ActionScript 3

Programmation séquentielle et orientée objet

Avec 180 scripts en téléchargement gratuit !

David Tardiveau

EYROLLES

ActionScript 3

Programmation séquentielle et orientée objet

CHEZ LE MÊME ÉDITEUR

Du même auteur

D. TARDIVEAU. – **150 scripts pour Flash CS3.**
N°12112, 2007, 520 pages.

D. TARDIVEAU. – **120 scripts pour Flash 8** (format semi-poche)**.**
N°12219, 2007, 462 pages.

D. TARDIVEAU. – **La vidéo dans Flash**.
N°12048, 2007, 190 pages.

Autres ouvrages sur Flash et le développement Web

A. TASSO. – **Apprendre à programmer en ActionScript 3.**
N°11199, 2007, 438 pages.

W. SIRACUSA. – **Faites vos jeux avec Flash !** – *Du Pacman au Sudoku.*
N°11993, 2006, 220 pages.

M. LAVANT. – **Flash 8 Professional.**
N°11950, 2006, 678 pages.

C. BERGÉ. – **Je crée mon site Internet avec Dreamweaver 8 et Flash 8.**
N°11977, 2006, 144 pages + CD-Rom vidéo.

J.-M. DEFRANCE. – **PHP/MySQL avec Flash 8.**
N°11971, 2006, 782 pages.

K. GOTO, E. COTLER. – **Redesign web 2.0.**
N°11579, 2005, 294 pages.

J. ZELDMAN. – **Design web : utiliser les standards.** *CSS et XHTML.*
N°12026, 2006, 382 pages.

A. CLARKE. – **Transcender CSS.**
N°12107, 2007, 370 pages.

A. BOUCHER. – **Ergonomie web.** *Pour des sites web efficaces.*
N°12158, 2007, 426 pages.

E. SLOÏM. – **Mémento Sites web.** *Les bonnes pratiques.*
N°12101, 2007, 14 pages.

ActionScript 3

Programmation séquentielle et orientée objet

David Tardiveau

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

© Groupe Eyrolles, 2008, ISBN : 978-2-212-12282-4

Mise en page : TyPAO
Dépôt légal : mars 2008
N° d’éditeur : 7795
Imprimé en France

Avant-propos

Pourquoi un tel ouvrage ?

ATTENTION : note aux néophytes en programmation

À la fin de la lecture de cet avant-propos, si les quelques pages qu'il contient vous semblent trop abstraites lisez le chapitre 1 et passez directement au troisième.

Le manuel officiel de l'ActionScript 3

Si vous avez déjà ouvert la documentation officielle du langage ActionScript 3, vous aurez sûrement remarqué à quel point il est difficile d'en comprendre les explications qui, bien que très précises, demeurent très abstraites pour les néophytes en programmation.

En tant qu'enseignant, j'aurais tendance à dire qu'à vouloir être trop précis, on perd souvent en clarté !

La société Adobe n'a malheureusement pas le choix car ce manuel dédié au langage ActionScript s'adresse à tout le monde, y compris les développeurs les plus expérimentés, et ces derniers ont également besoin d'informations qui leur parlent. Le guide de référence proposé par la société éditrice se doit donc d'être aussi exhaustif que possible.

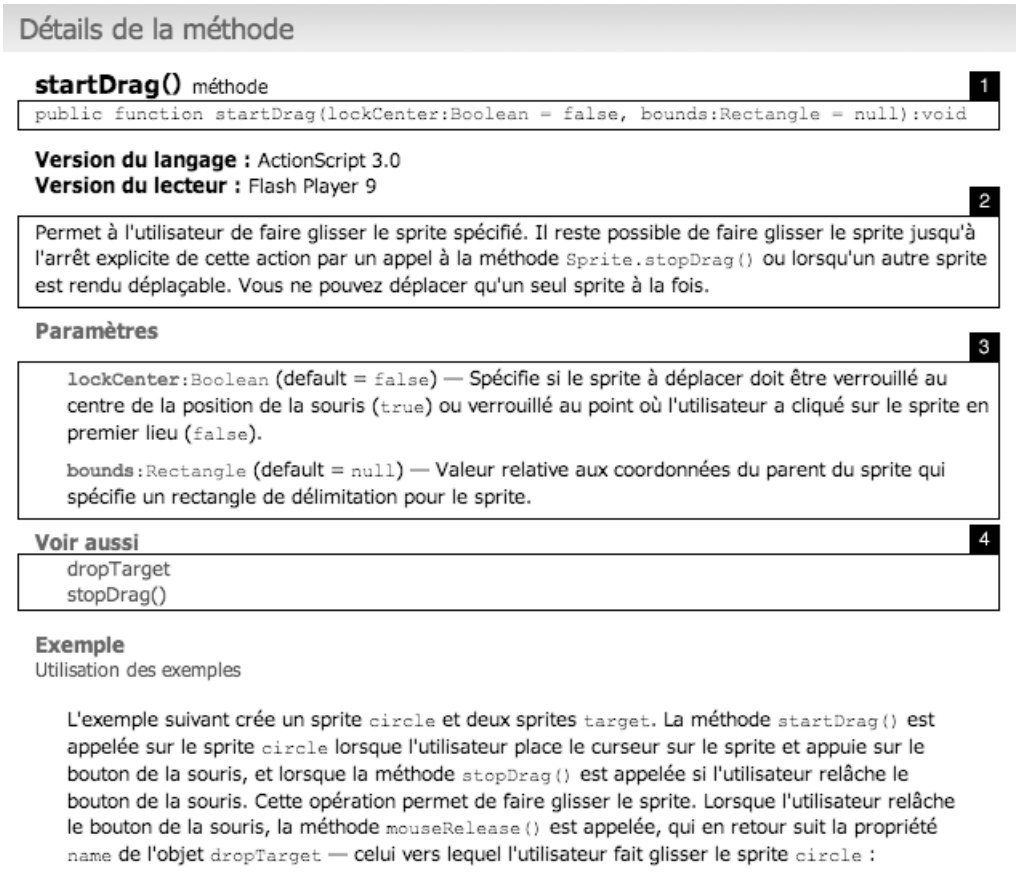
Mais ce manuel est-il si abstrait pour les débutants en programmation ? Prenons la définition de la méthode `startDrag()` de la figure A-1. Nous allons découvrir que, selon le niveau de chacun, on peut plus ou moins comprendre certains points. Analysons ensemble les quatre parties mises en avant au travers des cadres numérotés de 1 à 4.

Point numéro 1 : seuls les développeurs ayant déjà un bon niveau en programmation peuvent interpréter le sens de cette ligne d'instruction. Elle permet néanmoins de comprendre très rapidement la syntaxe de la méthode accompagnée des paramètres obligatoires et facultatifs.

Point numéro 2 : il s'agit certainement de l'explication la plus abordable par l'ensemble des développeurs, néophytes et confirmés, car elle définit le fonctionnement et/ou l'utilité du terme.

Point numéro 3 : cette partie de l'explication est primordiale car elle précise les paramètres à utiliser lorsque le terme est une méthode. Malheureusement, elle n'est compréhensible qu'à partir du moment où un développeur possède déjà les connaissances requises pour programmer.

Point numéro 4 : par analogie ou par opposition, il est parfois intéressant de comprendre des termes ayant une relation avec celui que vous cherchez à maîtriser initialement.



Détails de la méthode

startDrag() méthode 1

```
public function startDrag(lockCenter:Boolean = false, bounds:Rectangle = null):void
```

Version du langage : ActionScript 3.0
Version du lecteur : Flash Player 9

2

Permet à l'utilisateur de faire glisser le sprite spécifié. Il reste possible de faire glisser le sprite jusqu'à l'arrêt explicite de cette action par un appel à la méthode `Sprite.stopDrag()` ou lorsqu'un autre sprite est rendu déplaçable. Vous ne pouvez déplacer qu'un seul sprite à la fois.

Paramètres 3

`lockCenter:Boolean` (default = `false`) — Spécifie si le sprite à déplacer doit être verrouillé au centre de la position de la souris (`true`) ou verrouillé au point où l'utilisateur a cliqué sur le sprite en premier lieu (`false`).

`bounds:Rectangle` (default = `null`) — Valeur relative aux coordonnées du parent du sprite qui spécifie un rectangle de délimitation pour le sprite.

Voir aussi 4

dropTarget
stopDrag()

Exemple
Utilisation des exemples

L'exemple suivant crée un sprite `circle` et deux sprites `target`. La méthode `startDrag()` est appelée sur le sprite `circle` lorsque l'utilisateur place le curseur sur le sprite et appuie sur le bouton de la souris, et lorsque la méthode `stopDrag()` est appelée si l'utilisateur relâche le bouton de la souris. Cette opération permet de faire glisser le sprite. Lorsque l'utilisateur relâche le bouton de la souris, la méthode `mouseRelease()` est appelée, qui en retour suit la propriété `name` de l'objet `dropTarget` — celui vers lequel l'utilisateur fait glisser le sprite `circle` :

Figure

Les explications de l'aide officielle de l'ActionScript 3 ne sont pas toujours compréhensibles pour tout public.

```
import flash.display.Sprite;
import flash.events.MouseEvent;

var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFFCC00);
circle.graphics.drawCircle(0, 0, 40);
```

```
var target1:Sprite = new Sprite();
target1.graphics.beginFill(0xCCFF00);
target1.graphics.drawRect(0, 0, 100, 100);
target1.name = "target1";

var target2:Sprite = new Sprite();
target2.graphics.beginFill(0xCCFF00);
target2.graphics.drawRect(0, 200, 100, 100);
target2.name = "target2";

addChild(target1);
addChild(target2);
addChild(circle);

circle.addEventListener(MouseEvent.MOUSE_DOWN, mouseDown)

function mouseDown(event:MouseEvent):void {
  circle.startDrag();
}
circle.addEventListener(MouseEvent.MOUSE_UP, mouseReleased);

function mouseReleased(event:MouseEvent):void {
  circle.stopDrag();
  trace(circle.dropTarget.name);
}
```

Comme vous pouvez le constater, si quelqu'un vous dit : pour réaliser un puzzle, tu dois utiliser la méthode `startDrag()`, vous pourrez difficilement vous en sortir avec les explications ci-dessus, surtout si vous n'avez jamais programmé.

La présentation de ce script dans un contexte précis le rend, paradoxalement, encore moins évident à comprendre. L'utilisateur du manuel doit d'abord analyser le script avant de découvrir les parties indispensables pour réaliser un glisser-déposer.

Rappelons que l'aide officielle du langage vous sera un jour très utile car il s'agit d'un manuel extrêmement bien conçu, mais vous devez d'abord comprendre les concepts de la programmation et maîtriser ses mécanismes. Cette maîtrise passera certainement par une longue phase de pratique du langage. En attendant, si ce référentiel ne peut vous aider, il vous reste Internet et les livres pour apprendre l'AS3 (ActionScript 3).

Sur Internet, les forums constitueront pour vous une aide précieuse lorsque vous rencontrerez des problèmes. Le plus réputé dans la communauté francophone de Flash et de l'ActionScript est celui de mediabox : `http://flash.mediabox.fr/index.php`

Programmation séquentielle ou bien orientée objet ?

À ce jour, il est difficile d'évaluer si la programmation séquentielle (ou structurée) est plus utilisée en AS3 que la programmation orientée objet ; mais il est pour l'instant rare de trouver des explications qui n'aient pas une approche objet. Vu la difficile maîtrise des langages orientés objet et le succès de Flash, il est fort probable que la programmation séquentielle aura un bel avenir en AS3.

Concernant les publications papier, si vous avez acheté un livre pour apprendre l'AS3, c'est que vous savez déjà quels sont les avantages et les inconvénients d'une telle méthode d'acquisition. Allons tout de même plus loin et présentons l'approche de cet ouvrage.

Différences entre l'AS1, l'AS2 et l'AS3

L'ActionScript 3 constitue un tournant majeur dans l'histoire de Flash et plus particulièrement du langage AS. Sans entrer dans les détails, nous pouvons tout de même dire que cette version risque, une fois encore, de creuser l'écart entre les développeurs confirmés et les néophytes en matière de programmation. Pour commencer, l'un des objectifs de ce livre est de réduire cet écart.

L'approche de la programmation étant différente entre l'AS1 et l'AS2, un écart s'était déjà creusé entre ces deux populations. Aujourd'hui, tous les utilisateurs de l'AS2 vont tout naturellement passer à l'AS3. Pour ceux qui programment en AS1, ce livre tend à démontrer que l'AS3 leur est tout de même accessible. Nous reviendrons sur ce point.

Avant d'expliciter davantage l'approche pédagogique de ce livre face au besoin d'apprentissage de l'AS3, revenons sur les différences qui séparent les trois versions.

La vraie différence entre l'AS1 et l'AS2

Pour certains anciens utilisateurs de Flash, la différence entre l'AS1 et l'AS2 se caractérise, à tort, par le fait de placer les scripts d'une animation sur les occurrences (AS1) ou sur l'image-clé (AS2). Les explications ci-dessous vont vous démontrer le contraire.

Ce qui caractérise l'ActionScript 1, c'est le fait de placer les scripts d'une animation sur l'image-clé d'un scénario (le scénario général ou celui d'un clip) ou sur une occurrence. La création de l'AS1 n'est pas vraiment datée, mais nous pouvons dire qu'avant 2000 (la sortie de Flash 5), le système de gestion des scripts ne représentait pas vraiment un langage à part entière. De 2000 à 2003, l'AS1 aura connu deux syntaxes pour gérer ses événements : les scripts pouvaient être placés sur les occurrences ou sur l'image-clé pour effectuer une même action.

Les utilisateurs de Flash peu expérimentés en programmation préféraient placer les scripts sur les occurrences (de bouton et de clips), cette logique étant plus compréhensible pour eux : on place le script sur le bouton sur lequel l'utilisateur cliquera pour déclencher une action. Par ailleurs, la syntaxe était plus simple. Les utilisateurs plus expérimentés en programmation préféraient placer tout le code d'une animation au même endroit, sur une image-clé du scénario principal, leur argument étant le suivant : le débogage est plus simple car le code est centralisé. Au sein de la communauté des utilisateurs de Flash, puis ensuite des développeurs Flash, cette double approche du langage a constitué deux groupes d'utilisateurs.

Utilisateurs de Flash et développeurs Flash

C'est avec le temps qu'est née la différence entre les utilisateurs de Flash et les développeurs Flash. De 2000 à 2005, un utilisateur de Flash était qualifié de flasheur, même s'il utilisait considérablement ActionScript. Progressivement, à partir de 2005, le marché de l'emploi a fait la différence entre les flasheurs qui sont principalement représentés par les utilisateur de Flash au travers de son interface (utilisation importante du scénario) et les développeurs Flash qui font appel principalement au code pour contrôler une animation.

À la sortie de l'AS2, en 2003 avec l'arrivée de Flash MX2004, les développeurs qui avaient l'habitude de placer tout leur code sur une image-clé ont tout naturellement opté pour le nouveau mode de programmation : l'utilisation et l'appel d'un fichier externe pour rédiger les scripts d'une animation.

En résumé, l'AS1 se caractérise par le fait de placer ses scripts dans l'animation alors que l'AS2 fait appel à des fichiers externes. En programmation il existe deux approches : la programmation orientée objet et la programmation séquentielle appelée aussi programmation structurée. La première approche est donc tout naturellement associée à l'AS2 et la deuxième à l'AS1.

Création d'un nouveau document dans Flash CS3

Aujourd'hui, lorsque vous demandez la création d'un nouveau document dans Flash CS3, vous pouvez opter pour un fichier Flash AS2 ou un fichier Flash AS3. Il est important de comprendre qu'Adobe a décidé de regrouper les versions AS1 et AS2 sous une même étiquette car elles utilisent des syntaxes différentes mais un même langage. Les termes sont identiques, seules quelques spécificités les différencient. L'ActionScript 3 n'utilise plus du tout la même syntaxe, mais surtout, le langage est complètement différent (par exemple, la propriété `._x` en AS2 s'écrit `.x` en AS3). Lorsque la société Adobe fait référence à l'AS2, elle veut marquer la différence entre les deux registres de vocabulaires de l'AS1/AS2 et de l'AS3 (les approches de programmation orientée objet et séquentielle étant sous-entendues pour l'AS2).

Mais pourquoi tous les utilisateurs ne sont-ils pas passés à l'AS2 ? C'est effectivement la question que nous devons nous poser, car la réponse que nous allons formuler d'ici quelques lignes va, par la même occasion, nous faire comprendre la différence qui existe entre l'AS3 et l'AS2 !

Il est important de comprendre que la programmation orientée objet, qui est donc associée à l'AS2 et l'AS3 nécessite un bon niveau en programmation. Parallèlement, Flash est un logiciel qui connaît depuis 3 à 4 ans un énorme succès auprès de publics très divers, y compris des utilisateurs ne connaissant pas du tout la programmation et n'ayant pas un grand besoin en terme de développement informatique. Pourquoi un imprimeur aurait besoin d'apprendre un langage complexe pour réaliser une animation avec une légère interactivité ? Bien évidemment, nous avons pris ce corps de métier pour notre exemple, mais il en existe bien d'autres. Par ailleurs, pourquoi des personnes talentueuses dans leur domaine devraient-elles apprendre un langage de programmation pour réaliser une animation interactive ?

Flash, un logiciel à tout faire

D'une façon générale, tous les métiers de la communication ont trouvé dans ce logiciel un moyen facile et rapide de diffuser une information sur Internet. Flash proposant aussi de publier des animations pour les supports off-line, un marché supplémentaire s'est même créé, remplaçant par la même occasion la plupart des productions réalisées par le logiciel Director.

Répondons simplement à ces deux questions en affirmant que l'AS2, avec une approche de programmation séquentielle (ou structurée), va malheureusement perdurer encore quelques années, car les besoins en programmation ne sont pas les mêmes pour tout le monde. Pourquoi utiliser le terme malheureusement dans la phrase précédente ? Est-ce un jugement subjectif ? Non, cela signifie simplement que la différence que nous avions entre les deux communautés d'utilisateurs de l'ActionScript va non seulement perdurer mais aussi se renforcer entre l'AS2 (à l'approche de programmation séquentielle) et l'AS3.

L'ActionScript 3 est un langage complètement orienté objet car la rédaction des lignes d'instructions qui composent un script ou un programme est très précise. La structure d'un document a complètement été repensée ; le développeur manipule à présent des objets imbriqués et non plus des occurrences de type clip ou bouton, comme c'était le cas dans les version antérieures à Flash CS3. Cette version du langage n'est-elle donc pas plus accessible aux développeurs novices ? Pour ces derniers la réponse est simple : vous pouvez développer en AS3 avec une approche séquentielle !

L'un des objectifs de ce livre est non seulement de présenter en parallèle les deux syntaxes (programmation orientée objet et programmation séquentielle), mais aussi de démontrer qu'il est possible d'apprendre l'AS3 en étant un débutant total en programmation.

Pourquoi proposer à des développeurs de programmer en AS3 avec une syntaxe séquentielle ? Ces dernières années, l'enseignant que je suis est quelque peu excédé face aux comportements de certains collègues ou développeurs confirmés. Non, la programmation orientée objet n'est pas accessible à tout le monde et ce n'est pas non plus une question de pédagogie. Il a été assez agaçant de constater, avec la montée en puissance de l'AS2, à quel point certains individus peuvent être si obtus et penser que tout le monde possède les mêmes facilités. J'aurais envie de dire à ces gens-là : « Comment, vous ne savez pas préparer une charlotte au chocolat ? Il suffit pourtant de mélanger des ingrédients en suivant une recette… » Ah bon, chef cuisinier c'est un métier ?

Approche pédagogique de ce livre

La plupart des ouvrages dédiés à l'apprentissage d'un langage informatique proposent une table des matières basée sur les notions élémentaires pour maîtriser l'algorithme. C'est une excellente logique : il est ainsi plus facile de passer d'un langage à un autre. Une telle approche pédagogique permet également une montée en puissance de la difficulté en phase d'apprentissage. J'ai pu constater au long de ma carrière qu'il existe de nombreuses approches pédagogiques pour transmettre un savoir. Elles s'adressent toutes à des publics aux profils différents.

Un étudiant qui se forme aux métiers du développement aura besoin de connaître les bases de la programmation. Il devra savoir construire un algorithme, mais aussi comprendre et maîtriser les concepts de la programmation orientée objet. Lorsque tout cela est acquis, il peut alors passer d'un langage à une autre assez facilement ; il doit simplement se familiariser avec les spécificités de chacun.

Si vous-même connaissez déjà les bases de la programmation, ce livre va vous sembler évident et simple. En revanche, si vos connaissances en programmation sont réduites, je tiens à vous rassurer : la progression du livre s'appuie sur les besoins rencontrés en production. Il sera parfois nécessaire de faire quelques allers-retours d'un chapitre à un autre mais, dans l'ensemble, l'ordre des chapitres a été choisi de sorte que vous puissiez programmer une animation Flash, même si vous ne maîtrisez pas le contenu de la partie 2 de ce livre.

Depuis plus de 13 ans, mon métier est de transmettre mes connaissances, initialement dans le domaine de la PAO, puis dans les métiers du multimédia depuis l'an 2000. En plus de 7 ans, quelque 600 à 700 personnes sont passées dans les différentes formations Flash que j'ai dû assurer pour le compte de divers établissements privés et publics. Ces apprenants ont tous buté sur les mêmes difficultés et sont tombés dans les mêmes pièges !

Aujourd'hui, pour apprendre un langage, il est très simple d'accéder à une formation, mais cela reste coûteux. De nombreuses publications papier restent alors abordables : le prix d'un livre est compris entre 15 et 60 euros. Pour celles et ceux qui veulent investir du temps, mais pas d'argent, il reste le Web qui regorge de sites, mais il faut passer un certain temps à chercher les bonnes adresses. En 1990, lorsque j'ai dû apprendre par moi-même à utiliser mes premiers logiciels, puis mes premiers langages de programmation, j'ai rencontré les mêmes difficultés d'apprentissage que mes étudiants aujourd'hui (et puis le Web n'existait pas encore, il n'y avait pas les forums et, en dehors de la bibliothèque de la cité des sciences à Paris, les bibliothèques municipales ne proposaient que très peu de publications dans ces domaines). Je me souviens encore de ces soirées et nuits passées à essayer de comprendre certaines notions élémentaires... Cela me permet donc maintenant d'avoir des approches pédagogiques différentes selon les techniques à transmettre.

Chaque nouvelle explication présentée dans ce livre s'accompagnera d'un exemple simple. Seulement une ou deux notions nouvelles seront utilisées à la fois afin que vous compreniez bien. Notre approche ne sera pas systématiquement exhaustive : trop d'informations ou des informations trop précises empêchent bien souvent la compréhension d'une nouvelle notion.

Un livre dédié à la programmation orientée objet ou séquentielle ?

Qui peut le plus peut le moins ! Cela résume l'approche globale de ce livre…

À la lecture de la table des matières de cet ouvrage, vous pouvez découvrir qu'une part importante est consacrée à l'apprentissage de la programmation orientée objet, non seulement au travers de chapitres dédiés à ce mode de programmation, mais également au travers de nombreux exemples.

Lorsque vous apprendrez une nouvelle notion ou technique, nous vous expliquerons le mécanisme général, puis s'en suivra généralement un exemple de script à placer sur une image-clé. Lorsque cela s'avèrera nécessaire, nous vous présenterons également le script d'un fichier .as.

Tous les exemples de ce livre sont téléchargeables à l'adresse suivante :

```
http://www.yazo.net/eyrolles
```

Nous ne privilégions aucun mode de programmation car nous utilisons les deux dans la plupart des cas, mais nous mettons simplement en avant le script, qui est plus facile à appréhender. Pour les plus expérimentés d'entre vous, il sera très simple de copier-coller les lignes d'instructions dans un fichier externe lorsqu'un script en POO ne sera pas présenté. Rassurez-vous, pour les scripts les plus complexes de ce livre, nous vous présenterons un exemple dans chacun des deux modes de programmation.

Quelques termes à connaître

Si vous ne connaissez pas les termes employés en programmation, il est alors indispensable que vous lisiez les définitions ci-dessous. Attention, elle ne sont volontairement pas conformes à des définitions officielles que nous aurions pu trouver dans des livres spécialisés sur la programmation ou des sites dédiés, dans la mesure où la volonté d'être compris de tous constitue notre principal objectif. Des sites comme `http://www.commentcamarche.net` vous donneront sûrement les informations que vous pourrez rechercher.

Lorsque nous allons expliquer toutes les notions et techniques abordées dans ce livre, nous emploierons certains termes dont voici le sens.

Instance : il existe deux types d'instances qui résultent de deux opérations différentes. Si vous placez un symbole sur la scène, vous obtenez une occurrence, mais nous pourrions également dire qu'il s'agit d'une instance du symbole. En ActionScript, vous utiliserez parfois le mot `new` qui permet d'obtenir une instance de classe. Une instance est donc représentée par un mot. Lorsque vous aurez besoin d'écouter un son dans une animation, vous créerez une instance de la classe `Sound()` que vous manipulerez (charger un son dans cette instance, jouer le son de cette instance, régler le volume de cette instance).

Propriété : imaginez une instance (ou occurrence) sur la scène. Si vous souhaitez contrôler sa position, sa transparence ou sa rotation, il faudra alors écrire une ligne d'instruction

qui fera référence au nom de l'instance et à la propriété à modifier. La transparence d'une occurrence constitue une des propriété d'une instance, au même titre que son échelle horizontale, sa rotation, etc. En programmation orientée objet, ce terme s'étend à d'autres termes d'un script que nous n'évoquerons pas ici pour l'instant.

Méthode : dans un script ou un programme, vous écrirez certains termes avec des parenthèses à la fin. Ces derniers sont qualifiés de fonctions ou méthodes et nous pourrions les comparer aux verbes d'une phrase. Leur rôle est d'agir dans un programme ; par exemple : créer un rectangle sur la scène, changer la casse d'un texte, régler le volume d'un son, charger une image sur la scène, etc. Toutes les méthodes sont des fonctions, mais toutes les fonctions ne sont pas des méthodes. Dés que vous écrivez…

```
function marcher() {
  //ligne d'instruction à exécuter
}
```

il s'agit d'une fonction. Mais, dans certains cas, si cette fonction est précisément écrite à l'intérieur d'une classe, on la qualifie alors de méthode. Dans l'exemple ci-dessus, nous venons de créer une méthode, c'est-à-dire que vous expliquez au programme ce qu'il devra faire lorsque le mot `marcher()` sera écrit seul dans une ligne d'instruction.

Événement : pour temporiser un script, vous devez utiliser un événement. Le fait de survoler ou de cliquer sur une occurrence constitue un événement. De la même façon, si vous voulez exécuter une action lorsque la lecture d'un son est terminée, vous ferez appel à l'événement `soundComplete`. Un événement est un mot qui sera placé à un endroit fixé dans une ligne d'instruction. Pour être plus précis, cette ligne d'instruction est généralement placée dans un gestionnaire d'événement. Nous n'expliquerons pas ce terme car le chapitre 3 de ce livre est consacré intégralement à cette notion.

Ligne d'instruction : elle est comparable à la phrase d'un paragraphe dans un texte écrit en français. Au même titre qu'une phrase contient un sujet, un verbe, un complément, etc., une ligne d'instruction peut contenir un nom d'occurrence, une propriété, une méthode, des mots-clés, etc. Une ligne d'instruction se termine toujours par un point-virgule (qui est très souvent omis par les développeurs Flash néophytes).

Classe : c'est un regroupement de lignes d'instructions saisies avec une syntaxe précise pour structurer une partie d'un programme. Cette notion est pour l'instant trop abstraite pour lui donner davantage de sens.

Package : c'est une structure du code qui englobe une classe. Cette notion est également trop abstraite, pour l'instant, pour lui donner davantage de sens.

L'affichage de résultats sur la scène, dans les exemples de ce livre

Pour afficher un texte dans une animation Flash, il existe deux solutions, mais seule la première des deux techniques que nous allons aborder ci-dessous permet de visualiser un contenu textuel sur la scène, dans une animation au format SWF. La deuxième technique n'est valable qu'à partir du moment où vous travaillez dans l'IDE de Flash.

Un texte dynamique sur la scène

À de nombreuses reprises, nous aurons besoin d'afficher des textes sur la scène, nous utiliserons donc la méthode suivante :

1. Créer un texte sur la scène avec l'outil Texte (maintenez un clic sur la scène, faites glisser votre souris pour définir une largeur de texte, relâchez le bouton de votre souris).
2. Dans la partie inférieure gauche de la palette Propriétés, si cela n'est pas déjà le cas, sélectionnez le type Texte dynamique (ou Texte de saisie).
3. Nommez l'occurrence obtenue (exemple : `affichageResultat`).
4. Cliquez sur la scène puis dans la fenêtre Actions afin de saisir la ligne d'instruction ci-dessous :

```
affichageResultat.text = 3+45;
```

Comme vous pouvez le remarquer, nous utilisons la chaîne de caractères `.text` derrière le nom de l'occurrence, car nous faisons référence à la propriété `text` de l'instance pour stocker l'information (le résultat du calcul). Celles et ceux qui avaient l'habitude d'utiliser un nom de variable en AS1/AS2 noteront que cette technique n'est plus valable.

La fonction trace()

Il existe aussi une deuxième solution qui consiste à afficher une information dans la fenêtre Sortie de Flash.

Placez la ligne d'instruction ci-dessous pour tester cette fonctionnalité :

```
trace("Bonjour");
```

ou encore :

```
trace ("Bonjour, nous sommes le "+new Date().getDate());
```

Vous aurez peut-être remarqué que nous avons utilisé l'opérateur + dans les parenthèses de la fonction `trace()` pour pouvoir effectuer une concaténation, c'est-à-dire un regroupement de plusieurs informations.

Table des matières

1

Introduction

Avant d'aborder l'ActionScript 3 en détail, nous allons essayer de comprendre l'évolution qu'a connu ce langage à travers celle du logiciel Flash.

Historique de Flash et du langage ActionScript

Flash est bien *né* en 1996, comme tout le monde le dit, mais sa création remonte en fait à 1993. Cette année-là, la société Futurewave (créée par Jonathan Gay, Michelle Welsh et Charlie Jackson) publie le logiciel SmartSketch dédié à la création d'interface graphique. À l'époque il ne s'agit pas encore d'un logiciel d'animation avec une *timeline* (un scénario) et une bibliothèque. Rappelons que le Web en est à ses débuts et que pour créer un site Internet, il faut avant tout connaître le langage HTML et écrire le code soi-même ; les premiers logiciels WYSIWYG (What You See Is What You Get), tels que Claris Homepage ou Dreamweaver, n'existent pas encore.

Il faut attendre 1995 pour que l'application SmartSketch soit dotée d'un système de gestion d'images sur une échelle temporelle (timeline ou scénario). Elle change également de nom pour être rebaptisée FutureSplash Animator, plus connue sous le nom de FutureSplash. En décembre 1996, la société Macromedia (anciennement Macromind jusqu'en 1992) rachète FutureSplash et le rebaptise Flash : il s'agit de la version 1.0.

Afin de mieux comprendre les étapes qu'ont traversées le logiciel Flash et son langage dédié (ActionScript) au cours de ces 10 dernières années, passons en revue les différentes évolutions au travers des dates clés de l'histoire du logiciel.

Palette, panneau ou fenêtre ?

Nous allons évoquer à plusieurs reprises ces trois différents termes pour désigner généralement la même chose, c'est-à-dire une palette (palette Couleurs, palette Aligner, etc.). Rappelons que Macromedia a toujours utilisé le terme panneau (depuis Flash 5) pour désigner une palette. Par convention, tous les éditeurs de logiciels ont toujours utilisé le terme palette, et même dans les suites CS3, Adobe utilise ce terme pour désigner les panneaux que nous trouvons encore dans Flash. Dans la version CS4, les panneaux seront-ils rebaptisés en palettes ? Nous le verrons bien...

1996 – Flash 1

La société Macromedia rachète le logiciel FutureSplash et le rebaptise Flash 1.0.

1997 – Flash 2

Le logiciel gère les images bitmap et le son stéréo. Le symbole de type Bouton fait son apparition ainsi qu'une Bibliothèque. Cependant, la gestion des images et des images-clés du scénario reste difficile par manque d'ergonomie, même pour un utilisateur habitué de Director, le logiciel phare de l'époque pour le développement de produits off-line.

1998 – Flash 3

Un système d'interactivité est proposé. On ne parle pas encore de langage de programmation, mais de gestion des actions. Notons également que la timeline (scénario) devient plus facile à manipuler.

