aurons besoin d'utiliser une condition en nous servant du mot clé du mot clé « is » permettant de déterminer si l'instance est de tel ou tel type. Pour effectuer ce test nous aurons besoin de la propriété content de notre instance loader qui contient effectivement l'objet chargé pour la comparer au type bitmap.
if (chargeur.content is Bitmap)

Si tel est le cas nous aurons besoin de transformer le contenu en un objet bitmap à par entière.

```
if (chargeur.content is Bitmap)
{
      var bmp:Bitmap = Bitmap(chargeur.content);
      bmp.smoothing = true;
}
```

Ce procédé est nécessaire lorsque nous manipulons un objet que nous savons être d'un type, mais que flash traite pour raison pratique sous une forme plus générique dont dépend l'objet en question. Par exemple tout objet imbriqué dans un clip peut être récupéré avec la fonction getChildByName. L'objet renvoyé l'est sous la forme DisplayObject dont dépendent l'ensemble des objets affichable. Pour accéder à la propriété text d'un champ texte par ce biais nous aurions également besoin de le transtyper.
Voici le code intégré à notre fonction de fin de chargement :

```
function chargementTermine(even:Event):void
{
// Retrait de la barre de chargement :
container_principal.removeChild(barre_load);
// extraction du contenu :
var contenu : DisplayObject = chargeur.content;
// Redimensionnement
var ratio_largeur:Number= (container_principal.width-30 )/ contenu.width;
var ratio_hauteur:Number= (container_principal.height-30) / contenu
.height;
var ratio_final:Number= Math.min(ratio_largeur,ratio_hauteur);
if(ratio_final<1)
contenu.scaleX = contenu.scaleY = ratio_final;
// Lissage si bitmap
if (contenu is Bitmap)
{
```

```
var bmp:Bitmap = Bitmap(contenu);
bmp.smoothing = true;
}
//centrage du media
contenu.x= container_principal.width / 2 – contenu.width / 2;
contenu.y= container_principal.height / 2 – contenu.height / 2;
container_principal.addChild(contenu);
chargeur.contentLoaderInfo.removeEventListener(Event.INIT,chargementTermine
);
chargeur.contentLoaderInfo.removeEventListener(IOErrorEvent.IO_ERROR,erreur
);
chargeur.contentLoaderInfo.removeEventListener
(Event.OPEN,departChargement);
chargeur.contentLoaderInfo.removeEventListener
(ProgressEvent.PROGRESS,chargementEnCours);
}
```

2 Compréhension avancée des enjeux posés par le système évènementiel AS3.

Nous avons fait connaissance avec les évènements en programmant nos premiers click souris. Puis nous avons vu que d'autres types d'évènements pouvaient être émis par des processus non graphiques comme les chargements d'image, de fichiers texte.
Le système sur lequel repose l'architecture d'action script 3 est intimement lié avec les évènements. Pour quasiment chaque action que vous souhaiteriez détecter il existera un type d'évènement à écouter. Et si elle n'existe pas il est parfaitement possible de le créer par soi même à l'aide d'une classe d'évènement personnalisée.
Nous allons donc nous intéresser aux écouteurs un peu plus en détail.

Un évènement c'est quoi ?
C'est une classe spécialisée tout comme peuvent l'être une des classes natives de flash comme le MovieClip et SimpleButton. Comme ces classes, un évènement doit être instancié :

new Event(type_evenement) ;

Les DisplayObject de flash font cela automatiquement. Un click souris instancie automatiquement la classe d'évènement MouseEvent.

var clicksouris : MouseEvent = new MouseEvent (MouseEvent.Click) ;

Le diffuseur d'évènements

Continuons avec l'exemple du clic de souris sur notre clip. Nous avons vu que flash instanciait automatiquement cet évènement. Ce n'est pas tout d'instancier il faut encore l'émettre pour qu'il puisse être entendu. Il lui faut enfin diffuser cette instance au travers d'une autre classe spécialisée : EventDispatcher.

Grace à cette classe il est possible de s'abonner pour l'écoute de tous les types d'évènements avec la méthode addEventListener (type_a_ecouter,fonction_a_ecouter) proposée par cette classe.
Grace à cette classe, il est possible de diffuser tout les types d'évènements sans exception. Une méthode dispatchEvent(instance_evenement_a_diffuser).

Une instance donnée d'EventDispatcher est donc le point de jonction entre un évènement et l'éventuel écouteur qui peut s'être abonné à cette même instance.

Il se trouve que les objets interactifs de flash « ont » tous dans leurs classes respectives l'ensemble des fonctions qui forment la classe EventDispatcher. L'objet et le diffuseur sont donc imbriqués l'un dans l'autre.
C'est donc l'objet lui-même, le sprite, le MovieClip, (…), qui est en mesure d'émettre un évènement plutôt que par le biais d'une instance d'EventDispatcher. C'est ce qui nous permet également de nous abonner directement à l'objet à l'aide de sa fonction EventDispatcher. Et de le cibler avec la propriété target de l'événement émis

Les évènements et la liste d'affichage

Destruction des objets et écouteurs d'évènement.

Lorsque un objet de la liste d'affichage est retiré de celle-ci, il n'est pas pour autant détruit. Il faut pour cela passer sa référence à null :

objet = null ;

Les objets entreposés dans le cache peuvent à partir d'un certain poids et nombre encombrer l'application et ralentir son déroulement. Pour éviter ce cas de figure flash détruit du cache le objet éligibles.
Pour être éligible à la destruction (c.f. classe GarbageCollector) un objet doit remplir deux critères :

- Il ne doit pas se trouver dans la liste d'affichage : Le fait de passer la variable à null ne détruit pas véritablement l'objet, il ne fait que découpler la variable et l'objet auquel elle fait référence.
- Aucun écouteur ne doit faire référence à cet objet : Si un clip, par exemple, est écouté pour un évènement de type MouseEvent.CLICK, qu'il est retiré de la liste, et que sa valeur est passée à null. Il résidera malgré tout dans la mémoire, tout en étant plus accessible, l'empêchant d'être éliminé si l'application a besoin de faire un peu de place. Il est particulièrement recommandé de se préoccuper de cette question lors d'affichages multiples d'un même objet en grand nombre comme une galerie.

Les écouteurs d'évènement sont tous stockés dans un tableau général. Ce tableau n'est pas accessible depuis le code. On peut exploiter deux façons de traiter cette question :

1) On passe la variable useWeakReference à « true », dans notre déclaration d'écoute :

Objet.addEventListener(typeEcouteur,fonctionQuelqonque,useCapture,Priorité, useWeakReference) ;

Objet.addEventListener(MouseEvent.CLICK, fonctionQuelqonque, false, 0, true) ;

Ce paramètre à pour effet de placer l'écouteur en question dans un tableau secondaire qui annule l'obligation de désactiver l'écouteur pour rendre l'objet écouté éligible à la destruction. Il ne règle pour autant pas le problème dans tout les cas. Par exemple nous pouvons avoir besoin de retirer un objet de la liste d'affichage et d'avoir la certitude qu'il ne sera pas détruit parce que nous en aurons l'utilité plus tard.
Cela, de surcroit, n'est pas une solution propre puisque subsisteront dans la mémoire de l'application les écouteurs.