1999 – Flash 4

Cette nouvelle version marque un premier tournant dans l'histoire de Flash. Le langage des actions propose de nouvelles fonctions et une meilleure gestion des événements. L'interface est légèrement revisitée et l'accès au panneau gérant les différentes actions associées à une image ou à une occurrence est plus simple et plus rapide. Flash 4 propose une nouvelle barre d'outils, qui commence à ressembler davantage à celle que nous connaissons aujourd'hui, et qui contient trois outils dédiés au dessin (le trait, l'ellipse et le rectangle). Il devient également possible de se déplacer plus facilement sur la scène avec la main et la loupe. La gestion du son utilise le format MP3, mais il n'est toujours pas possible de le contrôler via une action.

2000 – Flash 5

Le tournant évoqué à propos de la version précédente n'est rien en regard des évolutions proposées dans Flash 5 : cette année-là, Macromedia nous présente un nouveau Flash !

Une fenêtre intitulée Actions (sur objet ou sur image) apparaît pour la première fois et permet à l'utilisateur de saisir directement du code. Il devient possible de bénéficier d'une aide lors de la rédaction de ses propres scripts.

Le menu déroulant, dans lequel nous avions jusqu'à présent l'habitude de sélectionner nos différentes commandes pour définir une action, est toujours disponible, mais la saisie de lignes d'instructions s'avère bien plus ergonomique ; la productivité est ainsi optimisée.

Flash 5 connaît une refonte complète de son interface en totalisant 21 palettes et change d'icône et d'écran d'accueil (*splashscreen*).

Un nouveau type de symbole est également créé : le Smart Clip. Il s'agit d'un clip auquel sont ajoutées quelques options afin de le transformer en symbole préprogrammé : c'est l'ancêtre du composant.

2002 – Flash MX

Macromedia change la numérotation de son logiciel (tous les produits de sa gamme adoptent le même changement). Il n'existera donc pas de version Flash 6, mais on parle tout de même du *player* Flash 6 ou lecteur Flash 6. Pour renforcer cette mutation, Macromedia uniformise l'aspect de toutes les icônes des ses logiciels ainsi que leurs écrans d'accueil.

Un nouveau système d'agencement des palettes permet un ancrage de celles-ci ; il s'agit d'un rangement par alignement avec magnétisme.

Sans le savoir, mais sûrement en l'espérant d'un point de vue marketing et technologique, Macromedia va faire évoluer partiellement Internet. À cette époque, Apple, Microsoft et Real proposent déjà depuis plusieurs années des solutions de diffusion de flux audio et vidéo, mais la possibilité d'incorporer une vidéo au format FLV dans une animation change la façon de gérer la vidéo sur le Web. En effet, l'interactivité avec ce média devient nettement plus facile et sa consultation est paradoxalement plus accessible. Le plug-in Flash connaît alors en 2002 un taux de pénétration très important. Précisons que le codec vidéo utilisé est le Sorenson Spark, mais pas encore le On2 VP6.

Côté programmation, non seulement la fenêtre Actions change, mais on peut à présent parler d'un vrai langage avec une syntaxe pointée. Les composants, qui font leur apparition et permettent d'optimiser davantage la productivité du travail dans Flash, facilitent ainsi la gestion des éléments de formulaires.

Le choix des lettres M et X avait, en 2002, fait l'objet de nombreuses évocations sémiologiques. La plus connue était celle de maximum représenté par MX. L'intention première était de contracter *Multimedia eXperience* en MX.

2003 – Flash MX 2004

Le langage ActionScript 2.0, apparu dans cette version, commence enfin à acquérir ses lettres de noblesse. Des développeurs provenant de langages plus traditionnels tels que le C ou Java se penchent sur l'AS (ActionScript) qui propose une approche de programmation orientée objet. Il devient possible de créer ses propres classes au travers de fichiers externes (qui portent l'extension `.as`).

De nouveaux composants dédiés aux médias et aux données facilitent et optimisent le développement pour les utilisateurs aguerris.

Le lecteur Flash évolue vers la version 7.

2005 – Flash 8

Du point de vue de la programmation, cette version du logiciel n'aura pas apporté de grandes nouveautés. En revanche, pour les utilisateurs au profil de graphiste, nous pouvons dire que Flash 8 aura marqué un grand tournant dans son histoire en proposant des filtres (comme l'avait fait Photoshop 2.5 avec les calques).

Nous n'allons pas vous présenter une liste exhaustive de toutes les nouveautés de cette version, mais voici celles qui ont changé les habitudes de tous les utilisateurs du logiciel :

- La vidéo fait appel au nouveau codec On2 VP6 qui propose un rapport poids/qualité extrêmement proche du H264. Le composant FLVPlayback facilite davantage encore la gestion de ce média dans une animation.
- Les palettes sont rangées par groupes et des onglets pour y accéder font leur apparition.
- Il devient possible d'appliquer des filtres et des options de surimpression à des occurrences. Cela modifie ainsi les habitudes de production graphique au travers de l'interface du logiciel.
- Le format PNG est enfin géré à partir d'un chargement dynamique exécuté en ActionScript.
- Une fenêtre gère les accélérations et décélérations dans une interpolation.
- Utilisation de la mise en cache des bitmaps à l'exécution.

2005 – Adobe achète Macromedia

À la fin de l'année 2005, pour faire face au géant de l'informatique Microsoft, Adobe décide de racheter la société Macromedia, propriétaire entre autres du standard Flash, avec un objectif précis : étendre son monopole de l'édition.

Cela fait alors plus de dix ans qu'Adobe est leader sur le marché de la publication papier, mais l'entreprise peine à conquérir le marché du Web. Ses tentatives, en 2001, d'imposer deux nouveaux logiciels sur le marché de la production online, Live Motion et Golive, respectivement des concurrents directs de Flash et Dreamweaver ont échouées. En novembre 2003, Adobe décide de ne plus distribuer Live Motion. Quant à Golive, le constat est simple : il ne fait partie d'aucune des suites CS3. Sur le site d'Adobe, à ce jour, rien n'est notifié quant à un éventuel retrait du marché, la société invite tout de même les utilisateurs à passer à Dreamweaver.

En rachetant la société Macromedia, Adobe a réussi à s'imposer comme leader sur le marché du Web et possède aujourd'hui suffisamment de technologies et logiciels pour pouvoir tenir tête à Microsoft.

Rappel

Pour information, voici la liste des logiciels/technologies que détient la société Adobe : Flash, Photoshop, Acrobat, Dreamweaver, Premiere, Illustrator, After Effects et InDesign pour ne citer que les principaux. Elle possède au total plus de 70 logiciels/technologies.

2006 – Lecteur Flash 9

En août 2005, deux semaines après la publication de Flash 8, Macromedia annonce la sortie du langage ActionScript 3.0 et du lecteur 8.5. Il faudra tout de même attendre la sortie de Flash CS3 avant de pouvoir développer avec ce langage (les développeurs Flex utilisaient déjà la syntaxe de l'ActionScript 3 plusieurs mois avant la sortie de la version CS3).

2007 – Flash CS3

Nouveau grand tournant dans l'histoire de Flash ! Adobe annonce non seulement l'édition de la nouvelle version du logiciel, mais également l'existence de 5 Creative Suites : Design Premium, Design Standard, Master Collection, Production Premium (suite de logiciels dédiés à la production vidéo), Web Premium et Web Standard.

Les grands changements qui accompagnent la sortie de Flash CS3 sont les suivants :

- L'ActionScript 3.0 : nouveau langage de programmation complètement orienté objet (même s'il est toujours possible d'adopter une approche de développement séquentielle). Les gestionnaires d'événements de l'AS3 s'uniformisent par rapport aux versions précédentes (l'AS1 et l'AS2).
- Possibilité d'importer un document Photoshop (ou bien même Illustrator et Freehand) avec ses calques.
- Barre d'outils sur une seule colonne pour un gagner de la place.
- Les palettes peuvent véritablement être ancrées sur plate-forme OS X.
- La palette Propriétés propose le rattachement d'un document ActionScript, qui utilise l'extension `.as`, comme classe du fichier.

Bien entendu, il ne s'agit pas d'une liste exhaustive, mais uniquement des principales nouvelles fonctionnalités qui vont, une fois encore, changer nos habitudes de production.

10 ans de Flash en images

Afin de mieux comprendre les propos ci-dessus, voici quelques copies d'écran des différentes versions du logiciel Flash. Vous noterez que les évolutions se sont souvent produites par paliers. Rares sont les nouveautés proposées dans une version, qui n'ont pas été ensuite reprises dans les versions ultérieures.

Remarque

Les copies d'écran des pages qui vont suivre ont été réalisées à partir de l'ordinateur de l'auteur de cet ouvrage, en lançant les applications les unes après les autres. Pour des raisons de droits d'auteurs, nous n'avons pas pris la décision de faire figurer des copies d'écran du logiciel Future Splash Animator disponibles sur Google © Image. Vous noterez également que les copies des écrans d'accueil de Flash MX 2004 à Flash 8 correspondent à des versions Professional ; il en existe également pour les versions Basic.

Les icônes

Traditionnellement, une application est représentée par une icône en forme de losange. Macromedia décide de ne plus respecter cette convention à partir de Flash MX. Notons également qu'il n'y a pas de distinction entre les versions Basic et Professional à partir de Flash MX 2004.

Figure 1-1

Les icônes du logiciel Flash n'ont cessé de changer d'aspect en 10 ans.

Les écrans d'accueil

On peut observer trois grandes périodes pour les écrans d'accueil : de 1997 à 2001, le même écran est utilisé pour quatre versions successives du logiciel (Flash 2 à Flash 5).

Figure 1-2

Sur la copie d'écran du logiciel Flash 3 Beta testé, le numéro de version figure en petit à droite du logo Macromedia. Sur la version finale, le chiffre 3 figurait à droite du mot Flash, comme pour les versions 4 et 5.

Figure 1-3

Macromedia conserve le même écran d'accueil jusqu'à Flash 5.

À partir de 2002, le fameux *f* de Flash fait son apparition : on le retrouve notamment sur les icônes du logiciel et le rouge commence à devenir la couleur de référence pour représenter ce produit de la gamme.

Figure 1-4

Une nouvelle génération d'écrans d'accueil apparaît à partir de 2002, avec le fameux f symbolisant le logiciel Flash.

C'est avec le rachat de Macromedia par la société Adobe que cette dernière décide de redéfinir une identité visuelle pour l'ensemble des produits qu'elle propose au travers de ses différentes suites. Le rouge est conservé, mais le *f* disparaît au profit des deux premières lettres de Flash : Fl.

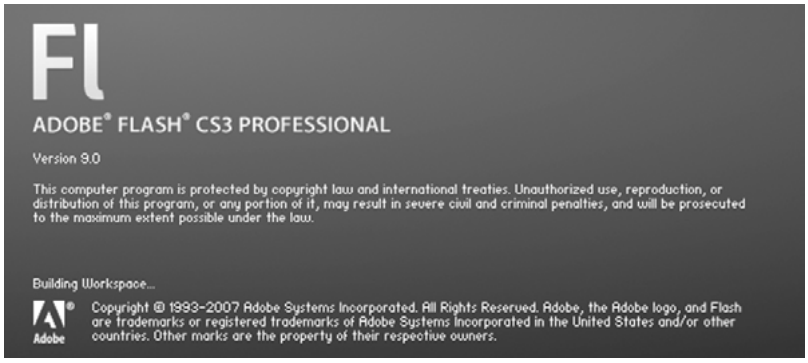


Figure 1-5

Adobe impose une nouvelle identité visuelle ; la paire de lettres Fl représente désormais le logiciel Flash parmi tous les produits que la société possède.

Les barres d'outils

On notera un premier petit changement avec Flash 4, puis une très grande évolution avec Flash 5 (souvenez-vous, nous avons vu que cette version avait marqué un vrai tournant dans l'histoire du logiciel). À partir de la version MX, la plus grande nouveauté est l'arrivée de l'outil Transformation. Il faut ensuite attendre Flash CS3, pour découvrir une réorganisation propre à Adobe.

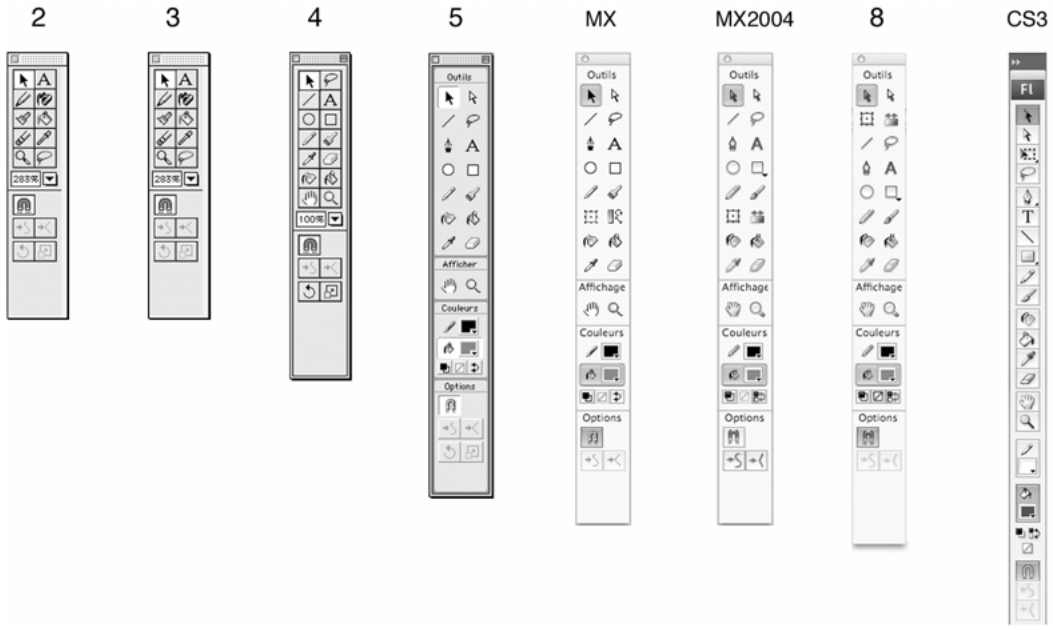


Figure 1-6

Il est à noter que la barre d'outils de Flash CS3 est également configurable sur deux colonnes.

Le scénario (timeline)

Le principal défaut que nous pouvions reprocher à Flash 2 était son manque de souplesse dans la manipulation du scénario. Depuis la première version, les notions d'images et d'images-clés ont toujours existé, mais la manipulation des cellules du scénario a été simplifiée uniquement à partir de Flash 3.

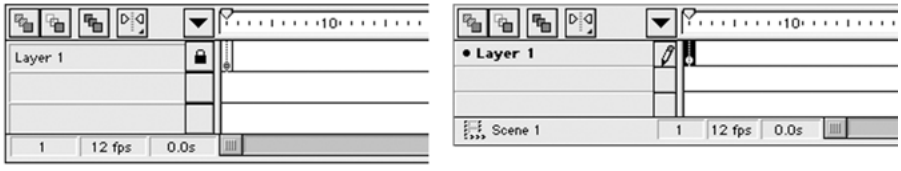


Figure 1-7

Scénarios de Flash 2 et 3. Des versions minimalistes et presque fonctionnelles !

Comme nous l'avons évoqué au début de ce chapitre, c'est à partir de Flash 5, en 2000, que nous avons connu un réel changement, notamment au niveau de l'ActionScript.

Cependant, comparez bien les figures 1-7 et 1-8 : vous pouvez constater que c'est un an plus tôt, en 1999 avec Flash 4, qu'une nouvelle structure de la palette Scénario donne le ton pour toutes les versions à venir. Macromedia avait réservé en 1999 des locaux majestueux à Disneyland Paris pour convoquer toute la presse à un petit déjeuner et une présentation du produit. Il est important de rappeler qu'à l'époque, Flash n'était pas encore un logiciel très connu.

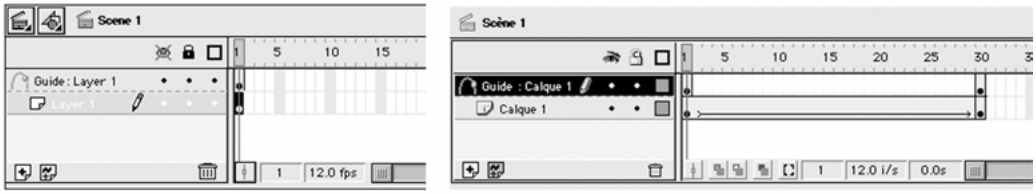


Figure 1-8
Scénarios de Flash 4 et 5

Il aura fallu attendre l'année 2002 pour voir apparaître l'icône d'un dossier jaune en bas à gauche de la fenêtre Scénario. À partir de cette date, il devient enfin possible d'organiser ses calques en les regroupant comme nous pouvions déjà le faire depuis plusieurs années dans des logiciels comme Photoshop ou Illustrator. Cette nouveauté va radicalement changer les habitudes de production au quotidien.

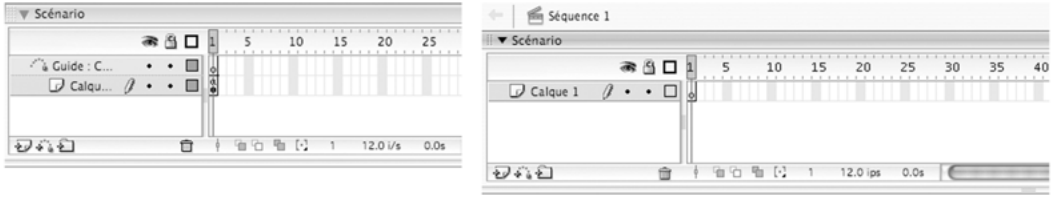


Figure 1-9
Scénarios de Flash MX et MX 2004

Figure 1-10
Scénario de Flash 8

Figure 1-11

Scénario de Flash CS3

En conclusion, si vous observez attentivement les copies d'écran des figures 1-7 à 1-10, vous constaterez qu'à partir de Flash 4 les fonctionnalités de cette palette n'ont pas changé. C'est uniquement l'apparence graphique de chaque nouvelle version qui possède plus ou moins son propre style.

La palette Propriétés

La palette Propriétés est celle qui a connu le plus grand nombre de transformations au fil du temps. En effet, dans Flash 2, on ne parlait pas vraiment de panneau et à peine de propriétés. Il s'agissait d'une fenêtre où il était possible d'effectuer quelques réglages.

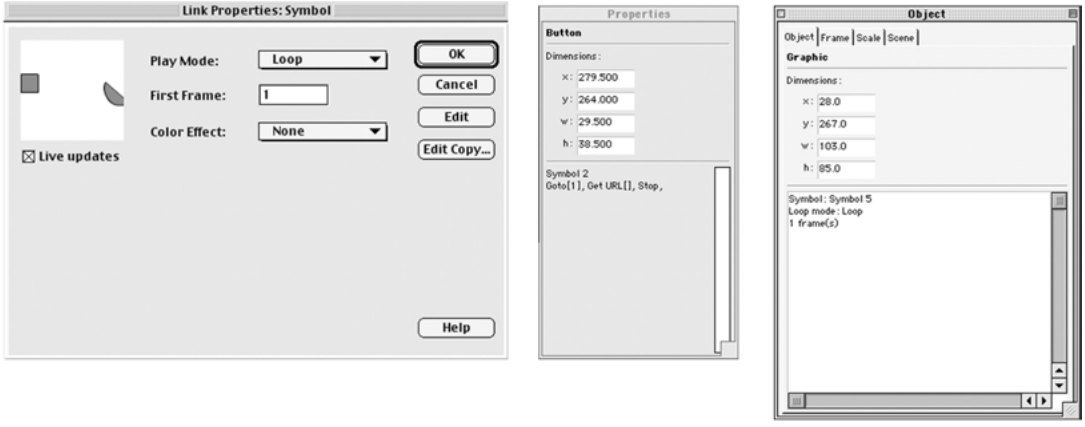


Figure 1-12

Palettes Propriétés de Flash 2, 3 et 4

La grande révolution aura été l'arrivée de palettes pour gérer les paramètres des différentes composantes d'une animation. Il devient non seulement possible d'accéder rapidement aux options de réglage des différentes propriétés de l'animation, des clips, des textes… mais nous disposons également d'une plus grande précision d'ajustement des paramètres.

Macromedia adopte ce système de palettes bien connu des autres éditeurs de logiciels, mais la société n'a pas encore inséré la fameuse barre horizontale très ergonomique qu'on va découvrir à partir de Flash MX (figure 1-14).

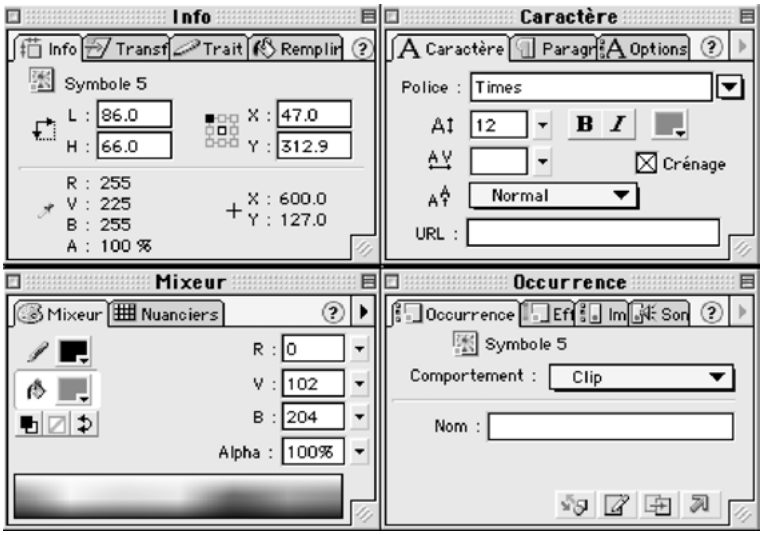


Figure 1-13
Palette Propriétés de Flash 5

Figure 1-14
Palette Propriétés de Flash MX

Les versions MX et MX 2004 présentent très peu de changements au niveau de l'interface du logiciel. Il s'agit de deux versions très proches, où même l'icône du logiciel reste identique. Rappelons que seul le langage ActionScript subit un profond changement en évoluant vers la version 2.0.

Figure 1-15
Palette Propriétés de Flash MX 2004

À partir de Flash 8, l'insertion d'onglets dans les groupes de panneaux permet un accès et un affichage plus rapides des palettes. Notons également l'ajout d'une option de mise en cache des bitmaps à l'exécution.

Figure 1-16

Palette Propriétés (d'un clip) de Flash 8

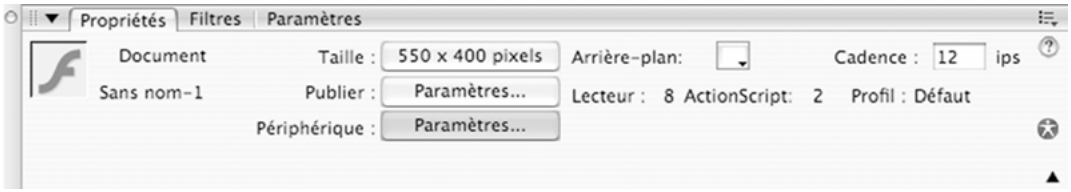


Figure 1-17

Palette Propriétés (du document) de Flash 8

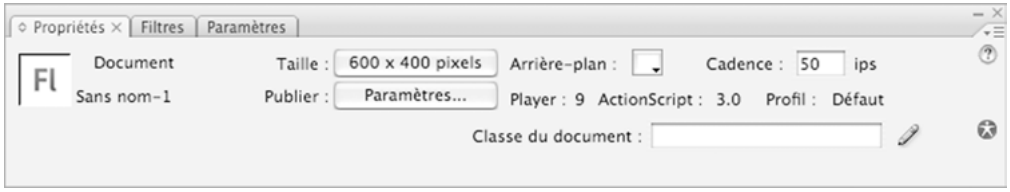


Figure 1-18

Palette Propriétés (du document) de Flash CS3

La Bibliothèque

L'apparence de la Bibliothèque a globalement subi peu de modifications. Seuls les boutons de sélection d'affichage des types de symboles disponibles dans les versions 2, 3 et 4 de Flash ont disparu dans les versions suivantes pour laisser très rapidement la place à un classement par noms ou types.

À partir de Flash 5, l'arrivée des dossiers a permis d'augmenter la productivité du travail en rendant enfin possible le classement des symboles (fig. 1-19).

À partir de Flash 8, un menu déroulant est proposé dans la palette afin de pouvoir afficher la bibliothèque d'une autre animation (fig. 1-20).

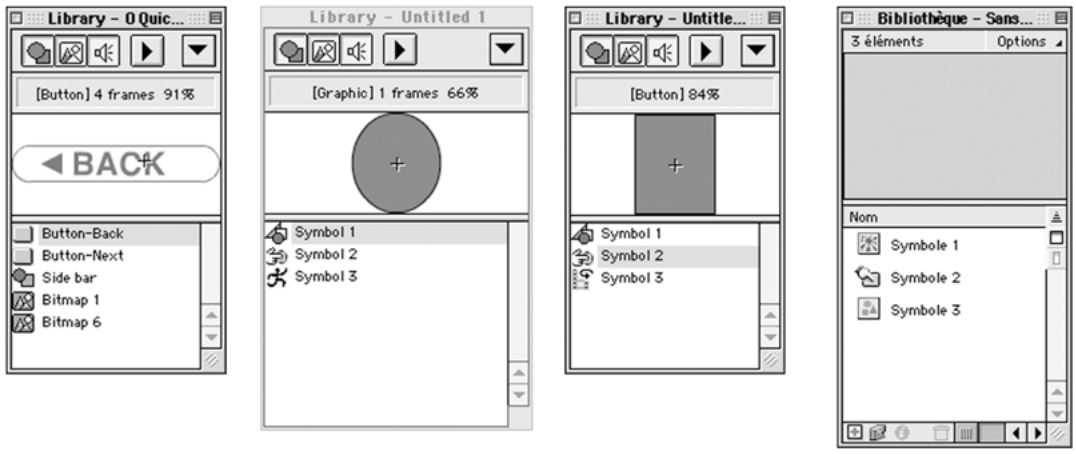


Figure 1-19

Les bibliothèques de Flash des versions 2, 3, 4 et 5

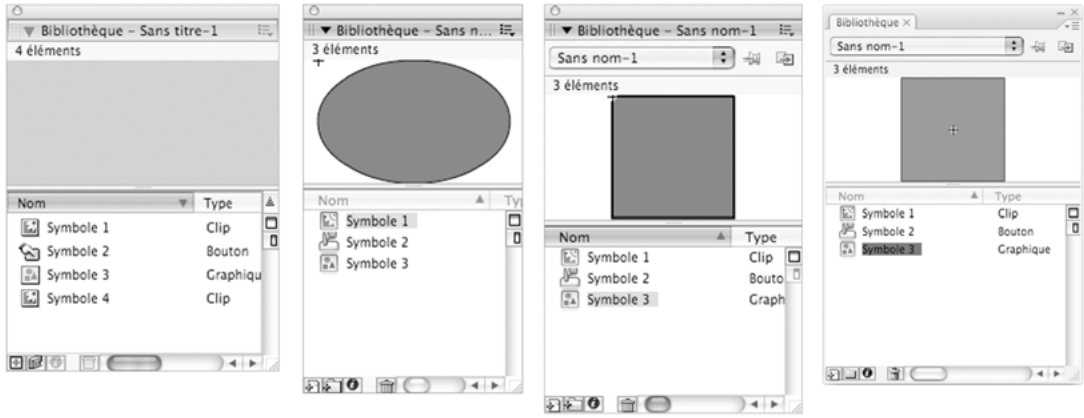


Figure 1-20

Les bibliothèques de Flash des versions MX, MX 2004, 8 et CS3

Globalement, la bibliothèque a toujours conservé les mêmes fonctionnalités : regrouper sous forme de liste l'ensemble des symboles disponibles dans une animation et prévisualiser ces derniers dans une vignette située dans la partie supérieure de la palette.

Les actions

L'évolution de la gestion de l'ActionScript au travers des interfaces des différentes versions de Flash est particulièrement intéressante. Les premières versions du logiciel ne comportaient pas réellement de langage de programmation. Il s'agissait plutôt d'un système de

gestion de scripts associés à un bouton ou à une image-clé. L'ajout d'une action, c'est ainsi que nous parlions de programmation, permettait d'apporter de l'interactivité à une animation au moyen de termes, assimilables aux instructions d'un langage, peu nombreux et accessibles seulement à partir de sous-fenêtres. L'interface de Flash était en fait peu adaptée à cette activité.

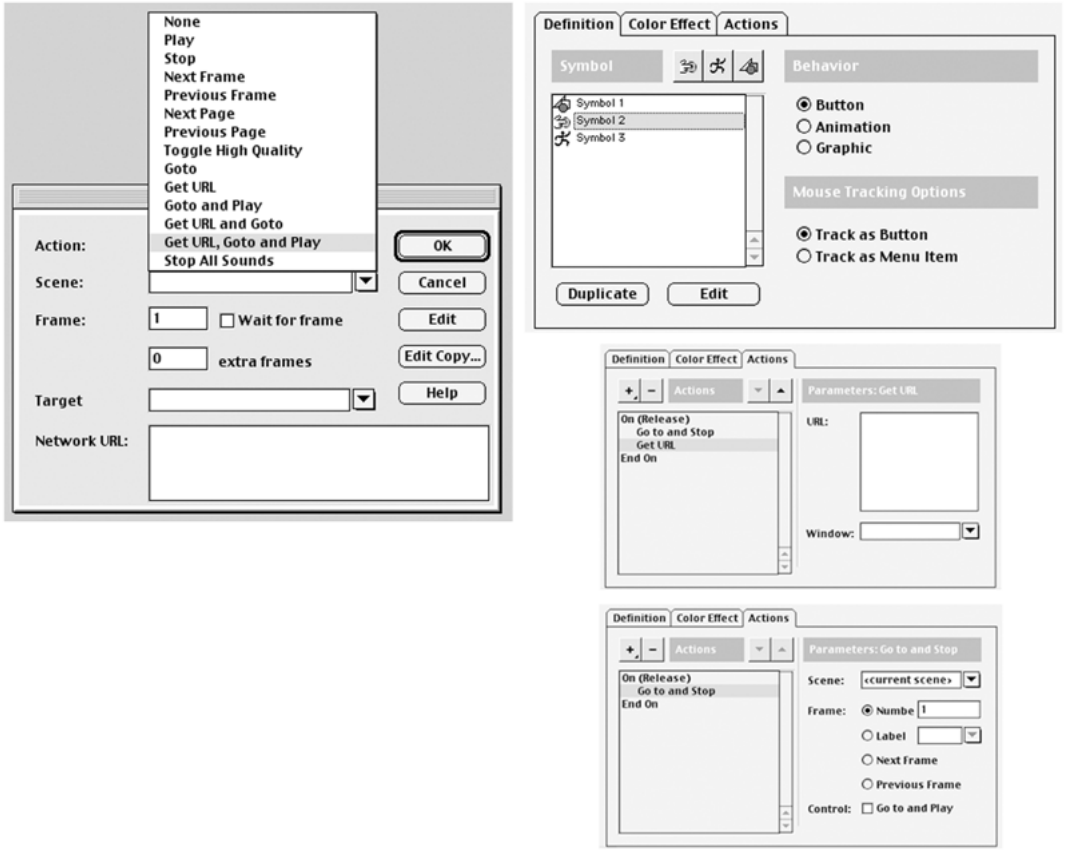


Figure 1-21

Système de gestion des scripts dans Flash 2 et 3

La version Flash 4 a été dotée d'une série importante de termes supplémentaires. C'est à partir de cette version que nous avons pu utiliser des variables, régler dynamiquement les propriétés d'une occurrence, mener des tests, écrire des itérations ou encore charger des images externes.

La gestion des événements a également été introduite dans Flash 4. Aujourd'hui, avec du recul, nous pouvons considérer que cette version du logiciel aura été la première à proposer un réel système de programmation.