2) On intègre à nos classes graphiques une gestion des écouteurs à l'aide des évènements émis lors de l'ajout et du retrait de la liste d'affichage. C'est bien sur le moyen le plus propre pour gérer les écouteurs.
Le modèle de classe suivant peut être utilisé :

```
package
{
	import flash.display.Sprite;
	import flash.events.Event;
	import flash.events.MouseEvent;

	public class Container extends Sprite
	{
		Private var layer_decor :Sprite = new Sprite () ;
Private var layer_controles :Sprite = new Sprite () ;

		public function Container ()
		{

// la fonction ajoutScene se declenchera lorsqu'une instance de
// cet objet sera ajouté à la liste d'affichage
			this.addEventListener(Event.ADDED_TO_STAGE, ajoutScene);

			// Je dessine une forme :
			layer_decor.graphics.beginFill(0x000000, .5);
			layer_decor.graphics.drawRect(0, 0, 200, 200);

			//je crée un comportement de bouton pour l'instance de
cet objet
			this.buttonMode = true;
			this.useHandCursor= true;

			//je désactive la sensibilité souris des objets se
trouvant au
// dessus de la racine de mon clip.
			layer_decor.mouseEnabled = false;
			layer_controles.mouseEnabled = false;

			this.addchild(layer_decor);
			this.addchild(layer_controles);

			// j'installe mon ecouteur principal
			this.addEventListener(MouseEvent.CLICK, clique);


		}

		private function ajoutScene (even:Event):void
		{
			// Une fois sur le scene, je pose un ecouteur sur
l'évènement
```

```
                  // retrait de la scène.
                  this.addEventListener(Event.REMOVED_FROM_STAGE,
retraitScene);
                  // Si je veux automatiquement supprimer l'écouteur :
                  this.removeEventListener(Event. ADDED_TO_STAGE,
ajoutScene);

            }

            public function retraitScene (even:Event = null):void
            {
                  this.removeEventListener(Event.REMOVED_FROM_STAGE,
retraitScene);
            }

            public function clique (even:MouseEvent = null):void
            {
                  // l'objet se retire de la liste d'affichage
this.parent.removeChild(this);
            }
      }
}
```

On trouve également les évènements Event.ADDED et Event.REMOVED. Respectivement , ils sont émis lors de l'ajout ou du retrait d'un objet parent même si celui-ci n'appartient pas à la liste d'affichage.

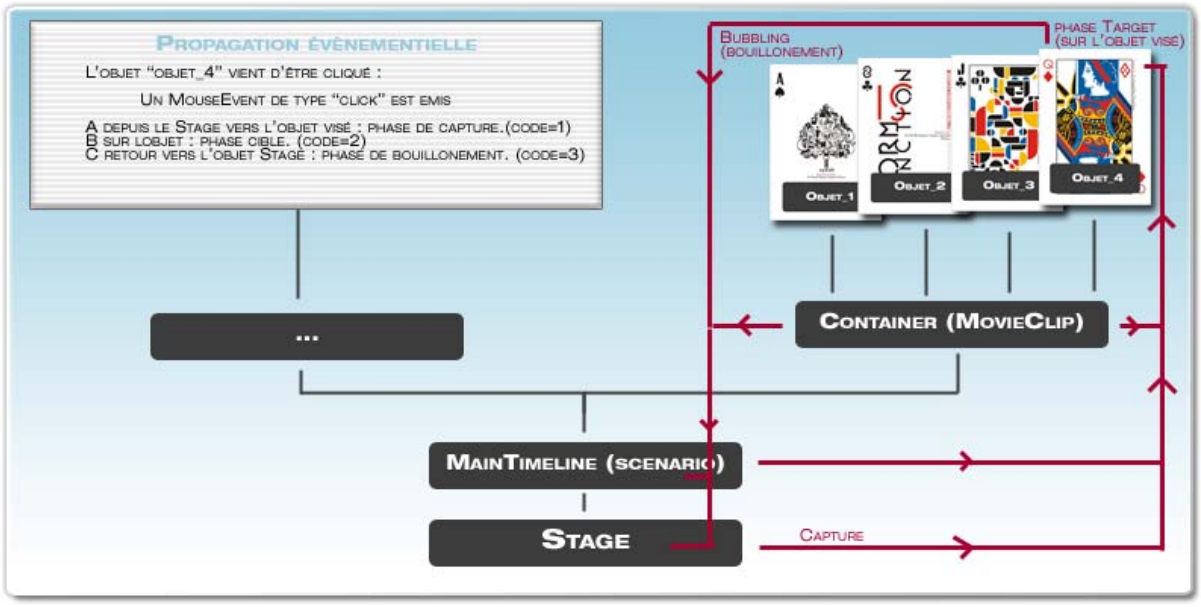
L'autre utilité de ce fonctionnement est de protéger l'application contre une tentative d'accès d'un objet d'affichage à des propriétés qui sont instruites seulement à partir du moment où l'objet se trouve dans la liste d'affichage.
Ces propriétés sont les suivantes :