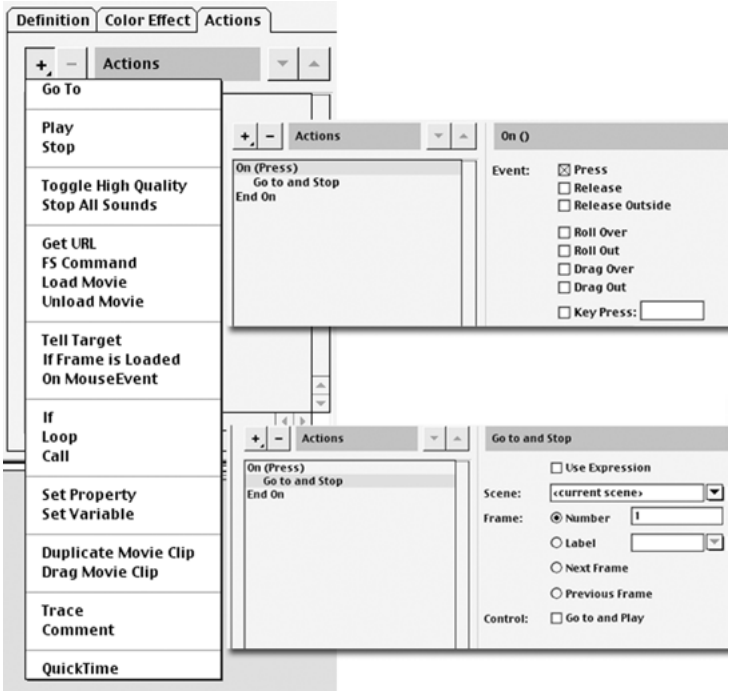


Figure 1-22

Système de gestion des scripts dans Flash 4

Flash 5 offre enfin la possibilité aux utilisateurs de programmer leurs propres scripts (on ne parle pas encore de développement Flash). Une fenêtre est dédiée à cette gestion de l'interactivité et le langage ActionScript 1 naît vraiment en 2000. Ce dernier comporte même des classes natives pour gérer le son, la date, la couleur, etc. Une aide à la saisie de script continue d'exister, comme dans les version précédentes, mais avec une meilleure ergonomie.

Quelques mois après la sortie de Flash 5, on commence à trouver de plus en plus de sites Internet dédiés à cette technologie, qui n'est pas encore qualifiée ainsi à l'époque. Rappelons que, pendant de nombreuses années, l'ActionScript a été boudé par la communauté des développeurs à cause de son manque de standardisation par rapport à d'autres langages typés tels que Java ou C.

Les scripts sont placés sur les occurrences de clip et de bouton ainsi que sur les images-clés. L'expansion que connaît l'ActionScript à cette époque est importante : la communauté des développeurs Flash commence à peine à se construire et se structurer, le langage de programmation se complexifie très rapidement, il est enfin possible d'écrire de réelles applications interactives sur Internet, les premiers sites extraordinaires, au sens étymologique du terme, commencent à être connus... L'ActionScript 1 va ainsi vraiment faire sortir Flash de l'ombre et le plug-in Flash commence peu à peu à s'imposer.

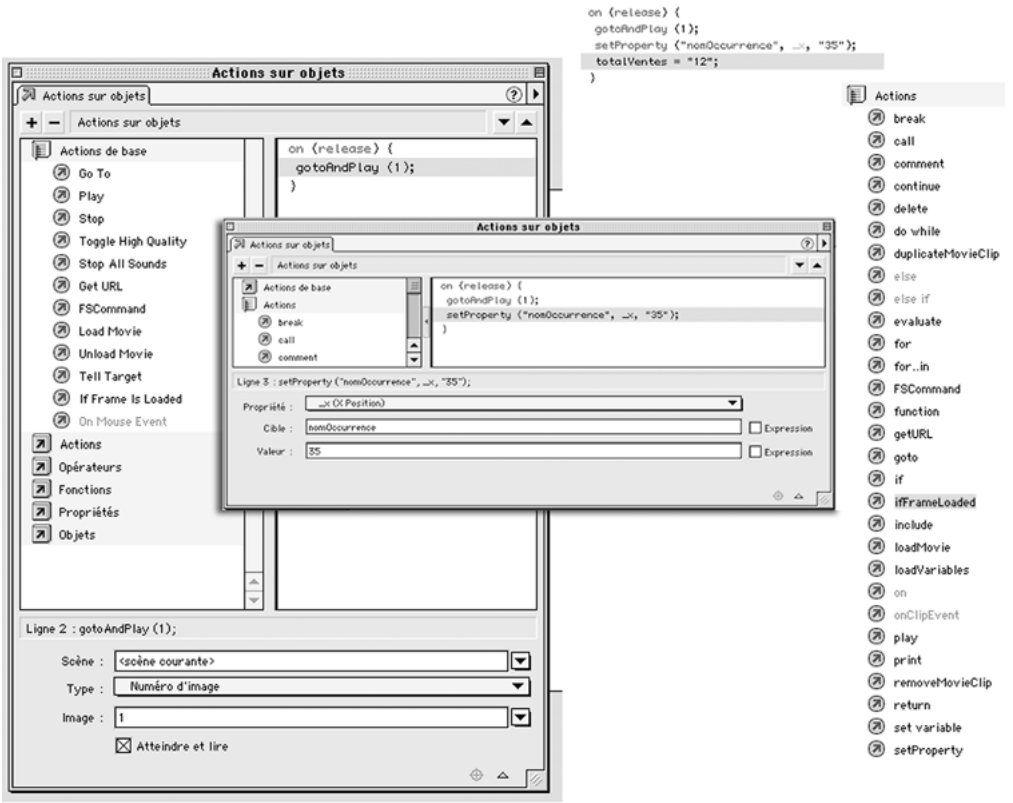
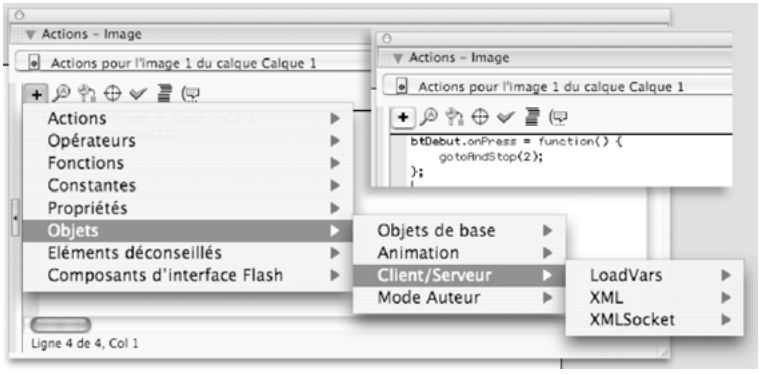


Figure 1-23
Fenêtre Actions de Flash 5

En 2002, Flash MX est dotée d'une nouvelle interface, mais le langage est poussé dans ses derniers retranchements. La communauté des développeurs Flash essaye de trouver des solutions pour pallier ses limites, malgré l'ajout de nouvelles classes.

Figure 1-24
Fenêtre Actions de Flash MX



La version Flash MX 2004 s'accompagne d'une nouvelle version du langage : l'ActionScript 2. Les limites rencontrées dans Flash MX sont enfin repoussées car il est à présent possible de créer ses propres classes dans des fichiers externes.

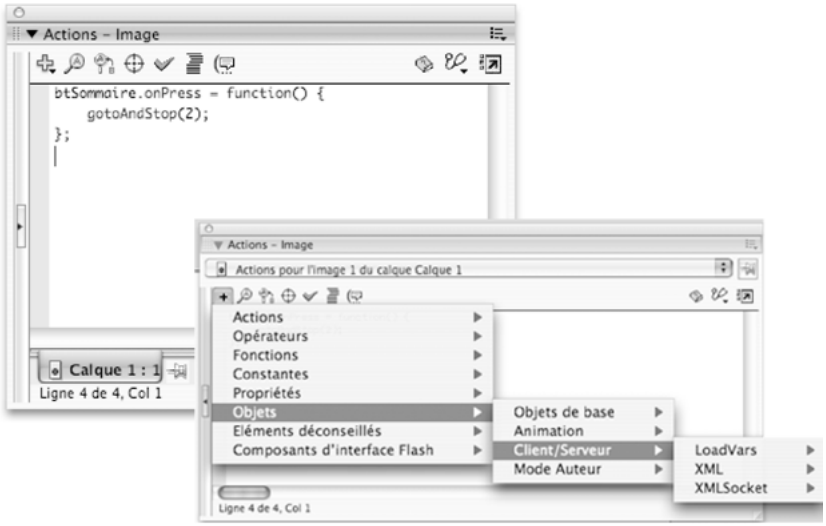


Figure 1-25

Fenêtre Actions de Flash MX 2004

Comme nous l'avons vu précédemment, Flash 8 aura été la version des graphistes. En effet, toutes les nouveautés de cette version auront été relatives aux attentes des créatifs, avec très peu de changements pour les développeurs.

En revanche, Flash CS3 (Flash 9) propose enfin l'ActionScript 3, la version annoncée près d'un an auparavant et donc très attendue.

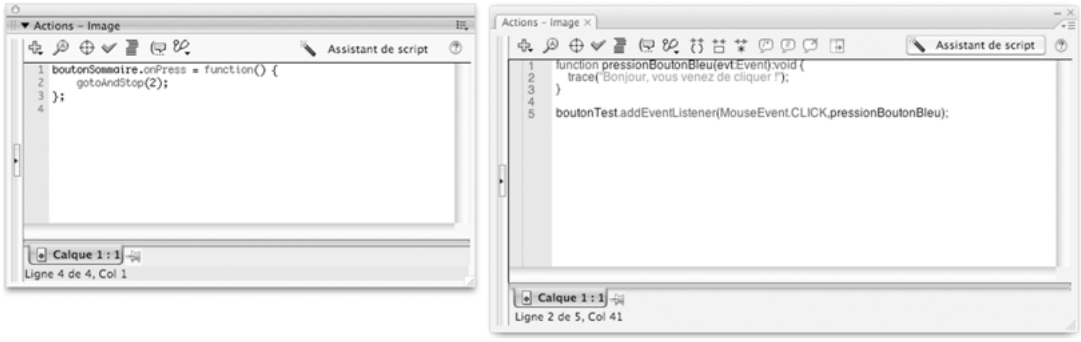


Figure 1-26

Fenêtres Actions des versions Flash 8 et CS3

Aujourd'hui, l'ActionScript 3 est un vrai langage orienté objet comparables aux langages de référence.

Conclusion

Qui aurait pu imaginer tout le chemin parcouru par un si petit logiciel ? Finalement, comme nous l'évoquions au cours de cette présentation, Flash aura mis du temps à acquérir ses lettres de noblesse. Encore aujourd'hui, avec la percée d'Ajax, certains détracteurs pensent que Flash ne sortira pas indemne du combat débuté il y a environ un an. Pour information et rappel, Ajax est une solution de développement qui utilise plusieurs technologies/standards, alors que Flash n'en représente qu'une et gère parfaitement et simplement la vidéo et le son. Ajax et Flash ne sont pas des solutions de développements concurrentes, mais complémentaires. La qualité d'un développement dépend toujours des compétences du développeur !

Précisons que Flash possède une mauvaise réputation auprès d'un certain public. Ceci est dû au fait que, parmi les nombreux développements assurés en Flash, tous ne sont pas réalisés par des spécialistes de l'AS3 ou de l'AS2. Avec la dernière version de l'ActionScript 3 et l'engouement d'une tranche de la population des développeurs d'origine Java ou C, le langage Flash devrait enfin connaître un réel succès. À titre de comparaison, et sachant que l'AS3 est proche à plus de 90 % du langage de Flex, on notera que Flex a déjà commencé à s'imposer auprès des développeurs chevronnés.

Au travers de cette rétrospective, nous n'avons pas évoqué les différentes dates de création des logiciels ou technologies tels que Generator, Flash Media Server (anciennement Flash Communication Server), Flash Remote, FlashPaper, FlashLite, Flex et Appolo car nous nous serions écartés de notre propos initial, l'histoire exclusive de Flash.

Les deux modes de programmation

Comme nous l'avons évoqué au cours des premières pages de l'avant-propos de ce livre, il existe deux modes de programmation. Puisque l'ActionScript 3 peut s'adapter à ces deux approches, il est naturel de s'interroger sur le mode à utiliser.

Il est très difficile, voire impossible de conseiller à un individu l'un des deux modes tant qu'on ne connaît pas son expérience personnelle et professionnelle dans le domaine de l'informatique en général, et dans celui de la programmation en particulier. Ainsi, il serait absurde de conseiller la programmation séquentielle à un adolescent, sous prétexte qu'il n'a jamais programmé auparavant ou qu'il est trop jeune. Cela n'aurait pas de sens non plus de préconiser une approche orientée objet à un développeur Web, sous prétexte qu'il utilise du XHTML et des CSS depuis de nombreuses années pour concevoir des sites Internet. Tout dépend des aptitudes de chacun à écrire dans un langage différent de son langage habituel.

Le seul conseil raisonnable que nous pouvons vous donner est de commencer à apprendre les deux modes en vous appuyant sur les premières pages de ce livre ; par la suite, vous choisirez tout naturellement celui qui vous semblera le plus logique. Quoi qu'il en soit, le mode de programmation séquentiel est plus facile à prendre en main car son utilisation est moins contraignante. Cependant, si vous êtes amené à utiliser l'ActionScript de façon intensive dans les mois et/ou années à venir, vous n'avez pas le choix : adoptez le mode de programmation orientée objet.

Conseil

En tant qu'enseignant et développeur Flash, je préconise de bien connaître les deux modes de programmation car ils donnent lieu à deux types de production. Lorsque vous aurez besoin de réaliser une simple animation avec une légère interactivité, il sera inutile de créer et de faire appel à une classe document. En revanche, lorsque vous devrez réaliser un projet assez important, nous ne saurions vous conseiller autre chose que la programmation orientée objet.

Si vous ne parvenez pas à maîtriser la programmation orientée objet après quelques tentatives infructueuses, optez alors pour la programmation séquentielle. Il sera toujours possible, même après plusieurs mois de pratique, de basculer finalement vers la programmation orientée objet.

Quelle que soit l'approche que vous choisissez, gardez toujours à l'esprit que vous devez longuement réfléchir à la fois aux objectifs que vous souhaitez atteindre à travers votre programmation, mais également aux fonctionnalités que doit posséder votre animation. Sachez que la programmation est un métier qui s'appuie sur des règles, normes, techniques et conventions.

Les deux développements ci-après ont pour objectif d'expliquer les spécificités de chaque mode.

La programmation séquentielle ou structurée

Avant de commencer à saisir son premier script…

Pour ajouter de l'interactivité à un bouton, ici une simple réaction au clic de souris, nous allons faire appel aux fenêtres Scénario et Actions. Commencez tout d'abord par organiser votre espace de travail afin de disposer de suffisamment de place dans la fenêtre Actions. La copie d'écran de la figure 1-27 vous montre un exemple de configuration de l'environnement de travail.

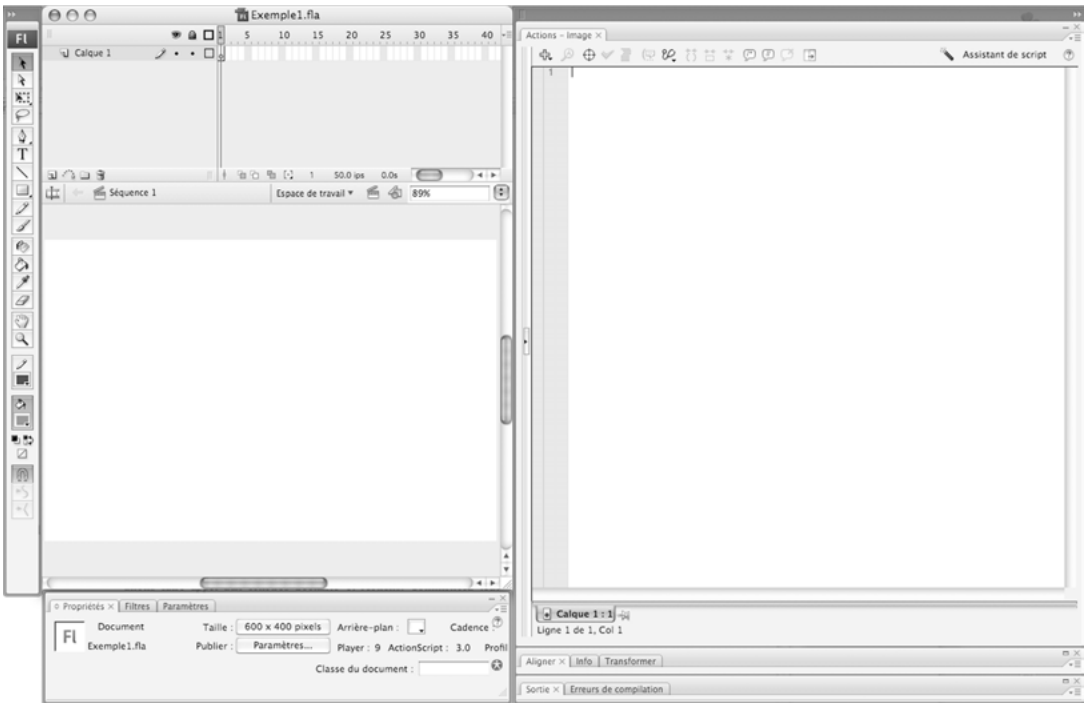


Figure 1-27

Donnez une place suffisamment grande à la fenêtre Actions afin de pouvoir saisir confortablement votre code ActionScript.

Nous conseillons de conserver présentes à l'écran les palettes Sortie et Erreurs de compilation afin de ne pas déstructurer la configuration de votre espace de travail. Par ailleurs, n'oubliez pas qu'un clic sur la barre de titres, dans laquelle se trouvent les onglets des palettes, permet d'afficher ou de masquer une palette comme le montre la figure 1-28.

Figure 1-28

Pour masquer ou afficher un ensemble de palettes, un simple clic sur la barre de titres suffit.

Lorsque vous aurez terminé de configurer votre espace de travail, nous vous conseillons de mémoriser cette disposition de la façon suivante :

1. Dans le menu Fenêtre et dans le sous-menu Espace de travail sélectionnez la commande Enregistrer la présentation active…

Astuce

Un double-clic sur l'outil de la main (dans la barre d'outils) permet de régler la taille d'affichage de la scène de façon à la rendre intégralement visible dans l'espace disponible entre la fenêtre Actions et la barre d'outils. Un double-clic sur l'outil de la loupe (dans la barre d'outils) permet de régler rapidement la taille d'affichage de la scène à 100 %, c'est-à-dire à l'échelle 1:1 de votre animation à l'écran.

2. Spécifiez un nom de configuration que vous pourrez alors retrouver en sélectionnant la commande éponyme dans le menu Fenêtre, commande Espace de travail.

Dans la copie d'écran de la figure 1-27, vous constaterez que nous n'avons pas affiché la fenêtre Bibliothèque pour gagner un peu de place dans l'interface. Un simple Commande-L (Mac) ou Ctrl-L (Windows) permet d'afficher rapidement la Bibliothèque de l'animation. Si vous souhaitez la conserver affichée à l'écran, ancrez-la à l'un des deux groupes de panneaux situés en bas à droite de l'écran.

Ancrer une palette

L'ancrage (le terme anglais est *to dock*) d'une palette est très simple : il suffit de la saisir en cliquant sur son onglet, puis de la faire glisser sur un groupe d'autres palettes (figure 1-29).

Figure 1-29

Avec Flash CS3, il redevient enfin possible de gérer le regroupement des palettes par un simple glisser-dépacer.

Par ailleurs, pour éditer rapidement un symbole, vous n'êtes pas obligé de passer par la fenêtre Bibliothèque : il est en effet utile de mémoriser que le bouton situé en haut à droite de la fenêtre contenant la scène (figure 1-30), permet un accès rapide aux symboles de l'animation en cours.

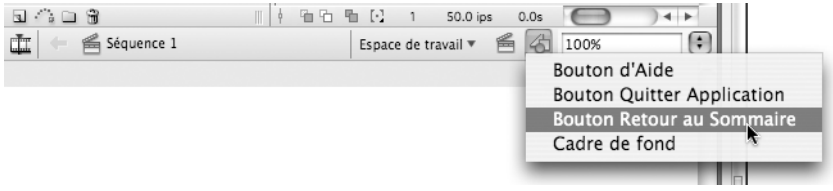


Figure 1-30

Le bouton situé en haut à droite de la fenêtre contenant la scène permet un accès rapide aux symboles de l'animation en cours.

Saisir son premier script

Voyons à présent comment procéder pour écrire un premier script dans une animation Flash. Pour cet exemple, précisons simplement que nous avons nommé `boutonAlert` une occurrence de symbole de type clip. Cette dernière sera utilisée dans le deuxième exemple.

Nommer une occurrence

Commencez par cliquer sur une occurrence de la scène puis, dans la palette Propriétés, cliquez dans la case de saisie qui contient le texte temporaire `<Nom de l'occurrence>`. Saisissez alors le nom d'instance (ou occurrence) que vous souhaitez ; il ne doit contenir aucun caractère accentué ou spécial ni le caractère espace.

Figure 1-31

Vous ne pouvez programmer une occurrence que si vous l'avez nommée.

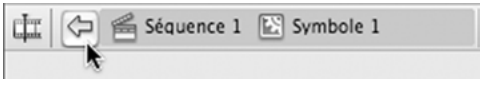


Figure 1-32

Un clic sur ce bouton permet de quitter la fenêtre d'édition d'un symbole.

Édition involontaire d'un symbole

Certaines personnes ont parfois le réflexe de double-cliquer pour sélectionner une occurrence au lieu d'effectuer un simple clic. Cette action vous place alors dans la fenêtre d'édition du symbole. Pour en sortir, il suffit d'effectuer un simple clic sur le bouton qui se trouve en haut à gauche de la fenêtre qui contient la scène (figure 1-32). Le raccourci-clavier Commande-E (Mac) ou Ctrl-E (Windows) permet également de revenir sur la scène.

Premier exemple

1. Cliquez sur une image-clé du scénario (figure 1-33).
2. Cliquez dans la fenêtre Actions (figure 1-33).
3. Saisissez votre script.

Exemple de script :

```
stop();
trace("Bonjour tout le monde !");
trace("Ceci est un premier exemple");
```

Attention : si vous cliquez directement dans la fenêtre Actions sans avoir sélectionné une image-clé, il est fort probable que vous saisissiez un script alors qu'une occurrence est encore sélectionnée. Vous allez, dans ce cas, générer une erreur.

Figure 1-33

Commencez par cliquer sur une image-clé du scénario, puis dans la fenêtre Actions avant de saisir votre script.

> **Conseil**
>
> Il est astucieux de créer un calque nommé, par exemple, Actions ou Scripts et de dédier ses images-clés à la saisie du ou des scripts de l'animation.

Lorsque cette procédure en trois points est terminée, vous devez lire l'animation pour vérifier la validité du code. À l'aide du raccourci clavier Commande-Entrée (Mac) ou Ctrl-Entrée (Windows), une fenêtre d'exécution du code apparaît.

Notre premier script est très simple : il consiste, tout d'abord, à bloquer la tête de lecture afin qu'elle n'aille pas plus loin dans le scénario (ce qui n'est pas nécessaire si l'animation ne comporte qu'une seule image, comme dans notre exemple). Ensuite, nous avons provoqué l'affichage, dans la fenêtre Sortie, de deux messages : ils sont placés entre guillemets et entre parenthèses.

Deuxième exemple

À présent, imaginons que nous souhaitions afficher ces deux messages lors d'un clic sur une occurrence de la scène.

> **Attention**
>
> Avant de poursuivre votre lecture, veillez à créer un symbole de type clip et à nommer l'occurrence obtenue, comme nous l'avons fait au début de ce développement.

Il suffit alors de remplacer le code précédent par celui ci-après :

```
function afficherMessage(evt:MouseEvent) {
  trace("Bonjour tout le monde !");
  trace("Ceci est un premier exemple");
}
boutonAlert.addEventListener(MouseEvent.CLICK,afficherMessage);
```

Ainsi, les deux messages ne sont plus affichés au lancement de l'animation, et nous contrôlons à présent l'instant de leur apparition.

Sans autre explication, il est normal que ce script vous semble complexe. Vous trouverez au chapitre 3 de très nombreuses pages consacrées au fonctionnement des gestionnaires. Dans ce deuxième exemple, nous avons simplement cherché à vous démontrer l'importance de la place d'un script dans une animation Flash en mode de programmation séquentielle.

Qu'est-ce qu'une fonction ?

Il s'agit d'une notion élémentaire en programmation qui consiste à écrire plusieurs lignes d'instructions entre des accolades, ensemble d'opérations associé à un terme (en fait, le nom de la fonction) que vous devez définir. Lorsque vous avez besoin d'exécuter les lignes de code contenues dans une fonction, vous faites alors simplement référence à son nom.

ATTENTION : note aux néophytes en matière de programmation

Si cette première explication de l'ActionScript vous semble trop abstraite, ne terminez pas la lecture de ce chapitre et passez directement au troisième.

La programmation orientée objet

La programmation orientée objet constitue un mode et un modèle de développement très particulier. Dans les lignes qui vont suivre, nous n'allons pas vous expliquer les bases fondamentales de la POO (Programmation orientée objet) qui nécessiteraient un développement de plusieurs dizaines de pages, mais plutôt vous en présenter une approche à travers l'ActionScript 3. Afin que nos explications puissent être comprises de tous, nous allons commencer par exposer la structure d'un package à l'aide d'un exemple précis. Dans la suite de cet ouvrage, nous découvrirons progressivement différents exemples où nous expliquerons alors certaines notions dans leurs contextes.

Avant de commencer à saisir son premier script…

Si vous ne connaissez pas encore l'ActionScript, nous vous conseillons d'utiliser l'interface de Flash pour écrire vos premiers scripts. En revanche, si vous êtes déjà familier avec un autre environnement de développement intégré (IDE) qui supporte l'AS3, n'hésitez pas à le conserver. D'une façon générale, si vos activités vous conduisent à passer de nombreuses heures par semaine à développer en AS3, nous pouvons vous conseiller d'utiliser l'IDE FlashDevelop. Si vous êtes un utilisateur d'Eclipse, téléchargez le plug-in qui vous permettra de développer dans cet environnement.

Définition

Un IDE est un environnement de développement intégré (Integrated Development Environment) qui propose à l'utilisateur un éditeur de code, des outils automatiques de fabrication, un compilateur servant à transformer le ou les fichiers source en un code exécutable et, enfin, un débogueur. Flash est donc un IDE qui possède également une GUI (Graphical User Interface) qui constitue l'interface graphique de l'outil de développement.

La classe document

Si vous avez lu les explications relatives à l'emplacement d'un script en programmation séquentielle, vous avez pu constater que le code est associé à une image-clé à partir de la fenêtre de scénario.

En adoptant une approche de développement orientée objet, l'image-clé ne constitue plus l'emplacement privilégié pour coder une animation : nous allons alors utiliser des fichiers externes. Pour débuter, nous devons créer un document de la façon suivante :

1. À partir de Flash, créez un nouveau document de type Fichier Flash (AS 3.0), afin que nous puissions réaliser les différentes parties de notre interface (figure 1-34).

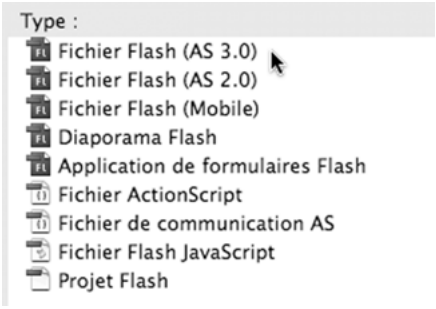


Figure 1-34

Différents types de nouveaux documents peuvent être crées. Les types Fichier Flash (AS 3.0) et Fichier ActionScript sont nécessaires en programmation orientée objet.

> **Remarque**
>
> Lorsque vous aurez davantage de facilités à programmer en AS3, ce premier document ne vous servira peut-être plus. Vous préparerez alors les différentes parties de votre interface dynamiquement à partir du langage ActionScript.

2. Dans le menu Fichier, activez la commande Enregistrer sous…
3. Donnez un nom de fichier comme `Exemple1.fla`.

   Votre document principal est maintenant terminé. D'après ce que nous disions au début de cette section, nous devrions à présent disposer sur la scène les différents symboles de la bibliothèque pour construire notre interface et ainsi donner forme à notre animation. Dans un souci de simplicité, nous n'allons rien placer sur la scène dans ce premier exemple.
4. Créez à nouveau un document en sélectionnant cette fois-ci le type Fichier ActionScript.
5. Dans le menu Fichier, activez la commande Enregistrer sous…
6. Attention, le nom que vous allez à présent donner au fichier est important. Rappelons qu'il ne doit pas contenir de caractères spéciaux ou accentués, ni même d'espace. Par exemple, `AfficherMessage.as` est un nom valide.

À partir de ce moment-là, vous devez posséder deux documents ouverts dans l'IDE de Flash, comme le montre la figure 1-35. Vous noterez que, par défaut, le fichier FLA est associé à notre document de type ActionScript. Il figure dans le menu déroulant situé en haut à droite de la fenêtre de code.

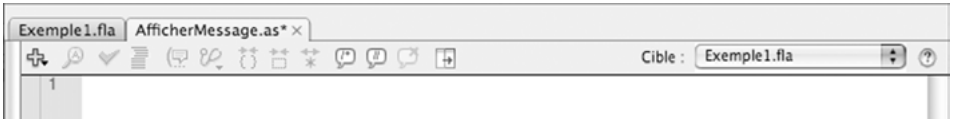


Figure 1-35

L'interface de Flash contient deux onglets, en haut à gauche de votre écran.

7. Revenez sur le document `Exemple1.fla` en cliquant sur l'onglet éponyme.
8. Dans la palette Propriétés, saisissez le mot `AfficherMessage` à l'emplacement du paramètre Classe du document (figure 1-36).

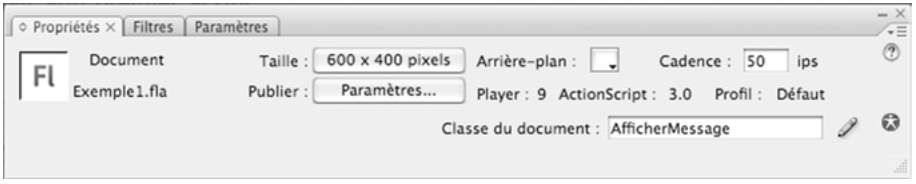


Figure 1-36

Spécifiez le nom du document ActionScript qui contient la classe du document.

Il ne nous reste plus à présent qu'à rédiger le script lui-même, c'est-à-dire le contenu du fichier `AfficherMessage.as`.

Créer son premier package

Tout d'abord, vous devez comprendre que les lignes d'instructions que nous nous apprêtons à saisir font appel à des règles de programmation très précises. Chaque terme a une signification et joue un rôle très important dans le fonctionnement du programme. À la fin de ce chapitre, nous développerons la notion de `displayList`. Il est indispensable que vous réussissiez à visualiser mentalement les organigrammes qui y sont proposés. Vous pourrez ainsi appuyer vos développements sur la hiérarchie des classes de l'ActionScript 3 et sur le système de gestion d'affichage des composants d'une interface (instances visibles sur la scène).

Exemple

Nous allons créer un fichier ActionScript dont le rôle est d'afficher un message de type texte dans la fenêtre Sortie de Flash ou sur la scène.

1. Cliquez dans la fenêtre principale (en haut à gauche) du document `AfficherMessage.as`, comme vous le feriez lorsque vous vous apprêtez à saisir un texte dans un logiciel de traitement de texte.
2. Saisissez le texte suivant :

```
package {

}
```

   De cette façon, vous définissez un package, c'est-à-dire un ensemble de lignes d'instructions regroupées en classes et autres fonctions. L'avantage de cette organisation est de pouvoir réutiliser ce code dans d'autres programmes ou animations. Vous rencontrerez parfois d'autres syntaxes qui emploient un autre mot après le terme package : il s'agit d'une hiérarchisation de l'organisation des différents packages.

3. À l'intérieur des accolades, vous créez une première classe comme ceci :

```
public class AfficherMessage extends Sprite {

}
```

Notez que les règles de nommage d'une classe sont les mêmes que pour les noms de fichiers. Veillez également à écrire le terme `class` avec une minuscule en début de mot et sans `e` à la fin.

La classe `AfficherMessage` ainsi créée est une extension de la classe `Sprite`, c'est-à-dire, en quelque sorte, la duplication d'une classe existante. Cette nouvelle classe possède les mêmes fonctionnalités de base que celle dont elle est l'extension, c'est-à-dire qu'elle hérite des événements, méthodes et propriétés de la classe `Sprite`. On parle alors de notion d'héritage.