Objet_graphique.stage : Permet d'accéder à la racine absolue de notre application.
Objet_graphique.root : Permet d'accéder à la racine relative de notre application.
Objet_graphique.parent : Permet d'accéder au parent de l'objet.

Si l'objet n'est pas dans la liste d'affichage ces propriétés renvoient « null ». Et l'application génère une erreur.

Parcours d'un évènement écran dans la liste d'affichage : L'utilisation avancée des évènements écrans (useCapture, bubbling).

Notre visite de l'actionscript nous pousse à nous intéresser de plus près à un aspect particulier du modèle évènementiel. Une nouvelle notion à ingérer mais qui va nous aider à simplifier notre code :
Nous l'avons vu précédemment. L'utilisation des écouteurs dans ActionScript 3 nécessite une gestion rigoureuse si l'application est un tant soit peu complexe. Il n'est donc pas question de placer des écouteurs dans tous les bouts de code sans savoir ce qu'il adviendra de ceux-ci.
Une des fonctionnalités du nouveau modèle évènementiel permet une économie de moyen, qui va s'avérer fort utile dans l'optimisation des nos applications.
Chaque émission d'évènement par une instance graphique de la liste d'affichage, parcourt l'ensemble de la liste d'affichage depuis l'objet Stage jusqu'à l'objet émetteur puis de cet objet à l'objet Stage. Il y a donc trois phase de propagation. Lorsque l'évènement va du Stage à l'objet ciblé, lorsqu'il a atteint cet objet, puis lorsque qu'il repart de l'objet à la racine de l'animation.



Il faut encore répéter que ce modèle n'est pas valable sur des objets non affichables où l'évènement ne se propage pas à travers l'animation.

La phase « at target »

Lorsque l'on écoute un évènement sans nous occuper des paramètres supplementaires de la méthode addEventListener nous écoutons en fait la phase at_target décrite par le code (2)

Nous gardons l'architecture décrite dans le shema ci-dessus et faisons appel à la propriété eventPhase de l'instance d'évènement transmis à la fonction :

```
Objet_4.addEventListener(MouseEvent.CLICK,ecouteClic);

function ecouteClic(even:MouseEvent):void
{
      trace("phase en cours : "+ (even.eventPhase); // Renvoie 2
      (at_target);
      trace("cible (target) : "+ (even.target.name); // Renvoie le nom
      d'occurrence du clip cible : « objet_4 »;
      trace("cible (currentTarget) : "+ (even.currentTarget.name); //
      Renvoie le nom d'occurrence du clip cible : « objet_4 »;

}
```

Cette méthode d'écoute est celle que nous connaissions dans actionscript 2. C'est la méthode la plus basique. Mais loin d'être la plus souple.

La phase de capture

Cette méthode est particulièrement intéressante pour l'écoute d'objets multiples de même type. Nous savons donc que pendant la phase de capture, l'évènement parcourt chaque objet de la hiérarchie de l'objet visé par l'évènement.