Afin que le compilateur de Flash reconnaisse tous les termes relatifs à la classe `Sprite`, lorsque nous y ferons référence, nous devons lui ordonner d'exécuter l'instruction ci-dessous :

```
import flash.display.Sprite;
```

Compilateur

Lorsque vous travaillez dans Flash, il est nécessaire, à certains moments, de visualiser le résultat de votre production. Pour ce faire, à l'aide du raccourci clavier Commande-Entrée (Mac) ou Ctrl-Entrée (Windows), vous pouvez obtenir un fichier au format SWF qui est directement lu dans l'interface de Flash. Il est également possible de lire ce fichier dans le lecteur Flash (Flash Player) fourni avec l'environnement auteur de Flash (dans le sous-dossier `Players` du dossier Flash CS3).

Comment est généré ce fichier SWF que vous utiliserez comme source dans une page Web ou que vous transformerez en projecteur ? C'est justement le rôle du compilateur qui est donc un programme intégré à Flash. Au moment de la publication d'une animation, le compilateur de Flash lit les lignes d'instructions de votre animation afin de les interpréter pour générer le fichier au format SWF.

Lorsque le compilateur parcourt les différentes lignes d'instructions de votre fichier ActionScript, il réagira à la directive `import` en recherchant le contenu du fichier indiqué en paramètres. Il s'agit de lignes d'instructions en ActionScript.

Vous devriez obtenir pour l'instant le code suivant :

```
package {
  import flash.display.Sprite;
  public class AfficherMessage extends Sprite {
  }
}
```

Définition simplifiée d'une classe

Une classe est un ensemble de lignes d'instructions organisées en fonctions suivant leurs rôles. Chaque fonction contient, à son tour, un ensemble de lignes d'instructions.

Vous noterez l'ajout du mot `public` devant le terme `class`, par opposition à `private`. Nous reviendrons plus tard sur le rôle de ce mot-clé (attribut).

4. Nous devons maintenant définir le contenu de la classe, qui peut varier considérablement d'un programme à un autre. Cependant, une classe a généralement toujours la même structure. Elle contient un constructeur, aussi appelé fonction constructrice, et d'autres fonctions qualifiées de méthodes. Nous reviendrons sur ces notions un peu plus loin dans ce chapitre. Pour l'instant, ajoutez simplement `trace("Bonjour");` à l'intérieur de votre classe, comme ceci :

```
package {
  import flash.display.Sprite;
  public class AfficherMessage extends Sprite {
    trace("Bonjour");
  }
}
```

Attention à l'enregistrement de votre fichier ActionScript…

Vous devez systématiquement enregistrer votre fichier ActionScript après chaque changement de votre code. Dans le cas contraire, vous ne verriez pas les modifications s'exécuter lors de la publication de votre animation, c'est-à-dire au moment de la compilation de votre code.

Le constructeur

Dans ce premier exemple, nous provoquons simplement l'affichage de l'expression `Bonjour` dans la fenêtre Sortie de Flash, mais nous n'avons pas encore obtenu un code très conventionnel. En effet, de la même façon qu'un code est exécuté au passage de la tête de lecture sur une image-clé, nous devons créer un constructeur dans notre classe `AfficherMessage()`, c'est-à-dire une fonction dont le code sera exécuté au chargement de la classe ou du package.

Rappelons que, puisque nous avons indiqué le nom du fichier de la classe document dans la palette Propriétés (figure 1-36), un package sera chargé au lancement de l'animation. Voici le code que vous devez à présent ajouter à votre classe.

```
public function AfficherMessage() {
  trace("Bonjour");
}
```

Il s'agit du constructeur, qui doit porter le même nom que celui de la classe et donc du fichier ActionScript. La présence du constructeur n'est pas obligatoire, mais elle permet d'initialiser certains paramètres ou certaines actions lors de l'appel de la classe, c'est-à-dire au chargement de l'animation (rappelons que nous travaillons sur une classe du document). Voici le code global que vous devriez à présent obtenir :

```
package {
  import flash.display.Sprite;
  public class AfficherMessage extends Sprite {
    public function AfficherMessage() {
      trace("Bonjour");
    }
  }
}
```

Vous noterez la présence du mot clé `public` qui n'a pas d'importance dans la déclaration du constructeur, dans la mesure où ce dernier est par défaut toujours de type `public`.

Les méthodes de classe

Nous allons à présent créer une autre fonction, non pas en remplacement du constructeur, mais en supplément. Examinez la fonction suivante :

```
package {
  import flash.display.Sprite;
  public class AfficherMessage extends Sprite {
    public function AfficherMessage() {
      trace("Bonjour");
    }
    private function afficherHeure() {
      trace("Bonjour, il est 6h31");
    }
  }
}
```

Cette fonction, qui est habituellement appelée une méthode de la classe, joue un rôle très important car elle permet de définir une action à partir d'une étiquette. Pour appeler cette méthode, et donc exécuter le code correspondant, il suffit d'écrire dans le code le nom de la fonction (son étiquette) suivie de parenthèses, ici `afficherHeure()`, comme dans l'exemple suivant :

```
package {
  import flash.display.Sprite;
  public class AfficherMessage extends Sprite {
    public function AfficherMessage() {
      afficherHeure();
    }
    private function afficherHeure() {
      trace("Bonjour, il est 6h31");
    }
  }
}
```

Nous avons remplacé la seule ligne d'instruction contenue dans le constructeur par un appel de la méthode `afficherHeure`. Cette dernière peut être appelée à tout moment, uniquement à partir de la classe.

Remarque

Pour afficher correctement l'heure sur la scène au chargement de l'animation, il faudrait ajouter un texte dynamique sur la scène (nom d'instance : `zoneAffichage`) et remplacer la ligne d'instruction `trace("Bonjour, il est 6h31");` par les lignes suivantes :

```
var instantTemps:Date = new Date();
var heures:Number = instantTemps.getHours();
var minutes:Number = instantTemps.getMinutes();
zoneAffichage.text = "Il est "+heures+"h"+minutes;
```

L'ajout d'un texte dynamique sur la scène nécessite d'insérer la ligne ci-après à la suite du premier import de classe :

```
import flash.text.TextField;
```

private ou public ?

Dans notre exemple, nous ne définissons qu'une seule classe. Nous pourrions être amenés à en gérer plusieurs et, de ce fait, à utiliser, pour certaines méthodes et propriétés, des noms identiques. Afin que ces méthodes et propriétés ne soient accessibles que dans la classe où elles ont été déclarées, nous employons le spécificateur de contrôle d'accès `private`.

C'est précisément le cas dans notre script, où nous souhaitons que la fonction (la méthode) `afficherHeure()` de la classe `AfficherMessage` ne puisse être appelée qu'à partir de cette dernière et non pas d'une autre classe. Nous spécifions donc que la méthode est de type `private`.

En revanche, si nous souhaitions qu'une méthode déclarée dans une classe puisse être accessible à partir d'une autre, nous utiliserions le spécificateur de contrôle d'accès `public`.

Le fait qu'une méthode soit de type `public` ou `private` permet donc de définir sa portée, mais autorise également la réutilisation d'un nom identique dans d'autres classes pour définir de nouvelles méthodes aux fonctionnalités différentes.

Portée d'une fonction ou d'une variable

Nous entendons par portée d'une méthode, l'étendue et la disponibilité du code contenu dans la fonction (la méthode). Certains développements vous conduiront à créer plusieurs classes. Si vous souhaitez limiter l'action d'une méthode à une seule classe, vous allez en fait limiter la disponibilité de la méthode à cette classe, et donc restreindre sa portée. Le terme portée inclut une notion de propagation du code.

Consultez le chapitre 7 dédié aux variables pour des explications supplémentaires sur les mots-clés `private` et `public`.

Le chemin de classe

Dans l'exemple que nous venons d'aborder, nous n'avons eu à gérer qu'un seul fichier. Dans les projets plus lourds, vous devrez manipuler plusieurs fichiers ActionScript. Il deviendra alors nécessaire de les ranger dans des dossiers pour les organiser. Dans ce cas, avant de pouvoir importer vos classes, vous devrez spécifier à Flash, et plus précisément au compilateur, l'emplacement du ou des dossiers qui contiennent les fichiers ActionScript.

Pour cela, sélectionnez, dans le menu `Fichier` de Flash, la commande `Paramètres de Publication`. Cliquez sur l'onglet `Flash` (situé en haut de la fenêtre) et cliquez sur le bouton `Paramètres` (voir figure 1-37). Dans la fenêtre qui apparaît, vous devez alors spécifier le chemin des classes à gérer pour votre projet en cliquant sur le bouton contenant une cible.

> **Remarque**
> Vous noterez que la classe du document peut également être définie à cet endroit-là.

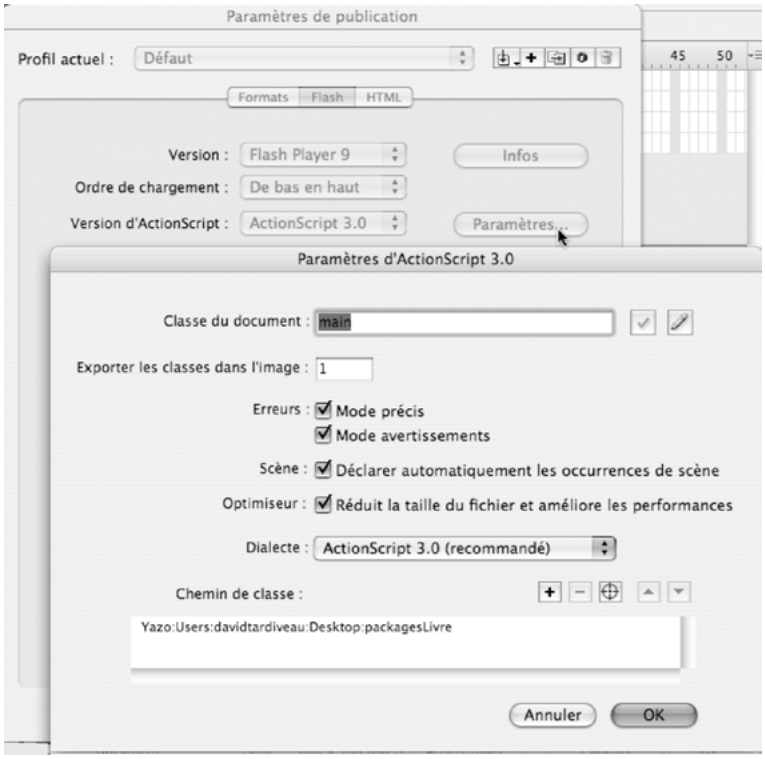


Figure 1-37

Spécifiez le chemin de classe afin que le compilateur de Flash sache où trouver les classes que vous lui demandez d'importer.

Les imports de classes et packages

Lorsque vous programmez directement dans la fenêtre Actions de l'interface de Flash, vous ne vous souciez pas d'un détail qui joue pourtant un rôle capital en mode de programmation orientée objet : l'import des classes et packages.

Prenons l'exemple d'un son que vous choisissez de jouer dans une animation. Vous saisissez tout naturellement vos lignes d'instructions dans la fenêtre Actions en appelant un certain nombre de méthodes des classes `Sound` et `SoundChannel`. Selon vous, comment Flash peut-il faire la distinction entre les termes qu'il connaît et ceux que vous créez (comme les noms d'instances) ? Par ailleurs, comment reconnaît-il un appel à une méthode précise d'une classe ?

Au démarrage de Flash, vous avez peut-être observé qu'en bas à gauche de l'écran d'accueil (figure 1-38), des textes se succèdent. Il s'agit du chargement des ressources dont le logiciel a besoin pour fonctionner. Parmi celles-ci, il en est une qui contient des informations capitales pour pouvoir programmer en mode séquentiel dans la fenêtre Actions de l'interface de Flash. Cette ressource se trouve dans un fichier texte intitulé `implicitImports.xml`, dont voici le contenu :

```
<implicitImportsList>
    <implicitImport name = "adobe.utils.*"/>
    <implicitImport name = "flash.accessibility.*"/>
    <implicitImport name = "flash.display.*"/>
    <implicitImport name = "flash.errors.*"/>
    <implicitImport name = "flash.events.*"/>
    <implicitImport name = "flash.external.*"/>
    <implicitImport name = "flash.filters.*"/>
    <implicitImport name = "flash.geom.*"/>
    <implicitImport name = "flash.media.*"/>
    <implicitImport name = "flash.net.*"/>
    <implicitImport name = "flash.printing.*"/>
    <implicitImport name = "flash.system.*"/>
    <implicitImport name = "flash.text.*"/>
    <implicitImport name = "flash.ui.*"/>
    <implicitImport name = "flash.utils.*"/>
    <implicitImport name = "flash.xml.*"/>
</implicitImportsList>
```

Il décrit les imports de classes et de packages indispensables en programmation orientée objet. Comme ce fichier est lu au lancement de Flash, il n'est pas nécessaire de débuter vos programmes par la liste de ces imports.

> **Remarque**
>
> Le défilement des textes au lancement de Flash étant trop rapide, la copie d'écran de la figure 1-38 ne correspond pas au chargement des ressources nécessaires en programmation orientée objet.

Si vous décidez de travailler en mode de programmation séquentielle, vous pouvez toujours éditer le fichier `implicitImports.xml` afin d'y inclure d'autres packages ou classes. Cela vous évitera de faire appel à la directive `import`.

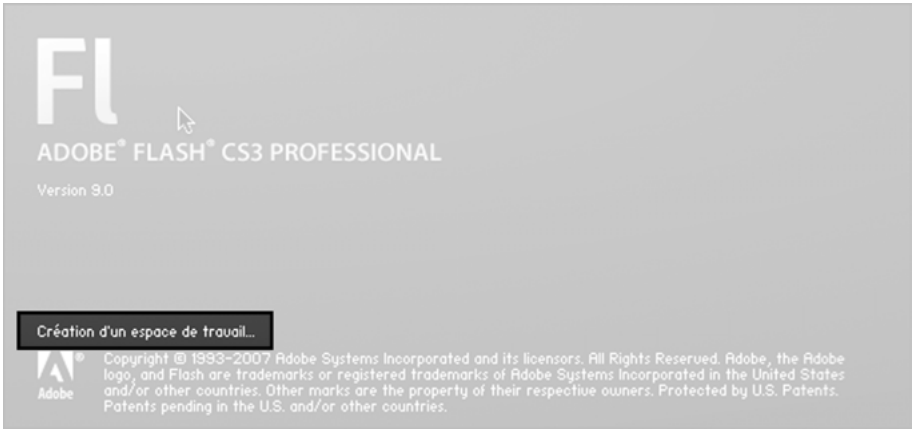


Figure 1-38

Le défilement des différents textes de la page d'accueil correspond au chargement des ressources nécessaires au fonctionnement de Flash.

Figure 1-39

Le fichier implicitImports.xml se trouve dans le dossier ActionScript 3.0 du dossier Configuration de l'application Flash CS3.

Lorsque vous rédigez un programme dans un fichier ActionScript, à partir de Flash ou bien même d'une application externe, vous devez utiliser la directive `import` afin de spécifier les noms de classes et/ou de packages à importer dans votre projet.

L'une des premières difficultés auxquelles vous aurez à faire face lors de la rédaction de vos premiers scripts sera de déterminer les classes à importer. Consultez la partie *Les packages et classes* de l'annexe de ce livre afin de découvrir les imports les plus courants.

Avantages et inconvénients des deux modes de programmation

Avant de vous présenter les avantages et inconvénients de ces deux modes de programmation, il est utile de prendre en considération le niveau du développeur. Comme nous l'avons indiqué précédemment, le choix entre ces deux modes est difficile, et dépend, notamment, de votre aptitude à comprendre la programmation en général.

La programmation structurée, appelée également séquentielle

Avantages

- Ce mode de programmation présente l'avantage d'être plus abordable et plus facile à comprendre que la POO.
- À court terme, vous serez donc plus productif car la POO nécessite une longue pratique pour atteindre une bonne maîtrise.
- Les animations sont plus rapides à développer si elles ne sont pas trop complexes.
- Si votre animation est bien réalisée (avec une certaine rigueur dans la rédaction de vos scripts), davantage de personnes pourront la corriger.
- Le code est contenu dans un seul fichier d'extension `.fla`. Il peut également être placé sur différentes images-clés, facilitant ainsi la gestion du déclenchement des scripts dans le temps.

Inconvénients

- Ce mode n'est pas adapté aux projets complexes.
- Le code présent sur des images-clés ne facilite pas le déboguage.
- Si vous n'utilisez pas les classes externes personnalisées (techniques propres à la POO), vous risquez de perdre du temps et être de ce fait moins productif.

La programmation orientée objet

Avantages

- Ce mode est adapté aux projets complexes, car son principe de programmation repose sur une organisation rigoureuse et méthodique des différentes parties d'un programme. Rigueur et méthode, deux qualités essentielles pour conduire un projet de grande envergure !
- Le code est stocké dans des fichiers externes, qui peuvent donc être réutilisés dans d'autres projets. Cela optimise, par la même occasion, la productivité.
- Il est plus facile de corriger ou de reprendre un projet développé en POO car la rigueur inhérente à ce mode de programmation a pour effet de générer des lignes d'instructions plus claires.

Inconvénients

- Ce mode de programmation est plus difficilement abordable pour un large public.
- Il est nécessaire de le pratiquer pendant de nombreuses semaines (voire des années) avant de pouvoir être vraiment opérationnel.
- Les animations réalisées en POO ne peuvent être reprises et corrigées que par des développeurs ayant le même niveau de compétence que vous en matière de programmation.

Comme vous pouvez le constater, les avantages et inconvénients de chacun des modes se valent.

2

La gestion des occurrences sur la scène

Afin d'optimiser votre code pour gérer les occurrences affichées sur la scène, il est important de comprendre le rôle et le fonctionnement de la liste d'affichage (displayList). Nous consacrons donc un chapitre à ce système de gestion. Nous présenterons ensuite la méthode `addChild()`, dont le but est de placer une instance de type objet d'affichage ou conteneur d'objets d'affichage sur la scène.

La liste d'affichage (displayList) d'une animation

Le fait de référencer tous les objets d'affichage et les conteneurs d'objets d'affichage présents dans une animation est la notion la plus élémentaire de l'ActionScript 3.

Dans les versions précédentes, nous devions indiquer le chemin d'une occurrence pour pouvoir l'utiliser. La hiérarchie qui existait entre les occurrences d'une animation découlait de l'emploi d'une succession d'imbrications d'instances de type MovieClip.

Une arborescence était tout de même constituée virtuellement dans la mémoire de l'ordinateur pour déterminer les parents et les enfants éventuels d'un objet d'affichage ou d'un conteneur d'objets d'affichage.

Avec l'arrivée de l'AS3, la gestion des occurrences affichées sur la scène est donc différente, plus souple et plus logique. Ne plus devoir cibler une instance, en précisant son chemin avec une syntaxe pointée, peut éventuellement troubler les anciens développeurs Flash. Cela dit, le développement consacré à la méthode `addChild()` à la fin de ce chapitre vous démontre la simplicité d'accès à une occurrence, même si elle se trouve imbriquée dans un ou plusieurs conteneurs d'objets d'affichage.

Par ailleurs, nous ne sommes plus limités à des clips et des boutons : il existe à présent plusieurs types d'objets regroupés en classes.

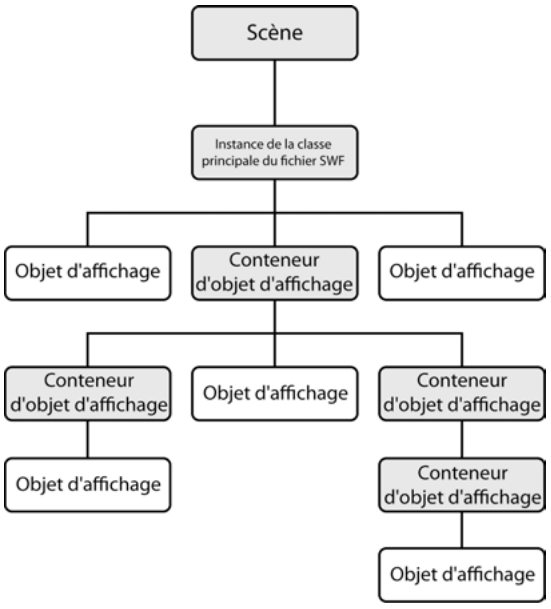
> **Remarque**
>
> Si ce dernier point vous paraît trop abstrait, ajoutons que chaque type d'objet possède son propre vocabulaire, c'est-à-dire un ensemble de mots réservés associé aux instances qu'il générera. Pour celles et ceux qui travaillaient avec Flash 8 et les versions antérieures, les symboles de type Bouton et ceux de type Clip constituent un bon exemple : ce sont deux objets de types différents.

Chaque fois que vous placez une occurrence sur la scène, manuellement (par glisser-déposer) ou dynamiquement (en AS, avec la méthode `addChild()` ou un moyen équivalent), vous alimentez une liste d'affichage, appelée également displayList. Celle-ci permet une gestion globale des occurrences de la scène. Il est ainsi possible de déterminer, par des instructions en ActionScript, le nombre d'imbrications d'occurrences (occurrences qui possèdent elles-mêmes d'autres occurrences), ce qui n'était pas le cas dans les anciennes versions de Flash.

Il est maintenant important de comprendre que vous pouvez disposer de deux types d'instances sur la scène : les occurrences qui peuvent en contenir d'autres (représentées par des rectangles gris sur la figure 2-1) et les occurrences qui ne peuvent contenir rien d'autre que la représentation graphique qui les symbolise (les rectangles blancs de la même figure). En d'autres termes, on distingue les occurrences qui peuvent posséder des enfants de celles qui ne le peuvent pas. La liste d'affichage des instances contenues sur la scène de Flash constitue donc une hiérarchie qui peut être gérée sous la forme de nœuds XML avec une racine (la scène d'une animation) et des nœuds enfants (les objets d'affichage ou les conteneurs d'objets

Figure 2-1

Sur la scène de Flash, vous devez distinguer deux types d'occurrences. Celles qui ne contiennent pas d'autres occurrences et celles qui en contiennent.



d'affichage). Cela s'avère très pratique pour cibler une occurrence précise de la scène. La scène de Flash est alors le premier conteneur de la liste d'affichage.

Quelle différence doit-on faire entre les objets d'affichage et les conteneurs d'objets d'affichage ?

Celles et ceux qui ont utilisé les anciennes versions de Flash, se souviennent que les occurrences de type clip pouvaient en contenir d'autres (de même type ou de type différent). De ce fait, en ActionScript, il était possible de faire référence à une occurrence imbriquée dans une autre. En revanche, une occurrence de type graphique ne pouvait pas être contrôlée par ce biais ; il ne servait donc à rien d'y placer des occurrences de clips nommées.

Nous pourrions ainsi comparer les conteneurs d'objets d'affichage à des occurrences de type MovieClip et les objets d'affichage à des occurrences de symbole de type Graphique, à une différence près : un objet d'affichage ne peut pas contenir d'autres instances (quel que soit son type).

Plus généralement, un conteneur d'objets d'affichage est une occurrence (ou une instance) qui peut en contenir d'autres, alors qu'un objet d'affichage ne le peut pas. Dès que vous placez un symbole sur la scène ou créez une instance en ActionScript et l'ajoutez avec la méthode `addChild()` (ou un moyen équivalent), vous créez un objet d'affichage ou un conteneur d'objets d'affichage. Vous augmentez par la même occasion le nombre d'objets contenus dans la liste d'affichage.

Un objet d'affichage est toujours typé (au même titre qu'une occurrence de symbole est de type MovieClip, Bouton ou Graphique). Le schéma de la figure 2-2 présente les différents types d'objets (ou classes) disponibles dans une animation Flash. Précisons ici la notion d'héritage : un objet hérite des caractéristiques (propriétés, méthodes et événements) de l'objet qui se trouve immédiatement au-dessus de lui dans la hiérarchie.

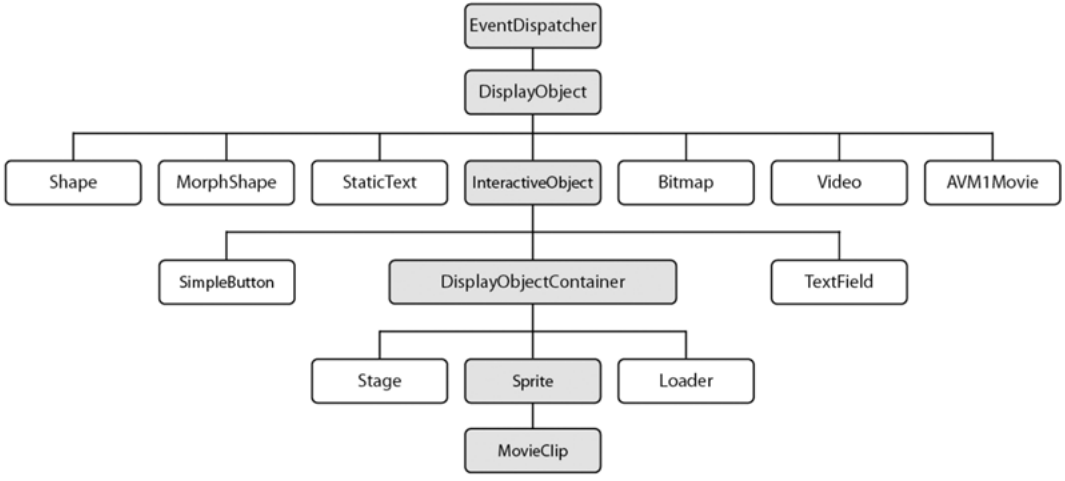


Figure 2-2

Cette arborescence vous présente l'ensemble des classes disponibles en AS3 et leurs héritages respectifs.

Les conteneurs d'objets d'affichage

Voici les différentes types de conteneurs d'objets d'affichage.

- Il est conseillé d'utiliser les instances de la classe `Loader()` pour charger des images sur la scène (ou également des SWF).
- Les instances de la classe `Sprite()` servent à définir les zones principales de l'interface de votre animation. D'une façon plus générale, la fonction d'une instance de `Sprite()` est de contenir d'autres instances.
- Les instances de la classe `MovieClip()` sont comparables à celles de la classe `Sprite()` à un détail près : elles possèdent un scénario (timeline), ce qui signifie qu'elles peuvent contenir des images-clés.

> **Rappel**
>
> Avant la création de la classe `Sprite()` (apparue dans Flash CS3), le MovieClip servait de conteneur d'affichage sans avoir pour autant les mêmes avantages que les instances de la classe `Sprite()`.

La section Structurer une mise en page avec les classes `Sprite()` ou `MovieClip()` permet de mieux comprendre comment organiser la construction d'une image.

Les objets d'affichage

Commençons par préciser que les objets d'affichage énumérés ci-dessous peuvent être obtenus par l'exécution d'instructions en ActionScript ou en utilisant les outils et les commandes proposés dans l'interface de Flash.

Lorsqu'un objet d'affichage est placé sur la scène, il ne peut pas servir de conteneur d'affichage. Prenons l'exemple d'un rectangle tracé sur la scène avec un outil de dessin. Vous pouvez définir ses propriétés (couleurs de fond et de contour, épaisseur du trait, etc.), mais pas lui ajouter une image ou un texte.

L'instruction « ajoute une image bitmap dans le rectangle que tu viens de dessiner sur la scène » ne signifierait d'ailleurs rien. En revanche, si nous précisions « ajoute une image bitmap dans le clip que tu viens de créer à partir du rectangle que tu avais préalablement dessiné », cela aurait plus de sens : vous comprendriez qu'il faut éditer le symbole pour lui ajouter une image bitmap. Le clip est alors, dans ce cas, un conteneur d'objets d'affichage.

Voici une liste des différents types d'objets d'affichage.

- Shape : permet de tracer dynamiquement des droites, des courbes et des formes géométriques sur la scène (à l'aide des outils de l'interface ou en faisant appel à des instructions en AS3).
- Bitmap : permet d'afficher une image bitmap sur la scène (en important une image à partir du menu Fichier>Importer ou à l'aide d'instructions en AS3).

- TextField : permet de créer un texte dynamique sur la scène (à l'aide de l'outil Texte, disponible dans la barre d'outils de l'interface, ou d'instructions en AS3).
- Video : permet d'afficher une vidéo sur la scène (à l'aide du symbole de type Video ou d'instructions en AS3).

Les instances des classes `MorphShape`, `StaticText` et `SimpleButton` ne peuvent pas être créées dynamiquement à partir d'instructions ActionScript, mais elles constituent tout de même des objets d'affichage.

Un TextField

Il s'agit d'une zone de texte dont le contenu peut être modifié dynamiquement au cours du déroulement de l'animation. Ce changement peut être réalisé par le programme ou directement via l'interface de l'animation par une saisie de l'utilisateur.

Vous aurez compris qu'il faut distinguer deux types d'objets d'affichage : ceux qui servent à regrouper plusieurs objets d'affichage en un seul (les conteneurs) et ceux qui ne servent à contenir qu'un seul élément (un texte, une image, une vidéo, une forme).

La difficulté relative aux objets d'affichage n'est pas de différencier ces deux types pour savoir quand utiliser l'un ou l'autre : ce choix découlera directement de vos besoins. Il s'agit plutôt d'être capable de définir judicieusement l'ensemble des conteneurs de votre animation.

Différence entre un objet d'affichage et une occurrence

En AS1/2, les notions d'objet d'affichage et de conteneur d'objets d'affichage n'existaient pas : nous parlions alors, à tort, uniquement d'occurrence et d'instance.

Généralement, le terme occurrence faisait référence à la représentation graphique d'un symbole sur la scène, alors qu'une instance résultait de l'instanciation d'une classe. Malheureusement, ce que peu de personnes savaient, c'est qu'un glisser-déplacer d'un symbole sur la scène revenait à instancier celui-ci. Nous pouvions donc parler d'instance pour évoquer une occurrence. Certains développeurs maîtrisant correctement la programmation orientée objet utilisaient et utilisent toujours ce terme. Aujourd'hui, avec la nouvelle méthode qui permet de placer un symbole sur la scène à partir de l'ActionScript, la notion d'instance prend tout son sens.

Remarque

Le terme occurrence se traduit en anglais par instance !

Partons du postulat qu'une occurrence et une instance désignent la même chose, la représentation d'un symbole ou un exemplaire d'une classe.

Lorsque, par exemple, nous plaçons un symbole de type Clip sur la scène, parle-t-on d'objet d'affichage (et de quel type) ou d'occurrence/instance ?

Il s'agit en fait des deux, ce que résume le script ci-dessous.

```
var monMessage = new TextField();
monMessage.text = "Bonjour";
addChild(monMessage);
monMessage.x=50;
monMessage.y=50;
```

Pour créer un texte dynamique ou de saisie sur la scène, nous créons une instance de la classe `TextField()`. Nous ajoutons ensuite cette instance à la liste d'affichage : elle fait donc partie de cette liste en tant qu'objet d'affichage. En définissant les propriétés `x` et `y` de l'instance `monMessage`, nous utilisons le fait que l'objet d'affichage est encore une instance.

Structurer une mise en page avec les classes Sprite() ou MovieClip()

Pour construire et gérer les différentes parties d'une animation, les utilisateurs de Flash, dont le profil est plutôt graphique, se servent généralement du scénario proposé dans l'interface du logiciel. Le déplacement de la tête de lecture fait alors apparaître les différents écrans de l'animation associés à des images-clés. Nous allons découvrir ici une autre technique, basée sur l'ActionScript, qui, bien qu'elle soit légèrement plus complexe et beaucoup plus abstraite, est plus efficace.

En effet, vous avez tout intérêt à gérer les changements d'affichage des différentes zones de votre animation en ajoutant et en supprimant, ou en affichant et en masquant, les objets d'affichage présents sur la scène. Cela sous-entend que l'analyse de l'interface de votre animation a permis de définir des zones principales et des zones secondaires.

Dans l'exemple ci-contre, vous pouvez constater que la mise en pages du site macgeneration est basée sur la grille que nous mettons en évidence dans la figure 2-4. Sans parler d'ergonomie, qui reste tout de même un objectif inéluctable lors de la construction d'une interface, vous noterez que la simplicité de lecture de cette page est due à une bonne structure.

Pour celles et ceux d'entre vous qui utilisent les CSS, vous comprendrez toute l'importance des développements suivants.

Avec l'arrivée de l'ActionScript 3, l'utilisation de la classe `Sprite()` nous permet d'organiser une mise en page sous la forme de blocs, c'est-à-dire des zones délimitant les différentes parties d'une animation. Par exemple, le découpage proposé dans la figure 2-4 (qui permet d'obtenir le résultat de la figure 2-3) contient des instances de la classe `Sprite()`.

Lorsque vous ferez appel à la classe `Sprite()` pour définir le point d'ancrage des différentes zones du quadrillage de votre mise en page, vous définirez ses coordonnées x et y.

Point d'ancrage d'une zone

Il s'agit du coin supérieur gauche d'un rectangle délimitant une zone.

Figure 2-3

La mise en forme de cette page est composée de zones.

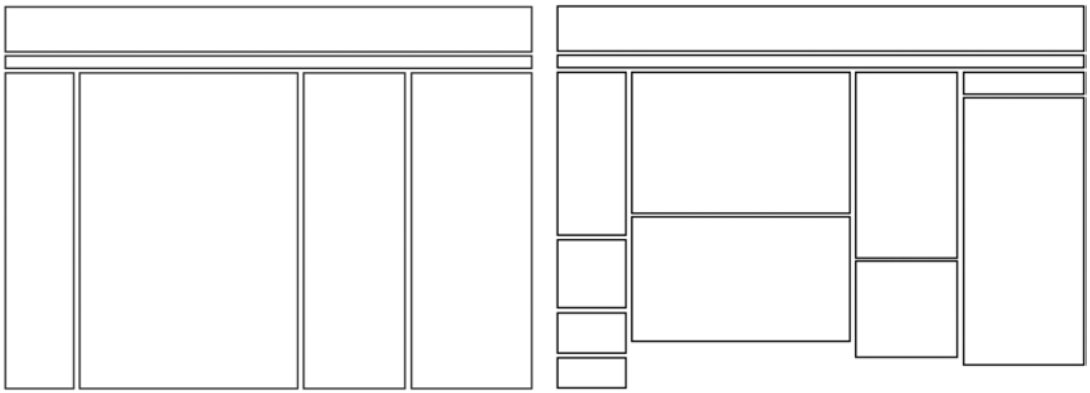


Figure 2-4

Le quadrillage de gauche met en évidence les zones principales du site macgeneration, à l'intérieur desquelles figurent des zones secondaires, présentées dans le quadrillage de droite.

La méthode addChild()

Cette méthode constitue l'action élémentaire pour placer dynamiquement un objet d'affichage ou un conteneur d'objets d'affichage sur la scène (ou dans une instance déjà présente sur la scène). En conséquence, l'objet ou le conteneur est ajouté à la liste d'affichage.

> **Note aux anciens développeurs en AS1/AS2**
>
> Les méthodes `createEmptyMovieClip()` et `attachMovie()` en AS1/AS2, ne peuvent plus être utilisées. Elles ont été partiellement remplacées par la méthode `addChild()`.

Lorsque vous aurez besoin de placer un symbole (avec liaison) de la bibliothèque sur la scène, vous utiliserez la méthode `addChild()`.

Celle-ci vous servira également à placer dynamiquement un texte sur la scène, après avoir créé une instance de la classe `TextField()`.

Elle sera encore utile à l'instanciation d'une classe personnelle, dans le but de construire un objet d'affichage sur la scène.

Nous pourrions continuer ainsi l'énumération des cas où vous emploieriez la méthode `addChild()` ; non seulement, elle ne serait pas exhaustive, mais cela ne servirait à rien. Vous comprendriez vite que cette méthode a toujours le même rôle : rendre visible sur la scène des instances préalablement créées.

Maintenant que nous savons définir les objets d'affichage et les conteneurs d'objets d'affichage et que nous connaissons leur rôle, nous devons apprendre à les gérer dans une animation et donc faire appel à la méthode `addChild()`.

Tout d'abord, distinguons les instances que vous placerez directement sur la scène de celles que vous ajouterez dans des instances (des conteneurs d'objets d'affichage) déjà présentes sur la scène. En considérant l'exemple de la figure 2-5, nous devons commencer par créer directement 6 instances de la classe `Sprite()` sur la scène. Nous pourrons ensuite y ajouter des objets d'affichages.

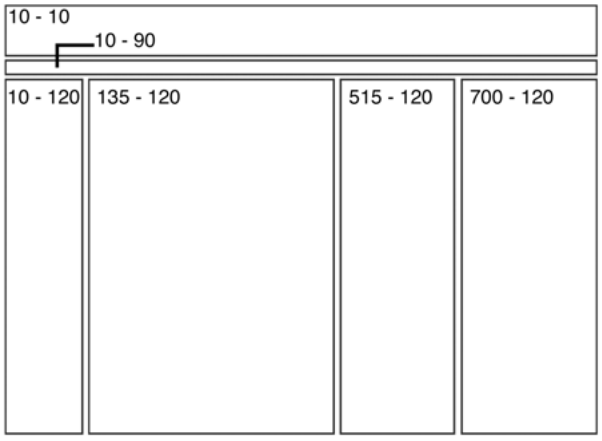


Figure 2-5

Les six zones de cette mise en page sont définies par des instances de la classe `Sprite()`.

Fichier de référence : `Chapitre2/addChildQuadrillage.fla`

```
var spEnTete:Sprite = new Sprite();
var spChapeau:Sprite = new Sprite();
var spMargeGauche:Sprite = new Sprite();
var spCorpsGauche:Sprite = new Sprite();
var spCorpsDroite:Sprite = new Sprite();
var spMargeDroite:Sprite = new Sprite();

addChild(spEnTete);
addChild(spChapeau);
addChild(spMargeGauche);
addChild(spCorpsGauche);
addChild(spCorpsDroite);
addChild(spMargeDroite);
```

Nous avons ajouté le préfixe `sp` devant chaque nom d'occurrence. Cela ne constitue en aucun cas une obligation, mais un simple repère visuel pour reconnaître qu'il s'agit d'une occurrence de la classe `Sprite()`.

Comme nous n'avons précisé aucune position, les instances sont placées par défaut en haut à gauche de la scène. Ajoutons les instructions suivantes pour obtenir la mise en page de la figure 2-5.

```
spEnTete.x =10;
spEnTete.y =10;

spChapeau.x =10;
spChapeau.y =90;

spMargeGauche.x =10;
spMargeGauche.y =120;

spCorpsGauche.x =135;
spCorpsGauche.y =120;

spCorpsDroite.x =515;
spCorpsDroite.y=120;

spMargeDroite.x =700;
spMargeDroite.y =120;
```

Les zones nommées dans notre interface étant définies, il est alors facile de programmer la suite du script. En nous basant sur la figure 2-4, nous allons ajouter deux zones (et non 4) pour pouvoir placer des textes.

```
var partieHaut:Sprite = new Sprite();
var partieBas:Sprite = new Sprite();
partieBas.y=100;

spMargeGauche.addChild(partieHaut);
spMargeGauche.addChild(partieBas);
```

Il est important de comprendre que la valeur 100, spécifiée à la troisième ligne du script, positionne l'instance à 220 pixels du haut de la scène, l'instance parent de l'instance `partieBas` se trouvant déjà à 120 pixels du haut de la scène.

Remarque

L'instruction `partieBas.y=100` peut être placée avant ou après la méthode `addChild()`.

Nous pouvons terminer cet exemple en ajoutant deux textes dans les instances `partieHaut` et `partieBas`.

```
var boutonAccueil:TextField = new TextField();
boutonAccueil.text="Accueil";
partieHaut.addChild(boutonAccueil);

var boutonContact:TextField = new TextField();
boutonContact.text="Contact";
partieBas.addChild(boutonContact);
```

Voici à présent une illustration du principal avantage de la liste d'affichage de l'AS3. Pour contrôler ou lire les propriétés de l'instance `boutonAccueil`, il est inutile de se référer à cette dernière en spécifiant un chemin à base de syntaxe pointée. Il suffit tout simplement d'écrire le nom de cette occurrence dans une instruction. Comme elle se trouve dans la liste d'affichage, sa position dans l'arborescence est parfaitement connue.

```
trace(boutonAccueil.text);
trace(boutonContact.text);
```

Le script ci-dessus provoque l'affichage des mots « Accueil » et « Contact » dans la fenêtre Sortie de l'interface de Flash.

Dans la figure 2-1, nous vous avions présenté un exemple d'arborescence de la liste d'affichage. Voici à présent la représentation de notre animation.

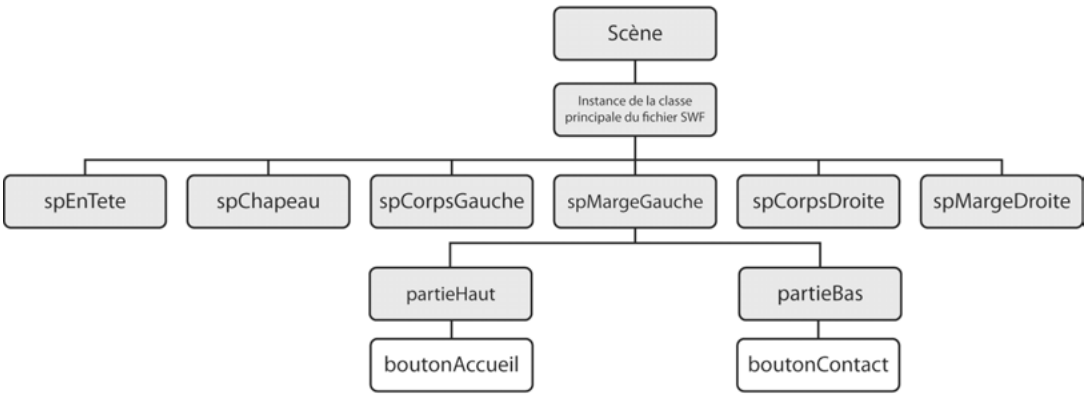


Figure 2-6

Arborescence de la liste d'affichage de notre animation.

Rappelons tout de même qu'une mise en page d'instances sur la scène de Flash n'est pas toujours basée sur des grilles comme celles que nous venons de présenter. Voici un deuxième exemple, l'interface d'un DVD-Rom, où nous avons créé des instances de la classe `Sprite()` pour définir des zones qui se trouvent à différents endroits sur la scène. La mise en page obtenue est présentée dans la figure 2-7.

Figure 2-7

Cette mise en page ne peut pas s'appuyer sur un quadrillage, mais nous utilisons tout de même des instances de la classe `Sprite()` pour définir des zones.

La structure correspondante est exposée dans la figure 2-8.

Dans l'exemple de la figure 2-8, vous observerez que nous avons tracé des zones sous la forme de rectangles blancs transparents. En réalité, la largeur et la hauteur de ces blocs n'ont pas été définies car nous n'avons spécifié que l'emplacement des instances de la classe `Sprite()`. Ces positions sont représentées par les carrés noirs de la figure 2-8. Vous noterez que le rectangle situé en bas à gauche de l'interface possède son point d'ancrage à droite. En effet, nous avons attaché un texte dynamique à cette zone, mais avec un alignement de texte à droite.

Figure 2-8

Les blocs de cette mise en page facilitent le développement, notamment pour ajouter ou supprimer un objet d'affichage.

Voici à présent un dernier exemple qui résume tout ce que nous venons de voir avec une approche plus simplifiée.

Fichier de référence : `Chapitre2/addChild.fla`

```
var marge:Sprite = new Sprite();
addChild(marge);
var corps:Sprite = new Sprite();
addChild(corps);
corps.x=100;

var texte1:TextField = new TextField ();
texte1.text="Zone de\rla marge";
corps.addChild(texte1);

var texte2:TextField = new TextField ();
texte2.text="Zone du corps\rde la page";
marge.addChild(texte2);
```

Contrôler l'ajout d'objets d'affichage avec l'événement ADDED

Lorsque vous ajouterez un objet d'affichage à la liste d'affichage, vous aurez la possibilité d'exécuter une ou plusieurs actions telles qu'un test, un comptage, le réglage d'une propriété, etc. La technique est extrêmement simple car il suffit d'utiliser l'événement `ADDED`.

Dans l'exemple ci-dessous, à chaque ajout d'un objet d'affichage dans l'instance `spMarge`, nous affichons, dans la fenêtre Sortie, le nombre d'enfants de cette instance.

```
var spMarge:Sprite = new Sprite();
addChild(spMarge);
spMarge.addEventListener(Event.ADDED,afficherInfo);

function afficherInfo(evt:Event) {
  trace(spMarge.numChildren);
}

var commande1:TextField = new TextField();
commande1.text="Accueil";
spMarge.addChild(commande1);


var commande2:TextField = new TextField();
commande2.text="Contact";
spMarge.addChild(commande2);
```

Cet événement s'avère très pratique pour mettre à jour certaines informations lors de chaque ajout d'un objet d'affichage à la liste d'affichage.

La propriété stage

Comme le précise très clairement le titre de cette section, la scène d'une animation est une propriété et non un objet directement accessible.

> **Remarque**
> Si le terme objet vous gêne et si vous avez déjà écrit des scripts en AS1/AS2, remplacez-le par le terme instance ou occurrence.

Avec l'AS1/AS2, il était possible de faire référence à la scène en tant qu'instance : elle représentait alors la racine d'un document. Rappelons que la classe `Stage()` était une classe de niveau supérieur. Lorsque nous écrivions le mot `Stage` (avec un S majuscule), nous faisions alors référence à la scène de l'animation Flash.

Pour afficher la largeur d'une occurrence dans la fenêtre Sortie de Flash nous devions écrire l'instruction suivante :

```
trace(nomDuneOccurrence._width);
```

La largeur de la scène pouvait être affichée à l'aide de l'instruction :

```
trace(Stage.width);
```

En ActionScript 3, toutes les instructions ci-dessous renvoient le même résultat, c'est-à-dire le chiffre 600 qui correspond à la largeur de la scène.

```
trace(stage.stageWidth);
trace(this.stage.stageWidth);
trace(rond.stage.stageWidth);
trace(carre.stage.stageWidth);
```

Pourquoi rencontre-t-on le mot stage en début de ligne dans le premier des quatre exemples ?

Aux 3e et 4e lignes de code, nous faisons référence à une instance particulière. Lorsque ce n'est pas le cas, comme à la première ligne, l'instance considérée est `this`.

Une animation désignée par `this` à la racine d'un script, possède ainsi une propriété `stage`. Toutes les occurrences placées sur la scène, que nous devons qualifier d'objet d'affichage (ou DisplayObject), possèdent également la propriété `stage` qui se réfère à la celle de l'animation.

En conséquence, si nous essayons d'afficher `this` à partir d'une image-clé de l'animation, nous obtenons :

```
trace(this);
[object MainTimeline]
```

Ce résultat peut se traduire par « objet de type timeline principale ».

Pour celles et ceux qui ont bien compris la notion de programmation objet, voici ce que renvoie le script ci-dessous, qui se trouve dans la classe du document d'une animation.

```
package {
  import flash.display.Sprite;
  public class main extends Sprite {
    function main() {
      trace(this);
      trace(this.stage);
      trace(this.stage.stageWidth);
    }
  }
}
[object main]
[object Stage]
600
```

Remarque

La classe `Stage()` était une classe de niveau supérieur en AS1/AS2, ce qui n'est plus le cas en AS3.

Classes de niveau supérieur en AS1/2 :

`Accessibility, Array, AsBroadcaster, Boolean, Button, Camera, Color, ContextMenu, ContextMenuItem, CustomActions, Date, Error, Function, Key, LoadVars, LocalConnection, Math, Microphone, Mouse, MovieClip, MovieClipLoader, NetConnection, NetStream, Number, Object, PrintJob, Selection, Selection, SharedObject, Sound, Stage, String, System, TextField, TextFormat, TextSnapshot, Video, XML, XMLNode, XMLSocket, XMLUI.`

Classes de niveau supérieur en AS3 :

`Array, Boolean, Date, decodeURI, decodeURIComponent, encodeURI, encodeURIComponent, escape, int, isFinite, isNaN, isXMLName, Number, Object, parseFloat, parseInt, String, trace, uint, unescape, XML.`

Supprimer un objet de la scène à l'aide de la méthode removeChild()

Remarque

Une occurrence qui a été obtenue sur la scène à partir d'un glisser-déplacer peut être supprimée à partir de la méthode `removeChild()` contrairement la méthode `removeMovieClip()` que nous utilisions en AS1/2.

Il est parfois nécessaire de supprimer un objet de la liste d'affichage.

Avant de vous proposer un exemple faisant appel à cette technique de suppression d'une instance sur la scène, il est important de préciser ou de rappeler que, dans certains cas, il sera plus judicieux de faire appel à la propriété `visible` que vous réglerez à `false` pour masquer temporairement une occurrence, plutôt que de la supprimer de la liste d'affichage.

La procédure de suppression d'un objet d'affichage de la liste d'affichage est extrêmement simple : elle se résume à l'exécution d'une seule instruction.

```
removeChild(cache1);
```

Remarque

L'occurrence ne sera plus visible sur la scène, mais elle conservera ses propriétés.

L'appel de cette méthode, sous-entend bien sûr l'exécution préalable des lignes d'instructions suivantes :

```
var cache1:Carte;
cache1 = new Carte();

addChild(cache1);
```

Que devient l'instance `cache1` après l'exécution de la méthode `removeChild()` ? En fait, elle existe toujours ; si vous demandez d'ajouter à nouveau cette instance à la liste d'affichage, elle est automatiquement replacée. Voici un exemple de synthèse des explications ci-dessus.

Fichier de référence : `Chapitre2/removeChild1.fla`

```
var cache1:Carte;
cache1 = new Carte();

addChild(cache1);

cache1.x = 250;
cache1.y =130;

btSuppression.addEventListener(MouseEvent.MOUSE_DOWN,retirerCache);

function retirerCache(evt:MouseEvent) {
  removeChild(cache1);
}
btReplacement.addEventListener(MouseEvent.MOUSE_DOWN,replacerCache);

function replacerCache(evt:MouseEvent) {
  addChild(cache1);
}
```

Si vous souhaitez supprimer l'objet d'affichage et plus précisément l'instance de la classe `Carte`, vous devez lui attribuer la valeur `null`, après l'avoir retirée de la liste d'affichage.

Contrôler la suppression d'objets d'affichage avec l'événement REMOVED_FROM_STAGE

Fichier de référence : `Chapitre2/removeChild4.fla`

Il est intéressant et important de savoir qu'à partir du moment où un objet d'affichage est supprimé de la liste d'affichage, l'événement `REMOVED_FROM_STAGE` est déclenché. Ainsi, dans l'exemple ci-dessous, cela permet de tenir une comptabilité des occurrences sur la scène.

```
var nbrCartes:Number = 3;
affichageNbrCartes.text= nbrCartes.toString();

bt1.addEventListener(Event.REMOVED_FROM_STAGE ,comptabiliserCarte);
bt2.addEventListener(Event.REMOVED_FROM_STAGE ,comptabiliserCarte);
bt3.addEventListener(Event.REMOVED_FROM_STAGE ,comptabiliserCarte);

bt1.addEventListener(MouseEvent.MOUSE_DOWN ,supprimerCarte);
bt2.addEventListener(MouseEvent.MOUSE_DOWN ,supprimerCarte);
bt3.addEventListener(MouseEvent.MOUSE_DOWN ,supprimerCarte);

function comptabiliserCarte(evt:Event) {
  nbrCartes--;
  affichageNbrCartes.text= nbrCartes.toString();
}

function supprimerCarte(evt:Event) {
  removeChild(DisplayObject(evt.currentTarget));
}
```

Remarque

Afin qu'il n'y ait aucune confusion, si vous souhaitez connaître le nombre d'objets d'affichage contenus dans un conteneur ou sur la scène, vous pouvez utiliser la propriété `numChildren`.

Ainsi, l'instruction ci-dessous (à placer sur une image-clé) permet de connaître le nombre d'objets d'affichage contenus sur la scène.

```
trace(this.numChildren);
```

Un tel gestionnaire d'événement facilite la gestion de la suppression de certaines catégories d'occurrences dans des jeux.

Nous allons découvrir, dans un deuxième exemple, que la méthode `removeChild()` doit être précédée du nom du conteneur d'objets d'affichage qui contient l'objet d'affichage que nous souhaitons supprimer. Au cours de ce chapitre, nous avons découvert qu'il est judicieux de regrouper certaines occurrences au sein d'un même conteneur. Ainsi, vous devez préfixer la méthode `addChild()` d'un nom de conteneur. Ce nom devra également être placé devant `removeChild()`.

Dans cet exemple, nous créons une instance de la classe `Sprite()` pour accueillir 3 objets d'affichage. Puis, nous programmons un bouton qui, lorsqu'on clique dessus, retire tous les objets d'affichage contenus dans l'instance `tableJeu`.

Fichier de référence : `Chapitre2/removeChild2.fla`

```
var tableJeu:Sprite;
tableJeu = new Sprite();

addChild(tableJeu);

var cache1:Carte;
var cache2:Carte;
var cache3:Carte;

cache1 = new Carte();
cache2 = new Carte();
cache3 = new Carte();

tableJeu.addChild(cache1);
tableJeu.addChild(cache2);
tableJeu.addChild(cache3);

cache1.x = illustrationTrain.x;
cache1.y =130;

cache2.x = illustrationBateau.x;
cache2.y = 130;

cache3.x = illustrationAvion.x;
cache3.y = 130;
```

```
btSuppression.addEventListener(MouseEvent.MOUSE_DOWN,retirerCache);


function retirerCache(evt:MouseEvent) {
  tableJeu.removeChild(cache1);
  tableJeu.removeChild(cache2);
  tableJeu.removeChild(cache3);
}
```

Nous allons découvrir, dans un troisième et dernier exemple, une mauvaise surprise. Pour mieux la comprendre examinons la signature de la méthode `removeChild()`.

Figure 2-9

La méthode removeChild() prend pour paramètre un nom de type DisplayObject, c'est-à-dire un objet d'affichage (ou celui d'un conteneur d'objets d'affichage).

Observez bien le type d'information contenu entre les parenthèses ! La méthode prend en paramètre le nom d'un objet d'affichage. Mais alors, lorsque nous programmons une fonction de rappel (CallBack) qui fait référence à la propriété `currentTarget` pour identifier un objet d'affichage à supprimer, comment préciser qu'il s'agit d'un objet d'affichage ?

Dans l'exemple ci-dessous, la ligne d'instruction ne peut pas être exécutée.

```
removeChild(evt.currentTarget);
```

Vous devez donc faire appel à la classe `DisplayObject()` pour préciser au compilateur que le paramètre `evt.currentTarget` est bien un objet d'affichage.

```
removeChild(DisplayObject(evt.currentTarget));
```

Voici un exemple complet qui illustre notre propos.

Fichier de référence : `Chapitre2/removeChild3.fla`

```
var cache1:Carte;
var cache2:Carte;
var cache3:Carte;

cache1 = new Carte();
cache2 = new Carte();
cache3 = new Carte();

addChild(cache1);
addChild(cache2);
addChild(cache3);

cache1.x = illustrationTrain.x;
```

```
cache1.y =130;

cache2.x = illustrationBateau.x;
cache2.y = 130;

cache3.x = illustrationAvion.x;
cache3.y = 130;

cache1.addEventListener(MouseEvent.MOUSE_DOWN,retirerCache);
cache2.addEventListener(MouseEvent.MOUSE_DOWN,retirerCache);
cache3.addEventListener(MouseEvent.MOUSE_DOWN,retirerCache);

function retirerCache(evt:MouseEvent) {
  removeChild(DisplayObject(evt.currentTarget));
}
```

removeChildAt()

Cette dernière méthode vous permettra de supprimer un objet d'affichage en spécifiant un numéro d'index. Afin de mieux comprendre le fonctionnement des index liés aux objets d'affichage, consultez le chapitre 4 de ce livre et plus particulièrement la section Gérer les plans entre deux ou plusieurs occurrences.

3

La gestion des événements

ATTENTION : note aux néophytes en programmation

Si vous débutez la lecture de ce livre directement à partir de ce chapitre, sans avoir lu intégralement les deux premiers, précisons simplement que les deux premières pages de ce chapitre risquent de vous paraître abstraites. Persévérez et n'abandonnez pas ; lisez le chapitre en entier pour avoir suffisamment de recul et comprendre ainsi qu'un gestionnaire d'événement permet de contrôler l'interactivité dans une animation Flash de différentes façons.

Pour rendre une animation interactive, il est indispensable de programmer une occurrence afin qu'elle réagisse aux clics et survols.

C'est le rôle d'un gestionnaire d'événement qui définit la relation qui existe entre une occurrence, un événement (comme un clic sur une occurrence) et une fonction contenant le code à exécuter lorsque l'événement survient.

Un événement ne se limite pas au clic sur une occurrence. Il en existe de nombreux autres que nous vous présenterons tout au long de ce chapitre tels que le survol (appelé aussi *rollover*), le double-clic, la pression sur une touche du clavier, etc.

En ActionScript 3, contrairement aux versions précédentes du langage, il n'existe qu'une seule méthode pour gérer l'interactivité d'une animation, mais elle est un peu plus complexe à mettre en place.

Tout au long de ce chapitre, nous allons faire référence, au travers des différents exemples abordés, à une occurrence intitulée `roue_inst`. Il s'agit de la roue de la figure 3.1.

Figure 3-1

Tout au long de ce chapitre, nous faisons référence au nom d'une occurrence intitulée roue_inst. Il s'agit de la roue crantée que vous voyez sur cette copie d'écran.

Fonctionnement des gestionnaires d'événements

Fichier de référence : `Chapitre3/evenement1.fla`

Que vous placiez votre gestionnaire d'événement dans un script qui se trouve sur une image-clé ou dans un fichier externe, la procédure est la même, seule la syntaxe diffère légèrement. Nous commencerons par un exemple de script placé sur une image-clé.

> **Différentes appellations**
>
> Un gestionnaire d'événement peut être également appelé un gestionnaire ou un écouteur. Les expressions suivantes signifient la même chose : créer, placer ou ajouter un gestionnaire ou un gestionnaire d'événement ; créer, placer, ajouter ou enregistrer un écouteur. Les écouteurs correspondent à la traduction du terme anglais *listeners* ; on emploie alors également l'expression créer, placer, ou ajouter un listener.

La déclaration de la fonction de rappel (callBack)

> **Préparation de votre animation**
>
> Afin de pouvoir mettre en pratique le script suivant, placez sur la scène d'une animation deux occurrences intitulées `roue_inst` et `btDeclencheur` ; leur apparence n'a pas d'importance. Enregistrez ensuite votre animation.

Tout d'abord, vous devez définir la fonction qui sera exécutée au moment du déclenchement de l'événement. Ce dernier sera défini et associé à une occurrence ultérieurement. La syntaxe à respecter lors de la déclaration d'une fonction est un peu particulière.

```
function tournerRoue (evt:MouseEvent) {
  roue_inst.rotation=5;
}
```

Comme vous pouvez le constater, la fonction possède (obligatoirement) une variable locale typée (`evt:MouseEvent`). L'oubli de ce paramètre entraîne une erreur lors de l'exécution du programme. Il s'agit du nom de l'objet qui va réceptionner toutes les informations renvoyées au moment de la détection du déclenchement de l'événement. D'ailleurs, si nous affichions le contenu de la variable `evt`, voici ce que nous obtiendrions :

```
trace(evt);
[MouseEvent type="mouseDown" bubbles=true cancelable=false eventPhase=2
localX=2 localY=3 stageX=102.5 stageY=142 relatedObject=null
ctrlKey=false altKey=false shiftKey=false delta=0]
```

Remarque

Dans cet exemple, nous avons choisi arbitrairement le terme evt, mais nous verrons dans ce chapitre que ce mot peut être choisi librement, sous certaines contraintes (pas de caractères accentués ou spéciaux, ni même d'espace, pas de capitale initiale).

Il s'agit d'informations relatives à l'occurrence cliquée ou survolée (tout dépend de l'événement défini). Ainsi, nous savons par exemple que l'occurrence se trouve à 102,5 pixels du bord gauche de la scène, que le clic s'est produit à 2 pixels (horizontalement) du point d'alignement de l'occurrence, que la touche Shift n'a pas été maintenue au moment du clic, etc. Toutes ces informations sont accessibles à partir de la simple lecture d'une propriété de l'objet `evt`, ce qui ne requiert pas de nombreuses lignes de code. Consultez la section Utiliser les informations stockées dans la variable locale de la fonction de rappel pour des explications complémentaires.

Remarque

Cette fonction est qualifiée de callBack car elle sera appelée (call) en retour (Back) de l'événement détecté. Les lignes de codes qu'elle contient seront alors exécutées.

La déclaration d'un écouteur

Une fois définie la fonction qui sera exécutée lors du déclenchement de l'événement, il faut l'associer à une occurrence et à un événement donné. C'est l'objet du code suivant :

```
btDeclencheur.addEventListener(MouseEvent.MOUSE_DOWN,tournerRoue);
```

Voici ce que signifie cette instruction qui fait appel à la méthode `addEventListener()` : `btDeclencheur` est le nom de l'occurrence sur laquelle nous allons devoir cliquer, événement `MouseEvent.MOUSE_DOWN`, pour exécuter la fonction de rappel `tournerRoue`.

Si vous étiez habitué aux gestionnaires d'événements de l'AS1/AS2, vous trouverez peut-être cette nouvelle syntaxe un peu plus complexe qu'un simple `onPress` ou `on(press)`,

mais vous vous y ferez vite. L'erreur que vous commettrez le plus souvent lors de vos premiers essais sera l'oubli du paramètre de la fonction de rappel (`evt` dans notre exemple). Soyez donc vigilant.

Remarque

Sur les exemples que vous pourrez trouver sur Internet, le nom de l'objet saisi en paramètre de la fonction de rappel est souvent `event`. Afin qu'il n'y ait pas d'ambiguïté, avec le type Event, nous avons volontairement utilisé un autre terme (evt comme abréviation d'event ou événement). Vous pouvez utiliser le mot de votre choix, à partir du moment où il ne contient pas de caractères accentués ou spéciaux, ni même d'espace, et ne débute pas par une majuscule.

Le script complet que vous devriez à présent obtenir est le suivant :

```
function tournerRoue(evt:MouseEvent) {
  roue_inst.rotation=5;
}

btDeclencheur.addEventListener(MouseEvent.MOUSE_DOWN,tournerRoue);
```

Le choix de l'événement

Revenons quelques instants sur l'événement passé en paramètres à la méthode `addEventListener()`. Lors de vos premiers essais en AS3, vous ne trouverez pas facilement l'événement approprié, d'une part parce que vous ne connaîtrez pas tous les événements disponibles, d'autre part car la syntaxe n'est pas évidente.

Pour vous aider, voici une liste des principaux événements que vous pouvez utiliser en AS3 :

- `MouseEvent.CLICK`
- `MouseEvent.DOUBLE_CLICK`
- `MouseEvent.MOUSE_DOWN`
- `MouseEvent.MOUSE_MOVE`
- `MouseEvent.MOUSE_OUT`
- `MouseEvent.MOUSE_OVER`
- `MouseEvent.MOUSE_UP`
- `MouseEvent.MOUSE_WHEEL`
- `MouseEvent.ROLL_OUT`
- `MouseEvent.ROLL_OVER`

Le double-clic

Afin que le double-clic soit reconnu dans votre animation, vous devez ajouter la ligne d'instruction ci-dessous à votre script.

```
btDeclencheur.doubleClickEnabled = true;
btDeclencheur.addEventListener(MouseEvent.DOUBLE_CLICK,tournerRoue);
```

Vous retrouverez dans les premières pages de l'annexe de ce livre, à la section intitulée Les packages et classes, la liste des événements couramment utilisés ainsi que tous les autres.

RollOverOutSide

Ne cherchez pas cet événement : il n'existe plus. Vous devez donc le programmer en initialisant une variable lors du clic sur une occurrence, puis en changeant sa valeur lorsqu'un `ROLL_OUT` est constaté, et enfin, en évaluant sa valeur sur le déclenchement de l'événement `MOUSE_UP`.

Trouver l'occurrence associée à l'événement

Vous aurez parfois besoin de concevoir une fonction susceptible de s'adapter à plusieurs occurrences.

Dans l'exemple ci-dessous, vous découvrirez que nous ne précisons pas de nom d'occurrence, contrairement au premier script de ce chapitre.

```
function tournerRoue(evt:MouseEvent) {
  evt.currentTarget.rotation=5;
}
roue_inst.addEventListener(MouseEvent.MOUSE_DOWN,tournerRoue);
```

Dès que l'utilisateur clique sur l'occurrence intitulée `roue_inst` (le nom d'instance spécifiée devant la fonction `addEventListener()`), elle tourne de 5 degrés.

Cette technique s'avère indispensable lorsque vous concevez des animations où la programmation dynamique joue un rôle important.

Script dans un fichier AS

Fichiers de référence : `Chapitre3/evenement1as.fla` et `evenement1.as`

Voyons à présent comment les gestionnaires d'événements s'inscrivent dans un fichier AS pour une approche complète de programmation orientée objet.