Il traversera donc chacun des parents. L'utilisation du paramètre useCapture de la méthode objet_parent.addEventListener (type,ecouteur,useCapture) va nous permettre d'intercepter l'évènement pendant ce parcours.
Nous choisissons donc de placer un évènement sur l'objet « container » de notre schéma

```
container.addEventListener(MouseEvent.CLICK,ecouteClic,true);

function ecouteClic(even:MouseEvent):void
{
trace("phase en cours : "+ (even.eventPhase); // Renvoie 1 (capture);
      trace("cible (target) : "+ (even.target.name); // Renvoie le nom
d'occurrence du clip cible : « objet_4 »;
      trace("cible (currentTarget) : "+ (even.currentTarget.name); //
Renvoie le nom d'occurrence du clip sur lequel est placé l'écouteur : «
container »;

}
```

Nous voyons dans l'exemple que even.currentTarget définit l'objet ecouté par la méthode. Et que l'objet que nous avons cliqué lui est designé par la propriété even.target.
Nous n'avons nulle part indiqué que c'est l'objet « objet_4 » qui doit être écouté, et pour cause ! C'est en fait toute la hierarchie descendante de l'objet écouté qui déclenchera la fonction. Dans notre exemple, indistinctement, les 4 objets enfants déclencheront la fonction dès lors où ils seront cliqués.
Nous voyons dès lors l'intérêt de placer un seul écouteur en amont, par exemple, sur un container de vignettes dans une galerie photo ! Au contraire, l'utilisation peut être très fastidieuse si la hiérarchie descendante contient beaucoup d'autres types d'objets et si l'on a pas pris soin de désactiver la sensibilité souris de ces derniers à l'aide de la méthode mouseEnabled des DisplayObject

La phase de bouillonnement

La dernière phase dite Bubbling ou bouillonnement ressemble fort à la précédente dans ses caractéristiques : l'écoute concerne alors la hiérachie descendante de l'objet sur lequel est placé l'écouteur.
La fonction est alors déclenchée dans un troisième temps. Il suffit pour mettre l'écoute en place de viser l'objet parent comme dans la phase de capture, mais cette fois ci de laisser le paramètre useCapture sur sa position par défaut. Ici en l'occurrence nous n'y faisons pas référence dans la méthode.

```
container.addEventListener(MouseEvent.CLICK,ecouteClic);

function ecouteClic(even:MouseEvent):void
{
trace("phase en cours : "+ (even.eventPhase); // Renvoie 3 (bubbling);
      trace("cible (target) : "+ (even.target.name); // Renvoie le nom d'occurrence d'un des clip
de la hierarchie déscendante : « objet_4 » par exemple;
      trace("cible (currentTarget) : "+ (even.currentTarget.name); // Renvoie le nom
d'occurrence du clip sur lequel est placé l'écouteur : « container »;

}
```

Les occasions d'utiliser ce dernier comportement restent toutefois assez rares, au contraire de la phase de capture très utilisée.

Utilisation des fonctions et propriétés proposées par les évènements
Toutes les occurrences d'évènements possèdent au moins ces propriétés en tant qu'elles héritent toutes de la classe mère Event :

target : définit l'objet émeteur.
Dans le cas de l'écoute d'un objet graphique nous pourrons accéder à chacune de ses propriétés et méthodes au travers de cette propriété.

currentTarget : définit l'objet écouté.

eventPhase : Indique si l'événement est en phase de capture (1) de at_target (2) ou de remontée(3).

type : envoie le type de l'événement utilisé.

Afin d'assouplir encore le code de nos applications, au lieu de retirer un écouteur d'évènement d'un objet défini, d'un évènement d'un type défini, nous pouvons utiliser à la place la syntaxe suivante en se servant des propriétés accessibles dans la fonction :

```
container.addEventListener(MouseEvent.CLICK,ecouteClic);

function ecouteClic(even:MouseEvent):void
{

even.target.removeEventListener(even.type,ecouteClic);
//revient exactement au meme que :
      // container.removeEventListener(MouseEvent.CLICK,ecouteClic);
}
```

L'intérêt de ce comportement réside dans le fait que l'objet écouté n'a plus à être défini dans la classe où s'effectue la fonction. Il est possible donc de déposer cette ligne partout ou un ecouteur à besoin d'être désactiver en modifiant uniquement la référence de fonction.