Attention

Si vous ne connaissez pas la programmation orientée objet, lisez la partie Les deux modes de programmation du chapitre 1.

```
package {

  import flash.display.Sprite;
  import flash.events.MouseEvent;
  import flash.display.MovieClip;

  public class evenement1 extends Sprite {

    public function evenement1() {
      btDeclencheur.addEventListener(MouseEvent.MOUSE_DOWN,tournerRoue);
    }
```

```
    private function tournerRoue(evt:MouseEvent) {
      roue_inst.rotation=5;
    }

  }

}
```

Vous devez commencer par importer la classe `Sprite()` car vous étendez la vôtre à partir de ce type. Vous ne devez pas non plus oublier d'importer la classe `MovieClip()`, car des instances de ce type sont présentes sur la scène.

Remarque

Une des erreurs les plus fréquentes lors de vos premiers essais en ActionScript 3 sera l'oubli de l'import de la classe `MouseEvent`. Ne tombez donc pas dans le piège.

Comme vous pouvez ensuite le constater, les fonctions intitulées `tournerRoue()` et `addEventListener()` sont utilisées comme en programmation séquentielle. La seule différence est l'attribut `private` de la fonction `tournerRoue()`.

Utiliser les informations stockées dans la variable locale de la fonction de rappel

Fichier de référence : `Chapitre3/evenement2.fla`

Remarque

Consultez le chapitre 9 dédié aux tests conditionnels si vous ne connaissez pas cette notion élémentaire d'algorithmique.

Même si vous ne comprenez pas intégralement l'exemple qui va suivre (car vous ne connaissez pas encore les tests conditionnels `if()` abordés au chapitre 9 de ce livre), voici un exemple où le code exécuté dans la fonction de rappel diffère selon le type d'événement déclenché.

```
function tournerRoue(evt:MouseEvent) {
  if (evt.type=="mouseDown") {
    roue_inst.rotation=5;
  } else {
    roue_inst.rotation=0;
  }
}

btDeclencheur.addEventListener(MouseEvent.MOUSE_DOWN,tournerRoue);
btDeclencheur.addEventListener(MouseEvent.MOUSE_UP,tournerRoue);
```

Nous utilisons la même fonction avec deux événements différents ! Grâce à la propriété `type` de l'objet `evt` (le paramètre de la fonction de rappel), nous déterminons l'événement

survenu. Nous pouvons alors effectuer un test pour exécuter une ligne d'instruction ou une autre en fonction du type d'événement.

Dans cet autre exemple, nous allons vérifier si la touche Shift est enfoncée au moment du clic. En fonction de la valeur renvoyée par le test (`true` ou `false`), nous allons effectuer une rotation de l'occurrence dans le sens des aiguilles d'une montre ou dans le sens anti-horaire.

Fichier de référence : `Chapitre3/evenement5.fla`

```
function tournerRoue(evt:MouseEvent) {
  if ( evt.shiftKey) {
    roue_inst.rotation-=3;
  } else {
    roue_inst.rotation+=3;
  }
}

roue_inst.addEventListener(MouseEvent.CLICK, tournerRoue);
```

Quelques mots supplémentaires sur les gestionnaires d'événements

Les exemples que nous venons d'aborder se limitent à des événements liés à l'activité de la souris, mais vous devez garder à l'esprit qu'il en existe de nombreux autres. Notre objectif premier était de vous faire découvrir la syntaxe et le fonctionnement d'un gestionnaire d'événement en ActionScript 3. Que vous téléchargiez un fichier XML ou que vous surveilliez la fin de la lecture d'un son, un gestionnaire d'événement possédera toujours la même syntaxe.

Détecter un clic

Détecter un simple clic sur une occurrence

Il existe 3 événements pour détecter le clic sur une instance. Ils ont chacun leur particularité.

- `MOUSE_DOWN` : événement détecté au moment où l'utilisateur enfonce le bouton de la souris.
- `MOUSE_UP` : événement détecté au moment où l'utilisateur relâche le bouton de la souris.
- `CLICK` : événement détecté au moment où l'utilisateur relâche le bouton de la souris.

Note

Il n'existe pas de différence entre les constantes `CLICK` et `MOUSE_UP`, elles sont toutes les deux compatibles avec `MOUSE_DOWN`. Dans le cas où vous feriez appel aux trois, voici leur ordre d'invocation : `MOUSE_DOWN`, `MOUSE_UP`, `CLICK`.

Nous vous rappelons, dans cet exemple, comment détecter un clic sur une occurrence :

```
function tournerRoue (evt:MouseEvent) {
  trace("Vous venez de cliquer sur l'occurrence intitulée btDeclencheur");
}
btDeclencheur.addEventListener(MouseEvent.CLICK,tournerRoue);
```

Consultez l'exemple proposé dans la section Utiliser les informations stockées dans la variable locale de la fonction de rappel, un peu avant dans ce chapitre, pour découvrir un exemple qui met en évidence la subtile différence entre un `MOUSE_DOWN` et un `MOUSE_UP`.

Détecter un double-clic sur une occurrence

Fichier de référence : `Chapitre3/evenement3.fla`

Le script est le même que dans le cas d'un simple clic sur une occurrence, mais l'utilisation d'un nouvel événement n'est pas le seul changement. Vous devez ajouter la propriété `doubleClickEnabled` et la régler sur `true`. Dans le cas contraire, le double-clic ne sera pas reconnu, mais il sera interprété comme des événements `MOUSE_DOWN` et `MOUSE_UP`.

```
function changerTaille (evt:MouseEvent) {
  roue_inst.scaleX=roue_inst.scaleY=Math.random();
}

roue_inst.addEventListener(MouseEvent.DOUBLE_CLICK,changerTaille);
roue_inst.doubleClickEnabled = true;
```

Détecter le clic sur la scène

Fichier de référence : `Chapitre3/evenement4.fla`

Dans certains cas, vous aurez besoin de détecter le clic effectué en dehors de toute occurrence, c'est-à-dire sur la scène. Vous pourrez utiliser les propriétés évoquées jusqu'à présent (`MOUSE_UP`, `MOUSE_DOWN`, `CLICK`), mais la cible à laquelle est rattachée la méthode `addEventListener()` est la scène qui se traduit par le terme `stage`. Dans l'exemple ci-dessous, nous souhaitons connaître les coordonnées X et Y de l'emplacement du clic afin, par exemple, d'y placer dynamiquement un symbole.

```
function indiquerCoordonneesSouris(evt:MouseEvent) {
  trace(mouseX+" - "+mouseY);
}

stage.addEventListener(MouseEvent.MOUSE_DOWN,indiquerCoordonneesSouris);
```

ou encore,

```
function indiquerCoordonneesSouris(evt:MouseEvent) {
  roue_inst.x=mouseX;
  roue_inst.y=mouseY;
}

stage.addEventListener(MouseEvent.MOUSE_DOWN,indiquerCoordonneesSouris);
```

Détecter la pression sur une touche du clavier

Fichier de référence : `Chapitre3/evenement6.fla`

Vous allez être surpris par la ressemblance du code nécessaire à la détection de la pression sur une touche du clavier avec celui que nous avons présenté pour la détection des clics de souris.

```
function tournerRoue(evt:KeyboardEvent) {
  roue_inst.rotation+=3;
}

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

Le type figurant en paramètre de la fonction `tournerRoue()` change pour devenir `KeyboardEvent` et, de ce fait, la constante `KEY_DOWN` vient remplacer celles que nous avions utilisées (`MOUSE_UP`, `CLICK`, etc.). De plus, la fonction `addEventListener()` est associée à l'instance `stage` et non plus à `btDeclencheur` ou `roue_inst`.

Pour l'instant, l'événement `KeybordEvent` traduit la pression d'une touche du clavier, mais ne permet pas encore de déterminer le code de la touche pressée.

C'est justement le cas dans l'exemple suivant, où la roue tourne dans le horaire ou anti-horaire selon que l'on appuie la touche flèche droite ou flèche gauche du clavier.

Fichier de référence : `Chapitre3/evenement7.fla`

```
function avancer(evt:KeyboardEvent) {
  if(evt.keyCode==37) carre.rotation = carre.rotation-3;
  if(evt.keyCode==39) carre.rotation = carre.rotation+3;
}

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

Script dans un fichier AS

Fichiers de référence : `Chapitre3/evenement7as.fla` et `mainclavier.as`

Le script ci-dessous se trouve dans le fichier texte nommé `mainclavier.as`. Il s'agit de la classe du document `evenement7as.fla`.

```
package {

  import flash.display.Sprite;
  import flash.events.KeyboardEvent;
  import flash.display.MovieClip;

  public class mainclavier extends Sprite {

    public function mainclavier() {
      stage.addEventListener(KeyboardEvent.KEY_DOWN,avancer);
    }

    private function avancer(evt:KeyboardEvent) {
```

```
      if (evt.keyCode==37) {
        roue_inst.rotation = roue_inst.rotation-3;
      }
      if (evt.keyCode==39) {
        roue_inst.rotation = roue_inst.rotation+3;
      }
    }
  }
}
```

Vous remarquerez que le code est strictement identique et qu'une fois de plus, seule la syntaxe propre au développement orienté objet diffère. Vous noterez que la classe `MouseEvent` n'est pas importée, mais que `KeyboardEvent` l'est.

Surveiller la saisie de l'utilisateur

> **Remarque**
> Le chapitre 14 de ce livre traite du même sujet et aborde deux exemples supplémentaires dans la partie Gérer les événements liés au texte. Consultez-le pour de plus amples renseignements sur cette problématique.

Fichier de référence : `Chapitre3/evenement8Bis.fla`

Lorsque vous positionnerez des champs texte de saisie sur la scène d'une animation, vous aurez parfois besoin de détecter les événements suivants :

- L'utilisateur clique dans la zone du champ texte de saisie.
- L'utilisateur ajoute ou supprime un caractère (donc il saisit un texte ou le modifie).
- L'utilisateur clique en dehors du champ texte de saisie.

Vous devrez alors faire appel à 3 types de gestionnaires d'événements :

- `FocusEvent.FOCUS_IN`
- `FocusEvent.FOCUS_OUT`
- `Event.CHANGE` ou `TextEvent.TEXT_INPUT`

Voici l'exemple d'une animation où l'utilisateur est invité à saisir un texte dans une zone de l'interface. Dès qu'il va cliquer et, de ce fait, rendre actif le champ texte de saisie, le message initial `Tapez votre texte ici…` va disparaître. À chaque fois que l'utilisateur ajoutera ou supprimera un caractère, un compteur affichera le nombre de signes contenus dans le champ texte de saisie. Enfin, si l'utilisateur clique en dehors du champ texte de saisie, un message final indique le nombre total de caractères frappés.

```
function initialiserSaisie(evt:FocusEvent) {
  if (zoneSaisie.text=="Tapez votre texte ici…") {
    zoneSaisie.text="";
    commentaire.text="Mot en cours de saisie";
  }
```

```
}
function terminerSaisie(evt:FocusEvent) {
  commentaire.text="Vous avez tapé "+nbrCaracteres.text+" caractères.";
  nbrCaracteres.text="";
}
function touchePressee(evt:TextEvent) {
  nbrCaracteres.text = zoneSaisie.text.length.toString();
}
zoneSaisie.addEventListener(FocusEvent.FOCUS_IN,initialiserSaisie);
zoneSaisie.addEventListener(TextEvent.TEXT_INPUT,touchePressee);
zoneSaisie.addEventListener(FocusEvent.FOCUS_OUT,terminerSaisie);
```

Script dans un fichier AS

Fichiers de référence : `Chapitre3/evenement8as.fla` et `mainclavier2.as`

Remarque

Ces fichiers correspondent à l'animation evenement8.fla et non evenement8Bis.fla proposée dans l'exemple précédent.

```
package {

  import flash.display.Sprite;
  import flash.text.TextField;
  import flash.events.TextEvent;
  import flash.events.FocusEvent;

  public class mainclavier2 extends Sprite {

    function mainclavier2() {
      zoneSaisie.addEventListener(FocusEvent.FOCUS_IN,initialiserSaisie);
      zoneSaisie.addEventListener(TextEvent.TEXT_INPUT,touchePressee);
      zoneSaisie.addEventListener(FocusEvent.FOCUS_OUT,terminerSaisie);
    }
    private function initialiserSaisie(evt:FocusEvent) {
      if (zoneSaisie.text=="Tapez votre texte ici…") {
        zoneSaisie.text="";
        commentaire.text="";
      }
    }

    private function terminerSaisie(evt:FocusEvent) {
      commentaire.text="Vous avez tapé "+nbrCaracteres.text+" caractères.";
    }

    private function touchePressee(evt:TextEvent) {
      nbrCaracteres.text = zoneSaisie.text.length.toString();
    }
  }
}
```

La temporisation d'une action avec l'événement ENTER_FRAME

Fichier de référence : `Chapitre3/evenement9.fla`

Jusqu'à présent, tous les événements que nous avons abordés sont déclenchés par l'intervention de l'utilisateur qui clique sur le bouton de la souris ou qui presse une touche du clavier. Toutes ces actions font donc suite à une action manuelle. Nous aimerions à présent que des instructions soient exécutées non seulement sans que l'utilisateur ait à cliquer sur la moindre occurrence présente sur la scène, mais également en continu.

Dans l'exemple suivant, nous allons faire tourner une roue crantée en exécutant en continu l'instruction ci-dessous :

```
roue_inst.rotation+=3;
```

Pour celles et ceux qui connaissent l'ActionScript 1 et/ou 2, vous vous souvenez peut-être du fameux `onEnterFrame` ! Nous allons faire appel au même événement, mais à travers la structure du gestionnaire que nous commençons à bien connaître à présent :

```
function tournerRoue(evt:Event) {
  roue_inst.rotation+=3;
}

stage.addEventListener(Event.ENTER_FRAME, tournerRoue);
```

En AS1/2, nous déterminions une occurrence pour y associer l'événement `onEnterFrame`. Dans notre exemple, vous constaterez que nous avons fait appel à la propriété `stage`, mais que nous aurions tout autant pu utiliser le nom d'instance `roue_inst`.

remove/addEventListener()

La notion que nous allons présenter ici n'a pas été abordée dans les exemples précédents : cela n'était pas nécessaire. En effet, lorsqu'un clic est effectué sur une occurrence ou lorsqu'une touche du clavier est pressée, les lignes d'instructions contenues dans la fonction de rappel ne sont exécutées qu'une seule fois. En revanche, dans le cas de l'événement `ENTER_FRAME`, il est indispensable de contrôler l'arrêt de l'exécution continue. Nous devons alors procéder à l'annulation de l'écouteur d'événement comme dans l'exemple ci-dessous :

```
roue_inst.addEventListener(MouseEvent.CLICK,arreterRoue);
function arreterRoue(evt:Event) {
  roue_inst.removeEventListener(Event.ENTER_FRAME, tournerRoue);
}
```

Dans cet exemple, l'utilisateur doit cliquer sur l'occurrence concernée par le mouvement pour arrêter la rotation. Bien entendu, nous pourrions choisir une autre instance.

Fonctionnalités associées à l'événement ENTER_FRAME

Vous découvrirez dans le chapitre 9 de ce livre qu'un test dans un programme peut être réalisé à différents moments de l'exécution d'un script ou d'une animation. Dans le cas

où vous auriez besoin d'effectuer une vérification en continu, l'utilisation de l'événement ENTER_FRAME constitue une solution intéressante.

Par ailleurs, nous vous invitons à consulter le chapitre 5 pour découvrir les mouvements obtenus à l'aide de l'événement ENTER_FRAME.

Script dans un fichier AS

Fichiers de référence : Chapitre3/evenement9as.fla et maintourner.as

```
package {

  import flash.display.Sprite;
  import flash.events.Event;
  import flash.display.MovieClip;

  public class maintourner extends Sprite {

    function maintourner() {
      roue_inst.addEventListener(Event.ENTER_FRAME, tournerRoue);
    }

    private function tournerRoue(evt:Event) {
      roue_inst.rotation+=3;
    }
  }
}
```

N'oubliez pas d'importer la classe MovieClip, car même si vous n'y faites pas directement référence dans le script, vous utilisez un symbole de type MovieClip sur la scène.

La temporisation d'une action avec la classe Timer()

Fichier de référence : Chapitre3/evenement10.fla

Comme nous l'avons présenté dans l'exemple précédent, la programmation d'une exécution en continu est très simple. Cependant, nous ne pouvons pas préciser le délai qui sépare deux répétitions.

La classe Timer() permet justement de préciser ce paramètre, mais également le nombre de répétitions ; il sera donc plus judicieux, dans certains cas, de faire appel à elle.

Commençons par instancier la classe Timer(), puis à faire appel à la fonction AddEventListener() pour associer un événement à une fonction de retour.

```
var moteur:Timer = new Timer(20,10);
moteur.addEventListener(TimerEvent.TIMER,tournerRoue);
```

Le premier paramètre passé à la fonction `Timer()` (20 dans notre exemple) correspond à la vitesse d'exécution. En choisissant la valeur 1 000, le délai entre deux répétitions serait de 1 seconde (soit 1 000 millisecondes).

Le deuxième paramètre (10 dans notre exemple) permet de préciser le nombre de répétitions. Si nous avions souhaité qu'il ne soit pas limité, nous aurions précisé la valeur 0.

À ce niveau là du script, nous devons préciser le contenu de la fonction de rappel.

```
function tournerRoue(evt:Event) {
  roue_inst.rotation+=3;
}
```

Il ne reste plus qu'à enclencher le timer.

```
moteur.start();
```

Nous avons volontairement choisi le nom d'instance `moteur`, afin de montrer l'analogie entre le fonctionnement de la classe `Timer()` et celui d'une horloge, d'une mécanique, d'un moteur…

Pour arrêter un timer, il suffit tout simplement d'utiliser le gestionnaire d'événement ci-dessous :

```
roue_inst.addEventListener(MouseEvent.CLICK,arreterRotation);
function arreterRotation (evt:MouseEvent) {
  moteur.stop();
}
```

Nous devons alors cliquer sur la roue pour arrêter le timer. Lorsque ce dernier se termine, nous pouvons prévoir une action à exécuter :

```
function finBoucle(evt:TimerEvent) {
  roue_inst.scaleX=0.8;
  roue_inst.scaleY=0.8;
}
moteur.addEventListener(TimerEvent.TIMER_COMPLETE,finBoucle);
```

L'occurrence `roue_inst` subit un changement d'échelle de 80 % au bout de 10 exécutions de la fonction de rappel.

Voici le script complet de l'animation `evenement10.fla` :

```
var moteur:Timer = new Timer(10,50);

moteur.addEventListener(TimerEvent.TIMER,tournerRoue);
moteur.start();

function tournerRoue(evt: TimerEvent) {
  roue_inst.rotation+=3;
}

function arreterRotation(evt:MouseEvent) {
  moteur.stop();
}
```

```
roue_inst.addEventListener(MouseEvent.CLICK,arreterRotation);


function finBoucle(evt:TimerEvent) {
  roue_inst.scaleX=0.8;
  roue_inst.scaleY=0.8;
}
moteur.addEventListener(TimerEvent.TIMER_COMPLETE,finBoucle);
```

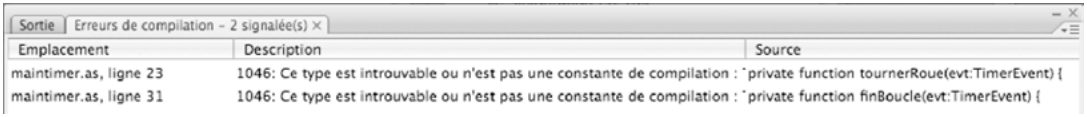
Comme nous l'évoquions pour l'événement `ENTER_FRAME`, nous vous invitons à consulter le chapitre 5 pour découvrir les mouvements obtenus à l'aide de la classe `Timer()`.

Script dans un fichier AS

Fichiers de référence : `Chapitre3/evenement10as.fla` et `maintimer.as`

```
package {

  import flash.display.Sprite;
  import flash.display.MovieClip;
  import flash.events.TimerEvent;
  import flash.events.MouseEvent;
  import flash.utils.Timer;

  public class maintimer extends Sprite {

    public static  var moteur:Timer;
    public static  var roue_inst:MovieClip;

    public function maintimer() {

      moteur=new Timer(10,50);
      moteur.addEventListener(TimerEvent.TIMER,tournerRoue);
      roue_inst.addEventListener(MouseEvent.CLICK,arreterRotation);
      moteur.addEventListener(TimerEvent.TIMER_COMPLETE,finBoucle);
      moteur.start();
    }
    private function tournerRoue(evt:TimerEvent) {
      roue_inst.rotation+= 3;
    }

    private function arreterRotation(evt:MouseEvent) {
      moteur.stop();
    }

    private function finBoucle(evt:TimerEvent) {
      roue_inst.scaleX=0.8;
      roue_inst.scaleY=0.8;
    }


  }

}
```

Puisque nous faisons appel à deux nouvelles classes, nous devons les importer. Il s'agit de `flash.events.TimerEvent` et `flash.utils.Timer`. Voici les messages que vous obtiendriez dans le cas contraire.



Le message d'erreur est très clair : Ce type est introuvable ou n'est pas une constante de compilation. Aux lignes 23 et 31, nous faisons référence au type `TimerEvent` alors que nous n'avons pas importé la classe !

Conclusion

En dehors de la classe `Timer()`, nous avons pu constater que tous les gestionnaires d'événements présentent la même syntaxe. La difficulté relative à cette gestion ne relève donc pas de la structure du code à saisir, mais plutôt de la mémorisation de tous les types d'événements. Voici un tableau récapitulatif de tous les scripts que nous avons abordés. Consultez chaque partie de ce chapitre pour des explications complémentaires.

Tableau 3-1 : synthèse des différents événements gérant l'interactivité

Ligne d'instruction	Instant
`addEventListener(MouseEvent.MOUSE_DOWN,tournerRoue);`	Clic sur une occurrence
`addEventListener(MouseEvent.DOUBLE_CLICK,tournerRoue);`	Double-clic sur une occurrence
`addEventListener(KeyboardEvent.KEY_DOWN, tournerRoue);`	Touche du clavier enfoncée
`addEventListener(FocusEvent.FOCUS_IN,initialiserSaisie);`	Clic dans un champ texte de saisie
`addEventListener(TextEvent.TEXT_INPUT,touchePressee);`	Saisie ou modification d'un texte (dans un champ texte de saisie)
`addEventListener(Event.ENTER_FRAME, tournerRoue);`	Exécution en continu du code contenu dans la fonction de rappel.

ATTENTION : note aux néophytes en matière de programmation

Si ce chapitre ne vous a pas semblé trop compliqué et que vous n'avez pas encore lu la partie consacrée à la programmation objet dans le chapitre 1 et le deuxième chapitre, revenez-y pour poursuivre votre découverte de la programmation orientée objet et le fonctionnement de la liste d'affichage. Dans le cas contraire, lisez le quatrième chapitre et revenez sur la présentation de la programmation objet du chapitre 1 lorsque vous aurez terminé de lire la partie I de ce livre.

4

Contrôler une occurrence et naviguer sur la scène

Ce chapitre, en plus d'être très abordable, est absolument indispensable pour être capable d'écrire ses premiers scripts.

Nous allons développer les notions de base de contrôle d'une occurrence, notions que nous avons été amenés à utiliser dans les chapitres précédents sans véritablement les expliquer.

Les propriétés d'une occurrence

Comme les gestionnaires d'événements, qui ont fait l'objet du chapitre précédent, les propriétés d'une occurrence sont incontournables. Elles font partie des différents réglages élémentaires que vous devez connaître et savoir appliquer à une occurrence sur la scène.

- Les propriétés `x` et `y` servent à contrôler la position de l'occurrence ; sa rotation peut être réglée par la propriété `rotation`.
- La propriété `visible` sert à masquer ou à afficher l'occurrence.
- Le réglage de la transparence de l'occurrence est possible à l'aide de la propriété `alpha`.
- Les propriétés `scaleX` et `scaleY` sont utilisées pour fixer l'échelle de l'occurrence.
- Enfin, les dimensions de l'occurrence sont déterminées par les propriétés `width` et `height`.

Le tableau 4-1 présente les unités, les limites des valeurs à utiliser et les spécificités de chaque propriété.

Tableau 4-1 Unités, valeurs limites et spécificités de chaque propriété

Propriétés	Fonction sur l'occurrence	Unité	Limites	Spécificités
`x` et `y`	Positionner horizontalement et verticalement une occurrence sur la scène. Le point 0 en x et 0 en y correspond au coin supérieur gauche de la scène.	Pixels	Pas de limite	Des valeurs négatives ou supérieures aux dimensions de la scène permettent de masquer une occurrence.
`rotation`	Règle la rotation d'une occurrence selon un ange précis. Cette propriété ne génère pas une animation.	Degrés	Pas de limite	Si vous cherchez à lire l'angle de rotation d'une occurrence, vous obtiendrez toujours une valeur comprise entre –180 et 180, alors que vous pouvez utiliser n'importe quelle valeur pour le réglage.
visible	Afficher/Masquer une occurrence	Booléen	true ou false	L'utilisation de l'opérateur ! permet de changer la valeur de la variable.
`alpha`	Régler la transparence d'une occurrence	Pourcentage	Entre 0 et 1. Exemple : 0,5 pour 50 %.	Aucune
`scaleX` et `scaleY`	Régler l'échelle horizontale et verticale d'une occurrence	Pourcentage	Entre 0 et 1 théoriquement. Exemple : 0.5 pour 50%, mais on peut aussi aller au delà de 1 dans la pratique (comme 2 pour 200 %).	L'utilisation de valeurs négatives permet d'obtenir une symétrie de l'occurrence.
`width` et `height`	Régler la largeur et la hauteur d'une occurrence.	Pixels	Entre 0 et l'infini.	Aucune

Remarque

Dans le cas d'une mise à l'échelle d'une occurrence ou d'un changement de taille à l'aide des propriétés `width` et `height`, optez pour une épaisseur de trait de type Filet dans la palette Propriétés afin de ne pas changer l'apparence du trait de contour de l'occurrence.

Il existe plusieurs syntaxes pour fixer la valeur d'une propriété.

Le réglage d'une simple valeur s'écrit : `roue.rotation = 12;`

Vous noterez que le code à saisir est des plus simples, car il vous suffit de faire référence au nom de l'occurrence suivie d'un point, puis de saisir le nom de la propriété, et enfin de terminer par affecter une valeur à l'aide de l'opérateur `=`. Dans ce premier exemple, le nombre 12 est une valeur numérique ; mais vous pouvez utiliser d'autres types de valeurs :

- Attribuer le résultat d'un calcul : `roue.rotation = 13*17;`
- Attribuer la valeur d'une variable : `roue.rotation = orientationInitialeRoue;`
- Attribuer la valeur de la propriété d'une autre occurrence (avec ou sans calcul supplémentaire) : `roue.rotation = roue2.rotation+5;`

- Régler la valeur de plusieurs propriétés : `roue.rotation = roue2.rotation = 5;`
- Attribuer la valeur renvoyée par une fonction : `roue.rotation = Number(positionRoue2);`

Analysons à présent quelques exemples.

Fichier de référence : `Chapitre4/proprietes1.fla`

```
roue.x = 150;
roue.y = 90;
roue2.addEventListener(MouseEvent.MOUSE_DOWN,placerRoue2);
function placerRoue2(evt:Event) {
  roue2.x=259;
  roue2.y=103;
}
```

- Lignes 1 et 2 : réglage de la position de l'occurrence `roue` à 150 pixels du bord gauche de la scène et 90 pixels du haut.
- Ligne 3 : attribution d'un écouteur à l'occurrence `roue2`.
- Lignes 4 à 7 : fonction de rappel appelée par l'écouteur de la ligne 3.
- Lignes 5 et 6 : réglage de la position de l'occurrence `roue2` à 259 pixels du bord gauche de la scène et 103 pixels du haut.

Ce premier script, très court, est extrêmement basique : il permet d'initialiser la position d'une occurrence et de définir un écouteur pour gérer le clic sur une autre (ou sur elle-même). Il résume assez bien le cœur des scripts que vous aurez à rédiger pour contrôler une occurrence (initialisation et contrôle par un événement souris).

Fichier de référence : `Chapitre4/proprietes1.fla`

```
roue.addEventListener(Event.ENTER_FRAME,tourner);
function tourner(evt:Event) {
  roue.rotation = roue.rotation+3;
  roue2.rotation-=3;
}
```

- Ligne 1 : attribution d'un écouteur à l'occurrence `roue`.
- Ligne 2 : fonction de rappel appelée par l'écouteur de la ligne 1.
- Ligne 3 et 4 : augmentation et diminution des propriétés des occurrences `roue` et `roue2` de respectivement +3 et –3 pixels.

Le choix de ce deuxième script n'est pas dû au hasard : il permet de démontrer la méthode d'incrémentation en continu de la valeur d'une propriété, générant ainsi un mouvement.

Précisons que la gestion l'événement `ENTER_FRAME` peut bien souvent être remplacée par une instance de l'objet `TIMER()`.

Les différences d'écriture des deux lignes d'instructions 3 et 4 montrent qu'on peut utiliser deux syntaxes. Nous aurions aussi bien pu écrire…

```
  roue.rotation = roue.rotation+3;
  roue2.rotation = roue2.rotation-3;
```

que…

```
  roue.rotation +=3;
  roue2.rotation-=3;
```

Fichier de référence : `Chapitre4/proprietes1.fla`

```
roue.x = 150;
roue.y = 90;
roue.alpha = roue2.alpha = 0.5;

roue2.addEventListener(MouseEvent.MOUSE_DOWN,placerRoue2);
roue.addEventListener(Event.ENTER_FRAME,tournerRoue);

function placerRoue2(evt:Event) {
  roue2.x=259;
  roue2.y=103;
  axe.rotation=-45;
  roue2.rotation=roue.rotation=0;
  roue2.addEventListener(Event.ENTER_FRAME,tournerRoue2);
}

function tournerRoue(evt:Event) {
  roue.rotation +=3;
}

function tournerRoue2(evt:Event) {
  roue2.rotation -=3;
}
```

Nous ne détaillerons pas ligne à ligne ce script car nous ne ferions que répéter les analyses présentées ci-dessus. Précisons simplement que nous avons cherché à combiner les deux premières animations pour démontrer qu'un clic sur une occurrence peut entraîner le déclenchement d'un mouvement continu.

Observez bien la position de l'écouteur gérant l'événement `ENTER_FRAME` : il se trouve dans la fonction de rappel qui est appelée uniquement au moment du clic sur une occurrence.

Par ailleurs, nous avons également cherché à démontrer qu'une occurrence peut se voir attribuer plusieurs écouteurs (`roue2`, `MOUSE_DOWN` et `ENTER_FRAME`).

Les encres

Lorsque nous faisons référence aux propriétés d'une occurrence en ActionScript, nous sous-entendons généralement celles que nous venons d'aborder dans la première partie de ce chapitre. Cependant, il en existe d'autres. Parmi celles-ci, les encres permettent de régler le mode de surimpression d'une occurrence par rapport au contenu de son arrière-plan. L'attribution de cette propriété à une occurrence se fait assez simplement en utilisant la même syntaxe que celle que nous venons de voir dans le développement précédent.

La différence est que le nom de l'encre doit toujours être précédé du terme `BlendMode` et d'un point.

```
maPhoto.blendMode = BlendMode.OVERLAY;
```

Attirons tout de suite votre attention sur les détails suivants : le nom de l'encre ne peut s'écrire qu'en majuscules et la propriété `blendMode` commence par une minuscule, alors que le terme situé après le signe égal débute par une majuscule.

Voici un premier exemple très simple, où le changement d'encre s'effectue lors d'un clic.

Fichier de référence : `Chapitre4/encres1.fla`

```
btAccueil.addEventListener(MouseEvent.MOUSE_DOWN,surbrillance);

function surbrillance(evt:Event) {
  btAccueil.blendMode = BlendMode.SCREEN;
}
```

Il existe 14 encres utilisables qui ont des effets très variés. En voici une liste exhaustive :

- `ADD`
- `ALPHA`
- `DARKEN`
- `DIFFERENCE`
- `ERASE`
- `HARDLIGHT`
- `INVERT`
- `LAYER`
- `LIGTHEN`
- `MULTIPLY`
- `NORMAL`
- `OVERLAY`
- `SCREEN`
- `SUBTRACT`

Afin de pouvoir vous rendre compte de l'effet de chaque encre, consultez l'animation `encres2.fla` qui démontre les modes de surimpression d'une image bitmap sur une autre.

Le script de cette animation est par ailleurs intéressant car il permet d'expliquer le changement d'apparence d'une occurrence lors d'un survol (ou rollover), remplaçant ainsi l'effet de survol de certains symboles de type Bouton.

Attention !

Comme vous pouvez le constater, le script suivant contient deux parties : il s'agit de deux exemples placés dans le même fichier. Les trois lignes de commentaires signalées par des barres obliques (doubles *slashs*) marquent la séparation entre les deux scripts.

Fichier de référence : `Chapitre4/encres2.fla`

```
btAccueil.addEventListener(MouseEvent.MOUSE_OVER,surbriller);
btAccueil.addEventListener(MouseEvent.MOUSE_OUT,annulerSurbrillance);

function surbriller(evt:Event) {
  btAccueil.blendMode = BlendMode.ADD;
}
function annulerSurbrillance(evt:Event) {
  btAccueil.blendMode = BlendMode.NORMAL;
}
//
//
//
btAccueil2.addEventListener(MouseEvent.MOUSE_OVER,surbrillance);
btAccueil2.addEventListener(MouseEvent.MOUSE_OUT,surbrillance);
btAccueil2.addEventListener(MouseEvent.MOUSE_DOWN,surbrillance);

function surbrillance(evt:Event) {
  if (evt.type=="mouseOver") {
    btAccueil2.blendMode = BlendMode.SCREEN;
  }
  if (evt.type=="mouseOut") {
    btAccueil2.blendMode = BlendMode.NORMAL;
  }
  if (evt.type=="mouseDown") {
    btAccueil2.blendMode = BlendMode.OVERLAY;
  }
}
```

Si vous avez lu le chapitre 3 de ce livre, ce script ne devrait vous poser aucune difficulté : il fait appel à des écouteurs et des fonctions de rappel. Seule nouveauté, l'utilisation de la propriété `blendMode`.

Si vous ne connaissez pas les structures conditionnelles utilisées pour effectuer des tests, mettez cette partie du script de côté ; vous y reviendrez lorsque vous aurez assimilé le chapitre 9. Pour l'instant, nous pouvons simplement résumer ces lignes d'instructions en précisant que le changement d'attribution d'une encre se fait en fonction de l'événement souris (`mouseOver`, `mouseOut` ou `mouseDown`).

Passons à présent à un deuxième script dont l'objectif est également d'attribuer une encre à une occurrence. Ici, le choix de la surimpression est proposé dans un menu déroulant.

Remarque

Nous sommes obligés d'importer les classes `ComboBox` et `DataProvider` car nous utilisons un composant de type ComboBox.

Fichier de référence : `Chapitre4/encres3.fla`

```
import fl.controls.ComboBox;
import fl.data.DataProvider;

var encres:Array = new Array("add","alpha","darken","difference","erase","hardlight",
➥"invert","layer","lighten","multiply","normal","overlay","screen","subtract");

var donneesEncres:DataProvider = new DataProvider();
for each (var nomEncre:String in encres) {
  donneesEncres.addItem({label:nomEncre,value:nomEncre});
}

listeDesEncres.dataProvider = donneesEncres;
listeDesEncres.rowCount = 10;
listeDesEncres.addEventListener(Event.CHANGE,entreeSelectionnee);

function entreeSelectionnee(evt:Event) {
  var encreSelectionnee = listeDesEncres.selectedItem.value;
  visuel.blendMode =encreSelectionnee;
}
```

Apportons quelques explications à ce script car il diffère de tous ceux que nous avons pu voir jusqu'à présent.

Remarque

Les chapitres 8 et 10 abordent les notions de tableau et de boucle auxquelles nous faisons appel dans ce script.

Ligne 4 : nous créons un tableau intitulé `encres` dans lequel nous stockons l'ensemble des encres disponibles pour la propriété `blendMode`.

Ligne 6 : nous créons une instance de la classe `DataProvider()` afin d'y stocker les données de notre composant.

Lignes 7 à 9 : cette boucle `for()` permet d'ajouter une à une dans l'instance de la classe `DataProvider()`, intitulée `donneesEncres`, toutes les propriétés contenues dans le tableau `encres`.

Ligne 11 : nous remplissons l'instance du composant de type `ComboBox` qui se trouve sur la scène et s'intitule `listeDesEncres`.

Ligne 12 : nous définissons 10 lignes visibles lorsque l'instance du `ComboBox` se déroule sur la scène (figure 4-1).

Ligne 13 : nous attribuons un écouteur à l'instance `listeDesEncres`, afin qu'un changement d'encre s'effectue au moment de la sélection d'une commande dans le menu déroulant.

Lignes 15 à 18 : la fonction de rappel (appelée par l'écouteur) évoquée à la ligne 13, fait référence à la valeur associée commande sélectionnée dans le menu déroulant. Rappelons qu'elle avait été définie à la ligne 8.

Figure 4-1

Afin de pouvoir visualiser les effets de surimpression d'une encre sur une image, sélectionnez-en une dans le menu déroulant de gauche (une instance de la classe ComboBox()), le résultat est alors instantané.

Les filtres

Comme vous le montrent les figures 4-2 et 4-3, les filtres proposés dans Flash ne sont pas comparables à ceux que nous retrouvons dans Photoshop ou d'autres logiciels de traitement d'image bitmap, à l'exception du flou. Ils sont plutôt comparables aux Options de fusion des calques (figure 4-4).

Figure 4-2

Les filtres peuvent être appliqués à une occurrence par le biais de l'interface ou en ActionScript.

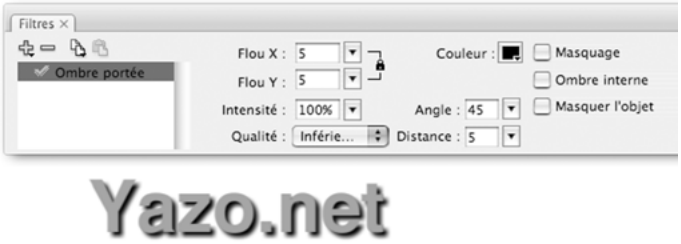


Figure 4-3

Flash propose 7 filtres différents tous très utiles en fonction de vos besoins.

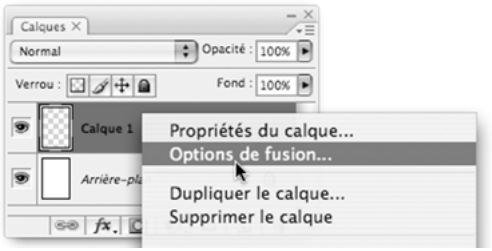


Figure 4-4

Les filtres de Flash sont comparables aux Options de fusion d'un calque dans Photoshop.

Les filtres peuvent être appliqués à une occurrence par le biais de l'interface mais également à partir de lignes d'instructions en ActionScript. Avant de découvrir un premier script, il est important de comprendre les deux points suivants :

- Vous n'appliquez pas un filtre à une occurrence, mais vous lui attribuez en fait une liste de filtres, représentée par un tableau en ActionScript.
- Vous devez importer les classes des filtres à utiliser, ce qui n'est pas effectué par défaut. De ce fait, cela sous-entend que vous allez faire appel à la directive `import` en début de script.

Voici un premier exemple simplifié :

Fichier de référence : `Chapitre4/filtres1.fla`

```
import flash.filters.DropShadowFilter;

var listeDesFiltres:Array = new Array();
var ombreLegere:DropShadowFilter = new DropShadowFilter();
listeDesFiltres.push(ombreLegere);

leLogo.filters = listeDesFiltres;
```

À la lecture de ce premier script, vous pourriez vous demander pourquoi il faut autant de lignes d'instructions pour appliquer une simple ombre portée.

Figure 4-5

Pour appliquer une simple ombre portée à une occurrence, il est nécessaire d'écrire 4 lignes d'instructions.

Tout d'abord, cela est dû à l'obligation d'utiliser un tableau : sa création puis l'ajout d'un élément nécessitent au minimum deux lignes d'instructions.

Ensuite, dans la mesure où il existe plusieurs types de filtres, nous devons en créer au moins un en instanciant la classe de filtre appropriée.

On peut enfin affecter à la propriété `filters` la liste des filtres choisis.

Dans l'exemple de la figure 4-5, nous pouvons constater que l'ombre portée est assez foncée et présente un faible décalage et un contour peu progressif. Pour modifier cet effet, nous allons, dans le script suivant, paramétrer un filtre avant de l'ajouter à liste des filtres applicables à l'occurrence.

Fichier de référence : `Chapitre4/filtres2.fla`

```
var listeDesFiltres:Array = new Array();

var ombreLegere:DropShadowFilter = new DropShadowFilter();
ombreLegere.distance = 6;
ombreLegere.alpha = 0.7;
ombreLegere.blurX = ombreLegere.blurY = 8;
ombreLegere.angle = 135;

listeDesFiltres.push(ombreLegere);

leLogo.filters = listeDesFiltres;
```

Le paramétrage d'un filtre est très simple ! Comme vous pouvez le constater, il suffit de faire référence aux propriétés à modifier et de leur attribuer une nouvelle valeur.

Pour celles et ceux qui ont une bonne mémoire, voici une ligne d'instruction qui permet de préciser les valeurs des propriétés au moment de l'instanciation de la classe `DropShadowFilter()`.

```
var ombreLegere:DropShadowFilter = new DropShadowFilter(4, 45, 0, 1, 4, 4, 1, 1,
➥false, false, false);
```

Le format de l'instanciation de la classe est le suivant : DropShadowFilter(distance, angle, couleur, transparence, flou horizontal, flou vertical, intensité, qualité, ombre interne ou externe, remplir l'occurrence de la couleur de fond de scène, occurrence cachée).

Il est évident que vous ne pouvez pas omettre une valeur ou en intervertir deux ; vous devez donc préciser toutes les valeurs ou aucune.

Voici un tableau de synthèse qui vous présente les valeurs minimale et maximale des différentes propriétés d'un filtre de type ombre portée (`DropShadowFilter`).

Propriété	Descriptif	Valeur par défaut	Valeurs mini/maxi
`distance`	Distance de l'ombre (figure 4-8)	4 pixels	De -255 à 255 en théorie, aucune limite en pratique
`angle`	Angle de l'ombre (figure 4-8)	45 degrés	De 0 à 360 en théorie, aucune limite en pratique.
`color`	Couleur de l'ombre	0 (pour le noir)	De 0x000000 à 0xFFFFFF (couleur hexadécimale)
`alpha`	Transparence de l'ombre (figure 4-6)	1 (Pour 100 %)	De 0 à 1 (0,5 pour 50 %)
`blurX`	Bord progressif horizontal de l'ombre (figure 4-6)	4	De 0 à 255. Au-delà de 80, l'ombre n'est plus réellement perceptible.
`blurY`	Bord progressif vertical de l'ombre	4	De 0 à 255. Au-delà de 80, l'ombre n'est plus réellement perceptible.
`strength`	Recouvrement entre l'ombre et l'arrière-plan (figure 4-7)	1	De 0 à 255 en théorie, aucune limite en pratique. Utilisez des valeurs faibles, comprises entre 0 et 1. Dans le cas contraire, vous annulez progressivement les bords progressifs (blurX et blurY).
`quality`	Qualité du rendu de l'ombre	1	Trois valeurs possibles théoriquement : `BitmapFilterQuality.LOW`, `BitmapFilterQuality.MEDIUM` et `BitmapFilterQuality.HIGH`. Vous pouvez aussi utiliser des nombres compris entre 1 et 3, ou encore plus grand (pour un meilleur rendu).
`Inner`	Ombre intérieure ou extérieure	false	True (intérieure) ou false (extérieure).
`knockout`	Occurrence remplie de la couleur de la scène.	false	True (remplie par la couleur de la scène) ou false (vide).
`hideObject`	Occurrence caché pour ne visualiser que l'ombre.	false	True (cachée) ou false (visible).

Figure 4-6

Le paramètre blurX permet de régler la valeur de fondu autour de l'ombre. À gauche, le contour de l'ombre n'est pas progressif alors qu'il l'est sur l'image de droite.

Figure 4-7

L'intensité de l'ombre est paramétrable avec les propriétés strength ou alpha.

Figure 4-8

Au centre, une ombre portée normale ; à gauche, l'angle de rotation a été réglé à 240° ; à droite, le décalage de l'ombre a été réglé à 9 pixels.

Nous avons abordé le filtre ombre portée (`DropShadowFilter`) pour vous présenter la technique d'utilisation des filtres en ActionScript. Il s'agit du filtre le plus couramment utilisé, avec le flou (`BlurFilter`), et le plus facile à comprendre.

Il en existe d'autres, tout aussi faciles à manipuler :

- `BevelFilter`
- `GlowFilter`
- `GradientGlowFilter`
- `GradientBevelFilter`

Et d'autres un peu plus complexes :

- `ColorMatrixFilter`
- `ConvolutionFilter`
- `DisplacementMapFilter`

Consultez l'aide en ligne de Flash pour découvrir les différents réglages à effectuer pour ces derniers.

Revenons sur les filtres `BevelFilter`, `GardientBevelFilter`, `GlowFilter`, `GradientGlowFilter` et `BlurFilter` pour vous présenter leurs propriétés respectives. Vous découvrirez qu'il s'agit des mêmes que celles du filtre `DropShadowFilter`.

- `BlurFilter` : `blurX`, `blurY` et `quality`.
- `BevelFilter` : `distance`, `angle`, `highlightColor`, `highlightAlpha`, `shadowColor`, `shadowAlpha`, `blurX`, `blurY`, `strength`, `quality`, `type`.
- `GlowFilter` : `color`, `alpha`, `blurX`, `blurY`, `strength`, `quality`, `inner`, `knockout`.
- `GradientGlowFilter` : `distance`, `angle`, `colors`, `alphas`, `ratios`, `blurX`, `blurY`, `strength`, `quality`, `type`.
- `GradientBevelFilter` : `distance`, `angle`, `colors`, `alphas`, `ratios`, `blurX`, `blurY`, `strength`, `quality`, `type`.

Pour les filtres `BevelFilter` et `GradientBevelFilter`, ajoutons que vous devez préciser des couleurs supplémentaires, mais qu'elles s'écrivent également en hexadécimal. Par ailleurs, la valeur de la propriété `type` est soit `inner`, soit `knockout`.

Pour les filtres `GradientBevelFilter` et `GradientGlowFilter`, les propriétés `colors`, `alphas` et `ratios` sont des tableaux.

Figure 4-9

Filtre BlurFilter avec une valeur horizontale uniquement (à gauche) et deux valeurs identiques horizontale et verticale (à droite).

Figure 4-10

Le filtre BevelFilter permet de donner un effet de relief.

Figure 4-11

Le filtre GlowFilter permet de donner un effet de lueur externe (ou interne).

Quel que soit le filtre que vous utilisez, n'oubliez pas d'importer les classes correspondantes.

```
import flash.filters.DropShadowFilter;
import flash.filters.BlurFilter;
import flash.filters.BevelFilter;
import flash.filters.GlowFilter;
import flash.filters.GradientBevelFilter;
import flash.filters.GradientGlowFilter;
import flash.filters.ColorMatrixFilter;
import flash.filters.ConvolutionFilter;
import flash.filters.DisplacementMapFilter;
```

Pour conclure, il est important que vous gardiez en tête les informations suivantes :

- Toutes les ombres de vos occurrences sur une image (de scénario) doivent avoir le même angle. Dans le cas contraire, vous obtiendrez une incohérence graphique.
- Ne durcissez pas trop vos ombres. Cela signifie que vous devez régler l'intensité ou l'alpha de votre ombre de telle sorte qu'elle reste légère.
- Des ombres sans contour progressif ne sont pas très esthétiques.
- Évitez les biseaux, ce n'est pas très esthétique. Au cas où vous devriez tout de même utiliser cet effet, essayez d'adoucir les bords.
- Trop de filtres tuent l'effet ! Évitez de combiner trop de filtres.
- Si vous utilisez de nombreux filtres sur une même image (de scénario), attention de ne pas tous les régler en qualité supérieure. Vous ralentiriez alors l'affichage, notamment dans les mouvements d'occurrences possédant un filtre.
- Pensez à utiliser l'option `hideObject` pour manipuler l'ombre d'une occurrence (vous devez alors posséder deux fois la même occurrence sur la scène : une sans ombre, l'autre avec, et l'option `hideObject` activée, valeur `true`).

Effet de bouton qui s'enfonce

L'effet d'enfoncement lors d'un clic sur l'occurrence d'un clip est une animation des plus classiques. Nous allons voir comment la programmer.

Fichier de référence : `Chapitre4/filtres3.fla`

```
import flash.filters.DropShadowFilter;
var listeDesFiltres:Array = new Array();

var ombreLegere:DropShadowFilter = new DropShadowFilter();
ombreLegere.distance = 6;
ombreLegere.alpha = 0.7;
ombreLegere.blurX = ombreLegere.blurY = 8;
ombreLegere.angle = 45;

listeDesFiltres.push(ombreLegere);
leLogo.filters = listeDesFiltres;

leLogo.addEventListener(MouseEvent.MOUSE_DOWN,enfoncerBouton);
leLogo.addEventListener(MouseEvent.MOUSE_UP,relacherBouton);

function enfoncerBouton(evt:MouseEvent) {
  leLogo.filters=null;
  leLogo.x+=6;
  leLogo.y+=6;
}

function relacherBouton(evt:MouseEvent) {
  leLogo.filters = listeDesFiltres;
  leLogo.x-=6;
  leLogo.y-=6;
}
```

Le début du script est commun à ceux que nous venons de présenter. Nous avons simplement ajouté deux écouteurs d'événements qui surveillent l'instant où le bouton de la souris va être enfoncé et celui où il va être relâché.

La fonction `enfoncerBouton()` va annuler le filtre appliqué à l'occurrence `leLogo` et décaler cette dernière de 6 pixels vers la droite et le bas de la scène. Symétriquement, la fonction `relacherBouton()` replace l'occurrence à sa position d'origine et réapplique le filtre.

Ce script est relativement simple mais plutôt long. Il serait utile de l'optimiser en n'ayant qu'une seule fonction contenant un test conditionnel, comme nous l'avons fait avec le fichier `encre2.fla`.

Script dans un fichier AS

Fichiers de référence : `Chapitre4/filtres3as.fla` et `mainFiltre.as`

Voyons à présent comment nous pourrions transcrire ce script dans un fichier externe (`mainFiltre.as`).

```
package {

  import flash.display.MovieClip;
  import flash.events.MouseEvent;
  import flash.filters.DropShadowFilter;

  public class mainFiltre extends MovieClip {

    private var listeDesFiltres:Array = new Array();
    private var ombreLegere:DropShadowFilter = new DropShadowFilter();

    function mainFiltre() {

      ombreLegere.distance = 6;
      ombreLegere.alpha = 0.7;
      ombreLegere.blurX = ombreLegere.blurY = 8;
      ombreLegere.angle = 45;

      listeDesFiltres.push(ombreLegere);
      leLogo.filters = listeDesFiltres;

      leLogo.addEventListener(MouseEvent.MOUSE_DOWN,enfoncerBouton);
      leLogo.addEventListener(MouseEvent.MOUSE_UP,relacherBouton);

      function enfoncerBouton(evt:MouseEvent) {
        leLogo.filters=null;
        leLogo.x+=6;
        leLogo.y+=6;
      }

      function relacherBouton(evt:MouseEvent) {
        leLogo.filters = listeDesFiltres;
        leLogo.x-=6;
        leLogo.y-=6;
      }
    }

  }

}
```

Vous observerez que nous devons importer la classe `DropShadowFilter()`, comme nous l'avions fait avec une syntaxe de programmation séquentielle. Les deux variables `listeDesFiltres` et `ombreLegere` sont de type `private` car elles ne servent qu'à l'intérieur de notre classe. En dehors de cela, rien ne change.

Remarque

Nous aurions pu utiliser ce script pour définir une classe pour notre symbole. Consultez le fichier `mainFiltre2.as` qui fonctionne avec `filtres3SymboleAS.fla` pour découvrir les adaptations du code.

Animer un filtre

Pour terminer ce développement sur les filtres, nous allons découvrir qu'il est possible d'animer un filtre pour obtenir un effet de mouvement de lumière.

Fichier de référence : `Chapitre4/filtres4.fla`

```
import flash.filters.BevelFilter;

var listeDesFiltres:Array = new Array();
var angle:Number;
var distance:Number;

var lueur:BevelFilter = new BevelFilter();
lueur.quality=3;
lueur.strength = 0.8;

listeDesFiltres.push(lueur);
laTache.filters = listeDesFiltres;

stage.addEventListener(MouseEvent.MOUSE_MOVE,reglerLumiere);

function reglerLumiere(evt:MouseEvent) {

  angle = (Math.atan2(129-mouseY, 258-mouseX)/Math.PI)*180;
  lueur.angle = angle;
  distance = Math.abs(129-mouseY)+Math.abs(258-mouseX);
  lueur.blurX = lueur.blurY = distance/10;

  listeDesFiltres = [];
  listeDesFiltres.push(lueur);
  laTache.filters = listeDesFiltres;

}
```

Les 12 premières lignes du script ci-dessus sont identiques à celles des scripts que nous avons abordés précédemment ; nous ne reviendrons donc pas sur ces explications. En revanche, essayons de comprendre comment la lueur illumine l'occurrence placée au centre de la scène (figure 4-12).

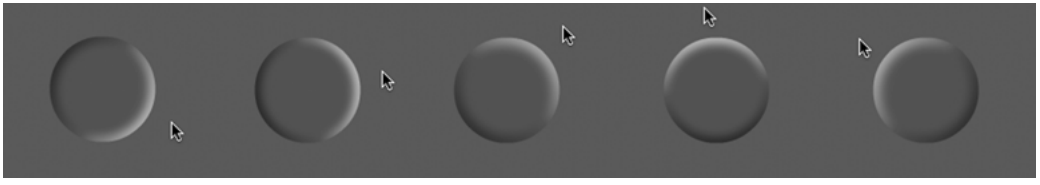


Figure 4-12

Le curseur de la souris de notre animation dégage une lueur, à tel point qu'en le déplaçant, le bord de l'occurrence le plus proche du curseur est éclairé par celui-ci.

- Nous utilisons, pour la première fois dans ce livre, l'événement `MOUSE_MOVE`. Précisons que la fonction de rappel intitulée `reglerLumiere()` ne sera exécutée qu'à partir du moment où la souris sera en mouvement sur la scène.
- Le code contenu dans la fonction se décompose en deux parties. Les trois dernières lignes appliquent le filtre, comme nous l'avons déjà vu au cours de ce développement. Les trois premières lignes permettent, elles, de calculer l'angle à utiliser pour la propriété `angle` de l'instance `lueur` et la quantité de flou à appliquer au bord de cette même occurrence, pour donner une impression d'éloignement (diminution de l'intensité de la lumière).

Vous noterez qu'exceptionnellement, le premier paramètre de la fonction `atan2()` fait d'abord référence à la propriété `mouseY` et non `mouseX`. De plus, les valeurs 129 et 258 correspondent à l'axe autour duquel la souris doit tourner pour calculer l'angle.

La couleur

Avant de vous apprendre à modifier la couleur d'une occurrence, essayons de comprendre comment fonctionne le codage de la couleur en hexadécimal. Si vous connaissez déjà cette représentation, passez les quelques pages qui vont suivre pour vous rendre au début du développement sur l'affectation d'une couleur à une occurrence.

Comprendre la couleur en hexadécimal

Que signifie 0xFF12F3 ou #FF12F3 ? C'est une question que vous vous êtes sûrement déjà posée en voyant cette codification de couleur. Nous allons donc expliciter cette représentation, et vous vous rendrez compte qu'elle est finalement assez simple à utiliser.

Cyan, magenta, jaune et noir

Si vous travaillez dans le domaine des arts graphiques, vous avez certainement l'habitude de travailler la couleur en quadrichromie, c'est-à-dire à base de cyan, magenta, jaune et noir. Le dosage de ces couleurs s'exprime en pourcentage. Ainsi, pour qualifier une couleur en CMJN, on l'exprime sous la forme suivante : cyan 10 % - magenta 23 % - jaune 37 % - noir 4 %. Le mélange des couleurs cyan, magenta et jaune dosées à 100 % permet d'obtenir théoriquement du noir. Ces couleurs sont constituées de pigments qu'on peut déposer sur une surface.

Le dosage d'une couleur s'exprime en pourcentage, partant de 0, pour une absence de la couleur (ce qui laisse alors apparaître la couleur du papier), à 100, pour un dosage maximal, recouvrant ainsi intégralement la couleur du papier. Lorsque vous travaillez dans ce mode de couleur (CMJN), on parle de méthode soustractive.

Rouge, vert, bleu

En revanche, si vous travaillez dans le monde de la vidéo, vous n'avez pas l'habitude de manipuler les couleurs que nous venons d'évoquer, mais plutôt le rouge, le vert et le bleu. Le dosage de ces couleurs ne s'exprime pas en pourcentage, mais sur une base de 256 valeurs allant de 0 à 255. Ainsi, pour définir une couleur en RVB, on l'exprime sous la forme suivante : rouge 128 - vert 0 - bleu 255. Le mélange de ces trois couleurs dosées au maximum (255) permet d'obtenir du blanc. Rappelons que les couleurs d'un téléviseur ou d'un vidéo projecteur sont obtenues grâce à des rayons de lumières. Lorsque vous travaillez dans ce mode de couleur (RVB), on parle de méthode additive.

> **Rappel**
> Le noir et le blanc ne sont pas des couleurs, mais des valeurs de couleurs.

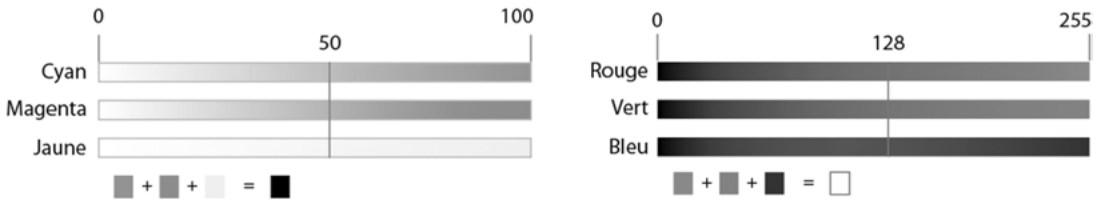


Figure 4-13

Les deux modes de couleurs CMJN et RVB sont utilisés respectivement dans le monde des arts graphiques et de la vidéo.

Vous aimeriez peut-être comprendre pourquoi nous évoquons ces deux modes alors que nous nous apprêtons à présenter le codage de la couleur en hexadécimal ? Il est important que vous sachiez dès à présent que cette représentation se base sur le mode RVB. En observant la partie droite de la figure 4-13 et l'intégralité de la figure 4-14, vous pouvez constater qu'il s'agit des mêmes dégradés de couleurs (même si vous les voyez en niveaux de gris). En revanche, vous observerez que la graduation ne varie pas de 0 à 255, mais de 00 à FF. C'est cette représentation (utilisation de chiffres et de lettres pour représenter un nombre), appelée notation hexadécimale, que nous allons vous présenter.

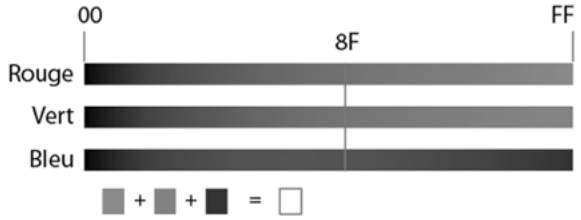


Figure 4-14

Le codage d'une couleur en hexadécimal se base sur toujours sur 256 valeurs par couche mais utilise une représentation hexadécimale des nombres.

Avant de vous expliquer comment lire une couleur en hexadécimal, revenons sur des notions que vous possédez déjà. Le tableau de la figure 4-15 contient des valeurs que vous manipulez depuis votre plus tendre enfance. Nous avons alterné la coloration des lignes paires et impaires pour vous démontrer qu'en les mettant bout à bout, nous obtiendrions une ligne comme celle que vous pouvez retrouver sur la figure 4-17. Il s'agit de la notation en base 10, dans laquelle nous avons besoin de dix chiffres pour représenter les nombres.

En hexadécimal, nous utilisons une base 16, c'est-à-dire qu'en plus des dix chiffres, nous ajoutons les lettres A, B, C, D, E et F. En effet, comme il n'existe pas d'autres chiffres pour compter, il a fallu utiliser des signes que nous connaissions déjà. C'est pourquoi les lettres ont été retenues pour faire partie de la représentation d'une nouvelle base.

À l'école, dans les classes primaires, l'échelle des notes s'étend généralement de 0 à 10 car les maîtres des écoles n'ont pas besoin d'utiliser plus de valeurs. Au collège puis au lycée, il est nécessaire de pouvoir nuancer davantage l'évaluation, les notes s'étalent donc de 0 à 20. Ces deux exemples nous permettent de comprendre que plus une échelle ou une graduation est étendue, plus elle offre de possibilités de nuances, de variations. Ainsi, si vous deviez remplir chaque cellule du tableau de la figure 4-15 avec une couleur différente, puis effectuer la même opération pour le tableau de la figure 4-16, vous obtiendriez bien plus de nuances dans le tableau utilisant les valeurs en hexadécimal. L'utilisation d'une notation hexadécimale permet d'obtenir, avec deux « chiffres », 256 couleurs, alors que nous n'en aurions que 100 en base 10…

	0	1	2	3	4	5	6	7	8	9
0	00	01	02	03	04	05	06	07	08	09
1	10	11	12	13	14	15	16	17	18	19
2	20	21	22	23	24	25	26	27	28	29
3	30	31	32	33	34	35	36	37	38	39
4	40	41	42	43	44	45	46	47	48	49
5	50	51	52	53	54	55	56	57	58	59
6	60	61	62	63	64	65	66	67	68	69
7	70	71	72	73	74	75	76	77	78	79
8	80	81	82	83	84	85	86	87	88	89
9	90	91	92	93	94	95	96	97	98	99

Figure 4-15

Les nombres de 0 à 99 exprimés en base 10.

Le tableau de la figure 4-16 est obtenu comme celui de la figure 4-15, en combinant les chiffres ou les lettres qui figurent à gauche de chaque ligne et en haut de chaque colonne. Si nous construisons maintenant deux échelles contenant toutes les lignes des tableaux des figures 4-16 et 4-15, nous voyons à quel point la notation hexadécimale permet de représenter un plus grand nombre de valeurs avec la même quantité de caractères.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
1	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
2	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
3	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
4	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
5	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
6	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
7	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
8	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
9	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

Figure 4-16

Les nombres hexadécimaux (en base 16) de 00 à FF.

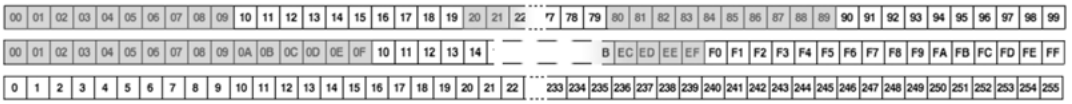


Figure 4-17

Ces trois échelles de valeurs représentent toutes un dosage, une mesure. Une notation sur 20 est comparable à une notation sur 10 : dans les deux cas, 0 est la plus mauvaise note alors que 20 ou 10 est la note la plus forte. Ce qui change, c'est la précision de la notation ; 10/20 est la même note que 5/10. On comprend alors que, dans notre figure, les valeurs 100, 255 et FF représentent toutes un même dosage maximum.

Voyons maintenant comment lire une valeur en hexadécimal. Gardez tout d'abord à l'esprit cette première évidence : sur une graduation en base 10, le 0 correspond à la plus petite valeur et le 9 à la plus grande. Dites-vous à présent, en vous aidant pour cela de la figure 4-16, que la plus petite valeur sur une graduation en base 16 est 0 et la plus grande est F.

Prenons l'exemple suivant : Luna, 11 ans, a une sœur Marine de 19 ans. Luna s'adresse à son grand-père, qui a 91 ans, et lui dit : « Tu sais papi, Marine, elle est vieille. Elle a de la chance, elle peut sortir avec ses copines, elle a une voiture, un amoureux… Elle est vieille ! » Le grand-père de Marine, devant la naïveté de sa petite fille, ne peut que lui répondre : « Oui, tu as raison. Elle est vieille ! ». Revenons maintenant sur les âges des trois personnages de notre histoire : 11, 19 et 91 ans. Bien que les âges du grand-père et de Marine soient tous les deux composés des mêmes chiffres, 1 et 9, le résultat n'est pas du tout le même car l'ordre des chiffres a de l'importance.

Nous constatons en effet que les âges de Luna et Marine se trouvent tous les deux sur la deuxième ligne du tableau de la figure 4-15, alors que celui de leur papi se trouve dans la dernière ligne.