Stopper la progression événementielle.

D'autres méthodes utiles se trouvent dans l'instance d'un évènement. Nous citerons en particulier les méthodes d'arrêt d'un évènement stopPropagation(), qui empêche l'évènement de se transmettre au delà de l'enfant direct d'un objet écouté en phase de capture.
La méthode stopImmediatePropagation() quand à elle, stoppe la propagation événementielle à l'objet écouté lui-même. Utile en cas de système de gestion d'un menu.

Installer un diffuseur d'évènement sur une classe non graphique.

Deux possibilités permettent à une instance de classe d'émettre des évènements sans être pour autant une classe héritant de DisplayObject. Nous n'en développerons ici qu'une.
La première possibilité sur laquelle nous ne nous étendrons pas est d'utiliser un mécanisme d'implémentation de la classe Event sur notre classe.
Sommairement, il s'agirait de déclarer que nous sommes fidèles au modèle de la classe EventDispatcher, en en reprenant toute les fonctions (interface) et de placer dans notre classe l'ensemble des fonctions qui composent la classe de diffusion. L'avantage de cette méthode est que l'instance de la classe et le diffuseur ne font qu'un, simplifiant ainsi l'utilisation de target.

Il nous sera facile alors de récupérer des informations dans l'objet sur lequel est placé l'écouteur.

L'autre solution nettement plus « simple » à l'usage est de créer dans notre classe une instance d'EventDispatcher La méthode implique que la propriété target de l'instance d'évènement ne renvoie que vers le diffuseur lui même et ne permet donc pas d'accéder aux propriétés et méthodes de la classe qui le supporte.
Si c'est possible de directement faire hériter la classe de EventDispatcher. De cette façon le diffuseur et l'instance de classe sont les mêmes, et la propriété target nous permet directement d'accéder au données de notre objet.

Exemple : une classe de chargement.

Nous l'avons vu précédemment un chargement d'image ou de swf est une opération coûteuse en lignes de code. Pourquoi ne pas placer ces lignes dans une classe que nous pourrions utiliser à loisir en une vingtaine de lignes comprenant la barre de chargement :

```
package utilitaires
{
	import flash.display.Bitmap;
	import flash.display.Loader;
	import flash.events.Event;
	import flash.events.EventDispatcher;
	import flash.events.ProgressEvent;
	import flash.events.IOErrorEvent;
	import flash.net.URLLoader;
	import flash.net.URLRequest;
	import flash.display.LoaderInfo;
	import flash.display.DisplayObject;


	public class ChargeurGenerique extends EventDispatcher
	{
		// le loader est accéssible à toutes les fonctions
		private var chargeur:Loader;
		// La variable receuille le produit du chargement
		private var contenuCharge:DisplayObject;
		// raccourci vers le diffuseur de l'objet loader
		public var infoLoad:LoaderInfo;
		// image remplaçante en cas d'erreur de chargement
		private static var image_defaut:DisplayObject;
		// pourcentage du chargement.
		public var ratio_chargement:Number = 0;

		public function ChargeurGenerique()
		{
			// instanciation du chargeur
			chargeur = new Loader();
			// mise en place des écouteurs
			infoLoad = chargeur.contentLoaderInfo;

			infoLoad.addEventListener ( Event.INIT, chargementTermine
);
			infoLoad.addEventListener ( Event.OPEN, redirigeEvenement
);
	infoLoad.addEventListener ( ProgressEvent.PROGRESS, calculRatio );
```

```
		infoLoad.addEventListener ( IOErrorEvent.IO_ERROR, echecChargement );
		addEventListener ( Event.REMOVED_FROM_STAGE, annulationChargement );
			}

// on donne la possibilité de signifier une image par défaut via une
fonction statique :
// - l'image sera disponible pour toutes les instances de
ChargeurGenerique.
// - Elle sera appelée de cette manière ChargeurGenerique.imageDefaut =
un_Display_Object;
public static function set imageDefaut(image:DisplayObject):void
{image_defaut = image; }
public static function get imageDefaut():DisplayObject { return
image_defaut; }
public function get contenu():DisplayObject { return contenuCharge; }

			public function annulationChargement():void
			{
				chargeur.unload();
				chargeur.close();
				desactivation();
			}

// cette fonction regroupe tout les écouteur à annuler. De cette manière je
n'ai pas à dupliquer ces lignes de code.

private function desactivation ():void
{
		infoLoad.removeEventListener ( Event.INIT, chargementTermine);
		infoLoad.removeEventListener ( Event.OPEN, redirigeEvenement );
		infoLoad.removeEventListener ( ProgressEvent.PROGRESS, calculRatio );
infoLoad.removeEventListener  ( IOErrorEvent.IO_ERROR, annulationChargement
);
		removeEventListener ( Event.REMOVED_FROM_STAGE, desactivation );

			}

// les évènements de progression sont redirigés et viennent alimenter une
variable publique indiquant le ratio de chargement.
			private function calculRatio ( even:ProgressEvent ):void
			{
				ratio_chargement =   even.bytesLoaded/even.bytesTotal;
				this.dispatchEvent(even);
			}

			// les évènements diffusés par LoaderInfo sont redirigés sur
cet objet pour permettre l'écoute.
			private function redirigeEvenement ( even:Event ):void
			{
				this.dispatchEvent(even);
			}

			public function charger ( pURL:String ):void
			{
				//création de la requète
				var requete:URLRequest = new URLRequest()
				requete.url = pURL;
				// debut du chargement
				chargeur.load ( requete );
			}
```

```
		// Le chargement à bien executé
		private function chargementTermine ( even:Event ):void
		{
			// retrait des écouteurs
			desactivation();
			// mise à disposition du contenu par l'objet
			contenuCharge = even.target.content;
			// Si l'objet chargé est un bitmap, on le lisse
			if(contenuCharge is Bitmap)
			{
				// transtypage pour acceder à la fonction
				var bmp:Bitmap = Bitmap(contenuCharge);
				// fonction de lissage.
				bmp.smoothing = true;
			}
			// Envoi de l'évènement complete
			this.dispatchEvent(new Event(Event.COMPLETE));
		}

		private function echecChargement ( even:Event ):void
		{
			trace ("erreur de chargement");
			// si l'image par défaut existe
			if (image_defaut != null)
			{
				desactivation();
				contenuCharge = image_defaut;
				this.dispatchEvent(new Event(Event.COMPLETE));
			}
			else
			dispatchEvent(even.clone())
		}


	}

}
```

Depuis la classe appelante :

```
import utilitaires.*
…
ChargeurGenerique.imageDefaut = new Erreur();
var chargeur:ChargeurGenerique = new ChargeurGenerique();
chargeur.addEventListener(Event.COMPLETE,finChargement);
chargeur.addEventListener(ProgressEvent.PROGRESS,progression);

chargeur.charger("medias/fruits.jpg");

private function progression(even:ProgressEvent):void
{
	trace(chargeur.ratio_chargement)
	mc_barre.scaleX = chargeur.ratio_chargement;
}

private function finChargement(even:Event):void
```

```
{
	removeChild(mc_barre);
	chargeur.removeEventListener(Event.COMPLETE,finChargement)
	var ratio:Number =
Math.min(stage.stageWidth/chargeur.contenu.width,stage.stageHeight/chargeur
.contenu.height);
	chargeur.contenu.scaleX = chargeur.contenu.scaleY = ratio;
	addChild(chargeur.contenu);
}
```

Créer ses propres évènements et transmettre des informations personnalisées.

Parfois nous pouvons être amenés à avoir besoin de transmettre un message spécifique, cela peut aussi faire partie d'un certain confort. La gamme des évènements dans flash est très importante, mais un évènement spécifique destiné à acheminer un message, ou un objet quel qu'il soit pourrait simplifier les choses dans nombre des cas.
Par exemple, classiquement, lors de la création d'un diaporama, nous aurions besoin de précharger les trois premières images avant de lancer le diaporama, de façon à optimiser le fonctionnement. Un évènement personnalisé nous faciliterait la tache et pourrait transmettre les information de l'application sous un format unique ce qui nous éviterai la multiplication des fonctions.
Je vous propose donc une classe d'évènement très générique : EvenementPerso. Elle permet de faire passer n'importe qu'elle chaine de caractère, et un objet générique secondaire. En somme cette classe très pratique peut acheminer n'importe quel message et n'importe quel objet actionscript du plus simple au plus complexe à travers n'importe que EventDispatcher.

```
package utilitaires
{
	import flash.events.Event;

	public class EvenementPerso extends Event
	{
		public static const TRANSIT_MESSAGE:String = "use_message";


		private var _message:String = "";
		private var _objet:Object;

		public function EvenementPerso(type:String,
bubbles:Boolean=false, cancelable:Boolean=false,pmessage:String="")
		{
			_message = pmessage;
			super(type, bubbles, cancelable);
		}

		public function get objetSecondaire():Object { return _objet; }
		public function set objetSecondaire(obj:Object ):void{ _objet =
obj; }

		public function get message():String {	return _message; }

		public override function clone():Event
		{
			var even:SimpleEvent = new SimpleEvent(type, bubbles,
cancelable, _message);
			even.objetSecondaire = _objet;
			return  even;
```

```
            }

            public override function toString():String
            {
                  return formatToString("SimpleEvent", "type", "bubbles",
"cancelable", "eventPhase");
            }

      }

}
```

Utilisation comme évoqué plus haut mon diaporama à chargé ses trois images. Je souhaite le faire savoir à la scène principale pour qu'il arréte l'animation de prechargement. J'ai aussi un événement à envoyer lorsque le diaporama est fini. Je peux également faire savoir quelle photo est en cours de diffusion pour afficher le titre par exemple. Trois types d'évènement distincts mais qui pourraient tous transiter par la même classe d'évènement. Voire par le même type d'évènement.

Sur celle-ci j'ai placé un écouteur sur le diaporama :

```
diaporama.addEventListener(EvenementPerso.TRANSIT_MESSAGE,evtDiapo);


function evtDiapo(even:EvenementPerso):void
{
      if(even.message==”tampon_charge”)
      {
            // retrait de l’animation de préchargement et affichage du
titre ;
            titre = String(even.objetSecondaire) ;
      }
      else if ( even.message == “ fin_diapo“)
      {
            diaporama.pause();
            gotoAndPlay(“fin”)
      }

}
```

Depuis le diaporama maintenant, depuis les fonctions appropriées je pourrais envoyer l'évènement :

```
var evenement :EvenementPerso = new
EvenementPerso(EvenementPerso.TRANSIT_MESSAGE,false,false,”tampon_charge”);
evenement.objetSecondaire = paramXML.image[index_en_cours].titre;
this.dispatchEvent(evenement);
```

Plus loin le même évènement sera produit pour la fin du diaporama ou à chaque nouvel affichage de photo.
La fonction evtDiapo sera déclenchée par l'émission de l'évènement. La propriété message nous servira a discriminer l'étape et répondre par le comportement approprié.

En conclusion nous avons pu observer quelques uns des principaux aspects des évènements dans flash. Une fois aguerris à ce système évènementiel vous n'aurez plus aucune difficultés

pour synchroniser vos applications et proposer de vraies scénarisations de vos applications. Notamment vous serez aussi à l'aise si vous vous intéressez à javascript et jquery car le système évènementiel en vigueur est le même. Vous serez même surpris d'y trouver des fonctions telles que addEventListener !