En conclusion, les chiffres en deuxième position dans les âges de Luna et de Marine (1 et 9) permettent de les situer chacune dans la première dizaine, mais les âges restent au début d'une échelle allant de 0 à 99, alors que le grand-père se trouve à la fin de l'échelle, même s'il est au début de la dernière dizaine du tableau ! Le premier chiffre d'un nombre a donc un poids beaucoup plus important. C'est exactement pareil dans la notation hexadécimale.

Ainsi, lorsque vous voyez une couleur sous la forme FFFF00, comprenez tout d'abord qu'il s'agit d'un codage de couleur en hexadécimal qui s'appuie sur le mode RVB (sous la forme RRVVBB plus précisément). Ainsi au lieu d'écrire 255 255 0, on écrit FF FF 00. En collant les valeurs, on obtient le nombre FFFF00. Mais pourquoi ce double F ? Regardez le tableau de la figure 4-16, vous constaterez que la valeur FF correspond à la 256e valeur du tableau, c'est à dire le nombre 255 en base 10 (la première valeur du tableau est 0, la dernière 255).

Essayez maintenant les deux couleurs suivantes codées en hexadécimal : 0FEFF0 et 10F0EF. Vous pouvez constater qu'elles sont quasiment identiques ; pour comprendre pourquoi, répondez à ces trois questions :

- Quelle valeur se trouve après 0F en hexadécimal ?
- Quelle valeur se trouve après EF en hexadécimal ?
- Quelle valeur se trouve avant F0 en hexadécimal ?

Si vous concaténez toutes vos réponses, vous obtenez la deuxième couleur : 10F0EF (voir la partie droite de la figure 4-19 pour comprendre comment la lire).

Peut-être ne comprenez-vous toujours pas. Rappelez-vous notre avant-dernier propos : seul le premier des deux chiffres de chaque paire est important. L'exemple de gauche de la figure 4-19 vous démontre que la couleur codée par cette valeur hexadécimale correspond à un magenta. F11FED signifie en effet rouge F1, vert 1F et bleu ED. On ne retient que les premiers chiffres de chaque couleur, c'est-à-dire F, 1 et E. Sur une échelle allant de 0 à F, la valeur F du rouge se trouve à la fin, cela signifie donc beaucoup de rouge. Toujours sur une échelle de 0 à F, la valeur 1 du vert se trouve au début, cela signifie donc peu de vert. Et enfin, en appliquant le même raisonnement à la composante bleue nous déduisons que la couleur finale contient beaucoup de bleu.

Figure 4-18

Ces deux valeurs, bien que composées des mêmes chiffres, ne se trouvent pas au même endroit sur une échelle en hexadécimal (de 0 à F) ; de même pour les nombre 19 et 91 sur une échelle de 0 à 99 en base 10.

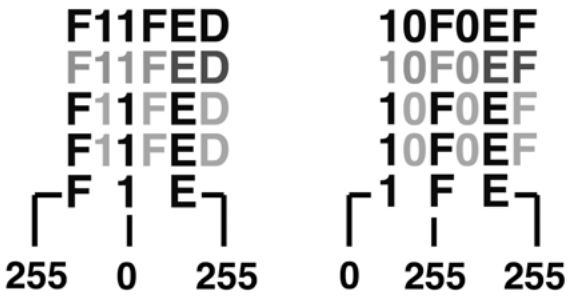


Figure 4-19

Voici comment faciliter la lecture d'une couleur hexadécimale. Évidemment nous simplifions lorsque nous associons à E la valeur 255 et 1 la valeur 0.

Mais comment peut-on savoir que du rouge additionné à du bleu permet d'obtenir du magenta, ou que du rouge additionné à du vert permet d'obtenir du jaune ?

Le schéma de la figure 4-20 vous montre que pour avoir une idée de la couleur obtenue à partir d'un codage RVB, il suffit de consulter la lettre C (cyan), M (magenta) ou J (jaune) qui ne se trouve pas en face des deux lettres à additionner dans l'autre mode. Ainsi, pour obtenir du cyan, il faut utiliser du vert et du bleu. Cela marche également dans l'autre sens : pour obtenir du rouge, il faut mélanger du magenta et du jaune.

Figure 4-20

Une couleur dans un mode s'obtient en ajoutant les deux couleurs opposées de l'autre mode.

Si la première lecture de ces quelques pages vous a parue un peu difficile, nous vous conseillons de ne pas relire tout de suite ces quelques explications, mais d'effectuer une deuxième lecture dans quelques minutes.

Affectation d'une couleur à une occurrence

Fichier de référence : `Chapitre4/couleur.fla`

L'application d'une couleur à une instance est nettement plus facile que la compréhension du mode de calcul d'une couleur en hexadécimal, mais elle n'est tout de même plus aussi simple qu'en AS1/AS2.

> **Remarque**
>
> Pour celles et ceux qui utilisaient déjà la classe `ColorTransform()` en AS1/AS2 , il n'y a pas de changement en AS3.

Considérons tout de suite l'exemple suivant :

```
var modifCouleur:ColorTransform = pastille1.transform.colorTransform;
modifCouleur.color = 0x008800;
pastille1.transform.colorTransform = modifCouleur;
```

La première ligne de ce script a pour but de créer un objet de type `ColorTransform` qui stocke une des informations d'ordre colorimétrique obtenue à l'aide de la propriété `colorTransform`. Dans notre exemple, nous nous servons de l'occurrence `pastille1` pour récupérer ses attributs, mais nous aurions pu partir de zéro.

La deuxième ligne fait appel à la propriété `color` pour affecter une couleur à l'instance `modifCouleur`.

La troisième ligne permet, quant à elle, d'appliquer concrètement la couleur à l'occurrence intitulée `pastille1` qui se trouve sur la scène.

Comme nous venons de l'évoquer, nous pourrions aussi créer un objet directement en instanciant la classe `ColorTransform()`.

```
var modifCouleur:ColorTransform = new ColorTransform();
modifCouleur.color = 0x008800;
pastille1.transform.colorTransform = modifCouleur;
```

Quelle que soit la méthode que vous utiliserez, que vous récupériez les informations d'une occurrence existante ou que vous instanciez la classe `ColorTransform()`, vous devrez toujours définir la couleur à attribuer en faisant appel à la propriété `color` et terminer votre processus par la méthode `colorTransform()`.

Rendre une occurrence mobile

Avant de vous présenter le code qui permet de rendre une occurrence mobile, distinguons dès à présent les deux cas suivants :

- Une occurrence peut être rendue mobile au lancement de l'animation, de façon automatique, sans que l'utilisateur n'ait à cliquer à un quelconque endroit de la scène. C'est notamment le cas pour un curseur personnalisé ou une infobulle qui suit le curseur : aucun gestionnaire d'événement n'est alors nécessaire, une simple ligne d'instruction suffit.
- Une occurrence peut être mobile uniquement si l'utilisateur la saisit pour la déplacer avec la souris.

Remarque

Il est impossible de rendre plusieurs occurrences mobiles avec la méthode `startDrag()`. Si vous aviez un tel besoin, il faudrait utiliser l'événement `ENTER_FRAME` ou faire appel à la classe `Timer()` pour exécuter en continu le déplacement d'une ou plusieurs occurrences (collées au curseur de la souris).

Mobilité automatique

Fichier de référence : `Chapitre4/draganddrop1.fla`

Comme nous l'avons présenté au début de ce livre, lorsque vous devez exécuter une action automatiquement au lancement d'une animation, il faut votre code sur une image-clé, en-dehors de tout gestionnaire d'événement, ou dans le constructeur de la classe de votre document. Quel que soit le mode de programmation que vous retenez, voici l'unique ligne d'instruction qui permet de rendre une occurrence mobile.

```
monCurseur.startDrag(true);
```

Dans notre exemple, `monCurseur` correspond au nom d'une occurrence sur la scène. Le paramètre `true` est nécessaire lorsque vous rendez une occurrence mobile automatiquement. Si vous le réglez à `false`, valeur par défaut si vous ne précisez rien, l'occurrence ne se retrouvera pas fixée au curseur de votre souris.

Masquer le curseur

Dans certains cas, vous aurez besoin de masquer le curseur de votre souris. Ajoutez alors simplement la ligne d 'instruction ci-dessous.

```
Mouse.hide();
```

Pour réafficher le curseur, utilisez la méthode `show()`.

```
Mouse.show();
```

Mobilité manuelle ou glisser-déposer

Fichier de référence : `Chapitre4/draganddrop1.fla`

Lorsque vous aurez besoin d'effectuer un glisser-déposer (*drag and drop*) d'une occurrence sur la scène, vous devrez la rendre mobile. Pour cela, analysons tout d'abord les étapes nécessaires à la programmation de cette opération.

1. Vous devez maintenir le clic sur une occurrence.
2. Vous devez rendre l'occurrence mobile.
3. Vous devez relâcher le bouton de votre souris pour relâcher l'occurrence.

En ActionScript, nous devons traduire et découper notre script de la même façon :

1. Définir un objet d'écoute avec l'événement `MOUSE_DOWN`.
2. Faire appel à la méthode `startDrag()` dans la fonction de rappel de l'objet d'écoute ci-dessus.
3. Définir un objet d'écoute avec l'événement `MOUSE_UP`.

Voici le code correspondant :

```
carte.addEventListener(MouseEvent.MOUSE_DOWN,carteMobile);
carte.addEventListener(MouseEvent.MOUSE_UP,carteArretee);

function carteMobile(evt:Event) {
  carte.startDrag(false);
}
function carteArretee(evt:Event) {
  stopDrag();
}
```

Si vous avez compris la manipulation des gestionnaires d'événements, vous pouvez constater que la structure de ce script est logique. Dans ce cas, essayons d'aller plus loin et d'ajouter une notion supplémentaire : effectuer une action lors du déplacement de l'occurrence.

Remarque

Si vous omettez le paramètre `false` de la méthode `startDrag()`, vous constaterez que la carte est bien rendue mobile, et que son point d'alignement ne vient pas se coller au curseur de votre souris (comme nous l'avons vu dans le cas d'une occurrence rendue automatiquement mobile). La valeur `false` ne s'avère donc pas indispensable et vous ne devez surtout pas utiliser la valeur `true` (sauf si vous souhaitez replacer le point d'alignement d'une occurrence sur l'extrémité du curseur).

Exécuter une action lors du mouvement d'une occurrence

Fichier de référence : `Chapitre4/draganddrop2.fla`

Situons notre exemple dans un contexte

Pour vous expliquer cette notion, prenons l'exemple d'une occurrence (un carré orange) dont la transparence va dépendre de la position d'une autre. Plus nous allons déplacer la carte à jouer de notre exemple à droite de la scène, plus le carré orange sera opaque.

La technique est extrêmement simple car il suffit d'ajouter un gestionnaire d'événement qui fait appel à la constante `MOUSE_MOVE`. En revanche, il est important de comprendre l'imbrication du code.

À quel endroit du script allez-vous ajouter un écouteur ? Dans la mesure où vous allez devoir cliquer sur une occurrence avant de pouvoir la déplacer, nous allons donc ajouter la ligne d'instruction ci-dessous dans la fonction de rappel de l'écouteur qui contient la constante `MOUSE_DOWN`.

```
carte.addEventListener(MouseEvent.MOUSE_MOVE,actionPendantMouvement);
```

Vous obtenez ainsi le code suivant :

```
function carteMobile(evt:Event) {
  carte.startDrag();
  carte.addEventListener(MouseEvent.MOUSE_MOVE,actionPendantMouvement);
}
```

Lorsque vous relâcherez le bouton de la souris, il faudra par contre penser à retirer/désactiver votre écouteur.

```
function carteArretee(evt:Event) {
  stopDrag();
  carte.removeEventListener(MouseEvent.MOUSE_MOVE,actionPendantMouvement);
}
```

Voici le script complet, comprenant notamment la fonction de rappel intitulée `actionPendantMouvement()`.

```
detecteur.alpha=carte.x/500;

carte.addEventListener(MouseEvent.MOUSE_DOWN,carteMobile);
carte.addEventListener(MouseEvent.MOUSE_UP,carteArretee);

function carteMobile(evt:Event) {
  carte.startDrag();
  carte.addEventListener(MouseEvent.MOUSE_MOVE,actionPendantMouvement);
}
function carteArretee(evt:Event) {
  stopDrag();
  carte.removeEventListener(MouseEvent.MOUSE_MOVE,actionPendantMouvement);
}

function actionPendantMouvement(evt:Event) {
  detecteur.alpha=carte.x/500;
}
```

Remarque

Le fichier `draganddrop2.fla` ne contient pas tout à fait le même code que celui présenté ici, car nous gérons le problème du curseur qui ne se trouve plus au-dessus de l'occurrence au moment où le bouton de la souris est relâché.

À la première ligne du script, nous commençons par régler la transparence de l'occurrence `detecteur` afin qu'elle ne subisse pas une variation soudaine au lancement de l'animation.

Ensuite, l'unique ligne d'instruction de la fonction `actionPendantMouvement()` est chargée de régler l'opacité de l'occurrence `detecteur` à chaque mouvement de souris.

Précisons que l'événement `MOUSE_MOVE` s'avérera indispensable lorsque vous réaliserez des variateurs (consultez la section suivante pour découvrir la technique de réalisation d'un variateur).

Il est très important de penser à retirer/désactiver un écouteur lorsqu'il n'est plus utile. C'est pourquoi nous ajoutons la fonction `removeListener()` dans la fonction de rappel `carteArretee()`.

Contraindre le déplacement dans une zone

Dans certains cas, il est nécessaire de contraindre le déplacement d'une occurrence dans une zone précise, l'exemple le plus fréquent étant le cas d'un variateur (figure 4-21).

Figure 4-21

Un variateur n'est ni plus ni moins qu'une occurrence rendue mobile dans une zone contrainte de zéro pixel de hauteur ou de largeur.

> **Remarque**
>
> Il est vrai qu'un composant intitulé `Slider` existe déjà dans l'interface de Flash, mais il se caractérise par un graphisme très froid propre aux composants. Nous souhaitons donc créer notre propre variateur.

Avant de découvrir le code nécessaire à la réalisation d'un variateur personnalisé, essayons de comprendre comment fonctionne la contrainte d'un déplacement.

Nous allons utiliser la méthode `startDrag()`, mais nous devons d'abord spécifier une zone de contrainte en-dehors de laquelle l'occurrence ne pourra pas sortir.

```
var zoneDeContrainte:Rectangle = new Rectangle(60,80,170,130);
```

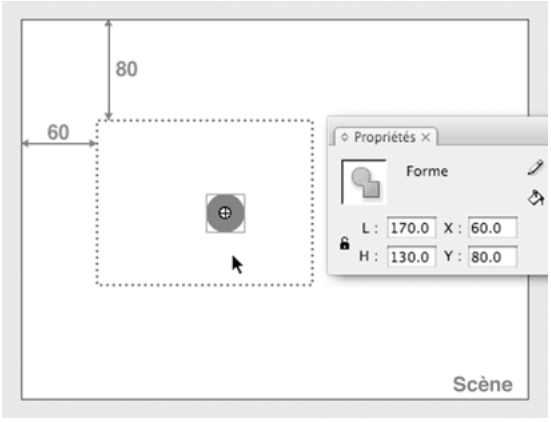


Figure 4-22

La zone de contrainte se définit en précisant la position des bords gauche et supérieur à ne pas dépasser ainsi que la largeur et la hauteur.

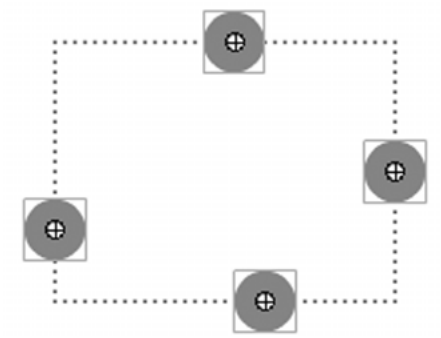
Nous pouvons ensuite utiliser l'objet `zoneDeContrainte` pour définir le deuxième paramètre attendu de la méthode `startDrag()`.

```
variateur.startDrag(false, zoneDeContrainte);
```

Dans ce premier exemple, l'occurrence `variateur` ne peut bouger que dans la zone définie en pointillés sur la copie d'écran de la figure 4-22. Pour être très précis, nous devons ajouter que c'est en fait le point d'accrochage de l'occurrence qui ne peut pas sortir de la zone, ce qui signifie que la moitié de l'instance peut se trouver à l'extérieur comme le montre la copie d'écran de la figure 4-23.

Figure 4-23

C'est le point d'alignement, ou point d'accrochage, d'une occurrence qui est contraint à ne pas sortir d'une zone de contrainte et non les bords de l'occurrence elle-même.



Pour pallier ce problème, il suffit de revoir les quatre valeurs de définition de l'instance de la classe `Rectangle()` à la baisse. Reprenons des valeurs de la figure 4-22 en sachant que l'occurrence (le rond bleu foncé) mesure 30 pixels de diamètre, soit 15 pixels de rayon. Voici les nouvelles valeurs :

```
var zoneDeContrainte:Rectangle = new Rectangle(75,95,155,115);
```

Nous avons tout simplement ajouté 15 pixels aux deux premières valeurs et soustrait 15 pixels à la largeur et la hauteur.

Revenons à notre objectif initial qui est de concevoir un variateur. Nous devons définir une hauteur de contrainte de 0 pixel pour que l'occurrence ne puisse plus se déplacer verticalement, mais uniquement horizontalement.

Fichier de référence : `Chapitre4/draganddrop3.fla`

```
var zoneLimite:Rectangle = new Rectangle(55,237,173,0);

variateur.addEventListener(MouseEvent.MOUSE_DOWN,variateurMobile);
variateur.addEventListener(MouseEvent.MOUSE_UP,variateurArrete);

function variateurMobile(evt:Event) {
  variateur.startDrag(false,zoneLimite);
}
function variateurArrete(evt:Event) {
  stopDrag();
}
```

Vous retrouverez ce script dans le fichier `draganddrop4.fla` , dans lequel nous avons ajouté quelques lignes de code pour qu'une occurrence change de taille au moment où le variateur est déplacé. Nous gérons également le problème abordé dans le développement suivant (`MOUSE_UP_OUTSIDE`).

Problème du MOUSE_UP_OUTSIDE ?

Lorsque vous relâcherez le bouton de la souris au cours d'un glisser-déposer alors que le curseur se trouve encore au-dessus de l'occurrence que vous venez de déplacer, il n'y aura aucun problème. En revanche, si le curseur ne se trouve plus sur l'occurrence, pour différentes raisons, au moment où vous relâcherez le bouton de la souris, la mobilité de l'occurrence sera toujours active : vous ne pourrez alors plus l'annuler.

Explications

Il faut simplement comprendre qu'en ayant relâché le bouton de la souris alors que le curseur n'était plus sur l'occurrence rendue mobile, l'événement `MOUSE_UP` ne s'est pas produit.

Fichier de référence : `Chapitre4/draganddrop4.fla`

Nous allons donc adapter notre script de la façon suivante :

```
var zoneLimite:Rectangle = new Rectangle(55,237,173,0);

variateur.addEventListener(MouseEvent.MOUSE_DOWN,variateurMobile);
variateur.addEventListener(MouseEvent.MOUSE_UP,variateurArrete);

function variateurMobile(evt:Event) {
  stage.addEventListener(MouseEvent.MOUSE_UP,variateurArrete);
  variateur.startDrag(false,zoneLimite);
}
function variateurArrete(evt:Event) {
  stopDrag();
  stage.removeEventListener(MouseEvent.MOUSE_UP,variateurArrete);
}
```

Si vous observez attentivement ce code, vous constaterez que nous avons ajouté deux lignes d'instructions. La première sert à détecter le relâchement du bouton de souris par l'utilisateur quelle que soit la position de la souris.

Rappel

Nous vous rappelons que la propriété `stage` d'une occurrence fait référence à la scène de l'animation.

```
stage.addEventListener(MouseEvent.MOUSE_UP,variateurArrete);
```

La deuxième sert, en revanche, à enlever l'écouteur que nous avons déclaré dans la fonction `variateurMobile()`.

Message de l'auteur

Au moment de l'écriture de ce livre, c'est la seule solution que j'ai trouvée pour pallier l'absence de gestion de l'événement `MOUSE_UP_OUTSIDE`. Cette technique présente tout de même l'inconvénient de déclencher les fonctions de rappel des occurrences sur lesquelles vous relâcherez votre clic si l'événement `MOUSE_UP` est géré.

Vérifier l'emplacement de l'occurrence

Lorsque vous effectuerez un glisser-déposer d'une occurrence sur la scène, vous souhaiterez peut-être effectuer un test au moment du relâchement pour vérifier sa position. Placez dans ce cas un test dans la fonction de rappel de l'écouteur qui gère l'événement `MOUSE_UP`. Voici un exemple :

```
function pionRelache(evt:Event) {
  stopDrag();
  if(pion.x>180) pion.x=55;
}
```

Tester la collision entre deux occurrences

Nous venons d'évoquer, dans le développement précédent, la possibilité d'évaluer la position d'une occurrence sur la scène. Dans certains cas, il sera plus facile de tester l'intersection de deux occurrences.

Pour cela, vous devez utiliser la méthode `hitTestObject()` en utilisant la syntaxe suivante :

```
nomduneoccurrence.hitTestObject(nomduneautreoccurrence);
```

L'exécution de cette ligne d'instruction a pour renvoie la valeur `true` ou `false`.

> **L'expression « renvoyer »**
>
> Le verbe renvoyer est utilisé en programmation pour exprimer le fait qu'une fonction peut afficher ou transmettre une valeur. Dans le cas présent, une fonction `trace()` sert à renvoyer ou afficher dans la fenêtre Sortie ce qu'elle contient entre parenthèses.
>
> La fonction `trace(12+35)` renvoie la valeur 47. Vous découvrirez, dans certaines fonctions, le terme `return` qui permet de renvoyer la valeur qui suit ce mot-clé.

Pour tester cette méthode, positionnez deux occurrences sur la scène, à cheval l'une sur l'autre. Nommez-les `piece1` et `piece2` et saisissez la ligne d'instruction ci-dessous :

```
trace(piece1.hitTestObject(piece2));
```

Au moment de la prévisualisation de votre animation (Ctrl-Entrée sur Windows ou Commande-Entrée sur Mac) la fenêtre Sortie de Flash affiche le mot `true`.

Fermez à présent la fenêtre de prévisualisation pour pouvoir séparer les deux occurrences et publiez à nouveau votre animation. Cette fois-ci, le mot `false` s'affiche dans la fenêtre Sortie. Cela démontre bien que nous pouvons utiliser cette ligne d'instruction dans un test conditionnel avec `if()`. Consultez éventuellement le chapitre 9 pour apprendre à écrire des tests dans un programme.

Dans l'exemple ci-dessous, nous allons faire rouler une balle en partant de la gauche de la scène pour aller vers la droite (figure 4-24). Une occurrence, symbolisant un mur, se trouve au milieu de la scène. Nous allons tester une éventuelle collision entre ces deux occurrences pour pouvoir interrompre le mouvement le cas échéant.

Figure 4-24

Lorsque l'occurrence représentée par la balle va entrer en intersection avec le mur (représenté par le trait vertical), le déplacement de la balle va être interrompu.

Fichier de référence : `Chapitre4/hittest1.fla`

```
balle.addEventListener(Event.ENTER_FRAME,avancerBalle);

function avancerBalle(evt:Event) {
  balle.x=balle.x+3;
  if (balle.hitTestObject(mur)) {
    balle.removeEventListener(Event.ENTER_FRAME,avancerBalle);
  }
}
```

Abordons un deuxième exemple un peu plus complexe, mais plus représentatif de notre problématique. Prenons une occurrence que nous souhaitons déplacer sur la scène par glisser-déposer et pour laquelle nous voulons savoir, au moment où nous la relâchons, si elle en touche une autre (figure 4-25). Voici, dans ce cas, le code à utiliser :

Figure 4-25

Au moment où l'utilisateur relâche le bouton de la souris, un test est effectué pour déterminer si les deux occurrences se touchent.

Fichier de référence : `Chapitre4/hittest2.fla`

```
piece1.addEventListener(MouseEvent.MOUSE_DOWN,pieceMobile);
piece1.addEventListener(MouseEvent.MOUSE_UP,pieceFixe);

function pieceMobile(evt:MouseEvent) {
  evt.currentTarget.startDrag();
}

function pieceFixe(evt:MouseEvent) {
  stopDrag();
  if (evt.currentTarget.hitTestObject(zoneRelache)) {
    evt.currentTarget.x=zoneRelache.x;
    evt.currentTarget.y=zoneRelache.y;
    piece1.removeEventListener(MouseEvent.MOUSE_DOWN,pieceMobile);
  }
}
```

Remarque

Si vous ne savez pas gérer le glisser-déposer d'une occurrence sur la scène, consultez le développement précédent dédié à cette technique avant de poursuivre la lecture des lignes qui vont suivre.

Observez bien la fonction `pieceFixe()` dans laquelle nous avons effectué un test qui détecte une éventuelle collision entre l'occurrence qui fait l'objet du déplacement (`evt.currentTarget`) et une autre intitulée `zoneRelache`. Si tel est le cas, nous plaçons précisément l'occurrence relâchée aux mêmes coordonnées `x` et `y` de l'instance `zoneRelache`.

Lorsque l'occurrence est à sa place, nous ne souhaitons plus qu'elle puisse être déplacée à nouveau. Nous terminons donc le script par une désactivation de l'écouteur qui fait appel à l'événement `MOUSE_DOWN`.

Remarque

Si vous ne savez pas encore effectuer une boucle en programmation avec un `for()`, n'abordez pas ce troisième exemple et reprenez-le lorsque vous aurez lu le chapitre 10.

Pour ce troisième et dernier exemple, nous allons chercher à vous expliquer comment le nom de l'instance figurant dans les parenthèses de la fonction `hitTest()` doit être écrit s'il est amené à changer. En parcourant rapidement le script ci-dessous, vous découvrirez que nous avons effectué une boucle pour placer plusieurs instances sur la scène. Nous ne pouvons donc pas utiliser la même référence, aussi bien pour le nom d'occurrence placé avant la fonction que pour celui situé après.

Fichier de référence : `Chapitre4/hittest3.fla`

```
for (var i:Number=0; i<=10; i++) {

  var unPion:Jeton = new Jeton();
  var uneZoneDeRelache:ZoneDeRelache = new ZoneDeRelache();

  unPion.addEventListener(MouseEvent.MOUSE_DOWN,pieceMobile);
```

```
    unPion.addEventListener(MouseEvent.MOUSE_UP,pieceFixe);

    unPion.x=50+(i*40);
    unPion.y=50;
    addChild(unPion);

    uneZoneDeRelache.x=50+(i*40);
    uneZoneDeRelache.y=150;
    addChild(uneZoneDeRelache);

    unPion.nomCible = uneZoneDeRelache;

}

function pieceMobile(evt:MouseEvent) {
    evt.currentTarget.startDrag();
}

function pieceFixe(evt:MouseEvent) {
    stopDrag();
    if (evt.currentTarget.hitTestObject(evt.currentTarget.nomCible)) {
        evt.currentTarget.x=evt.currentTarget.nomCible.x;
        evt.currentTarget.y=evt.currentTarget.nomCible.y;
        evt.currentTarget.removeEventListener(MouseEvent.MOUSE_DOWN,pieceMobile);
    }
}
```

La clé pour comprendre ce script réside dans les premières lignes et dans celle qui contient le code `unPion.nomCible = uneZoneDeRelache`.

Avant d'aller plus loin dans notre analyse, voici ce que le script ci-dessus nous permet d'obtenir sur la scène.

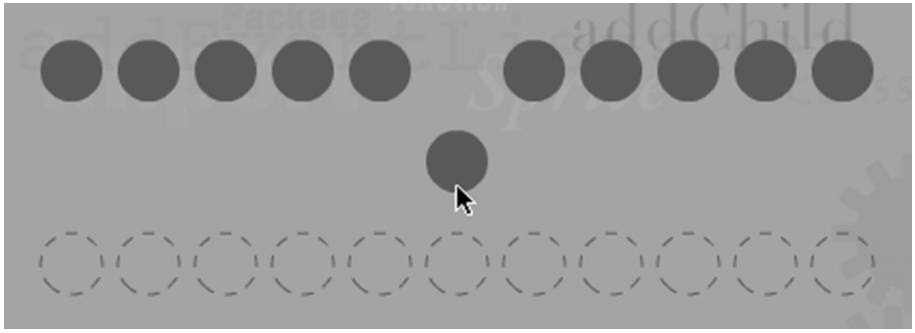


Figure 4-26

Toutes les occurrences ont été placées dynamiquement.

Pour placer ces 11 occurrences sur la scène, nous avons effectué une boucle et chargé dynamiquement deux symboles.

Rappelons que le chargement d'un symbole est effectué de la façon suivante :

1. Cliquez droit sur un symbole dans la bibliothèque, sélectionnez la commande Liaison… et cochez la case Exporter pour ActionScript (figure 4-27). Donnez alors un nom de classe qui ne contient ni caractères spéciaux ou accentués ni espace. Nous vous conseillons, en revanche, de placer une majuscule en début de nom.

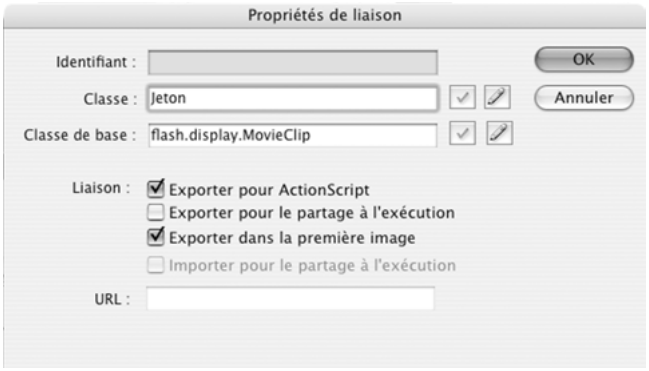


Figure 4-27

Pour placer un symbole sur la scène, il doit posséder un nom de classe.

2. Lorsque vous avez défini un nom de classe, il ne vous reste plus qu'à placer ce symbole tel que nous l'avons fait aux lignes 3 et 4.

Pourquoi avons-nous demandé l'exécution de la ligne `unPion.nomCible = uneZoneDeRelache` ?

Rappelons que l'objectif de notre animation est de positionner une occurrence à un emplacement précis. À chaque itération de la boucle, nous plaçons donc un symbole représentant le pion et un autre représentant l'emplacement de la dépose du pion. Un test de collision doit être effectué entre ces deux occurrences, mais elles ne possèdent pas de nom (en dehors de celui qui leur est donné par défaut : `instance1`, `instance2`, etc.). Nous définissons donc pour chaque occurrence de la classe `Pion()`, une propriété intitulée `nomCible` qui mémorise le nom par défaut de l'instance du symbole `zoneDeRelache`. Il ne nous reste alors plus qu'à effectuer un test de collision en faisant référence à `evt.currentTarget` dans la fonction de rappel et la propriété `nomCible` de `evt.currentTarget`.

En définissant, dans l'instance de la classe `Jeton`, une propriété qui contient la référence de l'occurrence à utiliser pour évaluer la collision, nous simplifions notre script, car nous n'avons ainsi pas besoin de gérer les noms des occurrences.

Gérer les plans entre deux ou plusieurs occurrences

Comme le montre très bien la copie d'écran de la figure 4-28, vos mises en pages présenteront parfois des chevauchements d'occurrences. Il est alors plus difficile de manipuler celles qui se trouvent en arrière-plan. Pour remédier à ce problème, vous pouvez faire appel à la fonction `setChildIndex()`.

Figure 4-28

Nous avons disposé trois cartes à jouer sur la scène, les unes sur les autres. Chacune d'entre elles va passer au premier plan si elle reçoit un clic.

Chaque occurrence contenue sur la scène ou dans un conteneur d'objets d'affichage possède un numéro d'index (un numéro de plan ou de niveau) qui définit son ordre d'affichage. Ainsi, lorsque vous placez une première occurrence sur la scène, l'index 0 lui est attribué. Si vous placez à présent une deuxième occurrence sur la scène (à cheval sur la première ou à côté), elle portera le numéro d'index 1 et se trouvera au premier plan.

Ni en ActionScript, ni à partir de l'interface de Flash, vous ne pourrez définir un numéro d'index qui ne soit pas consécutif à celui qui se trouve déjà au premier plan.

AS1/AS2

En ActionScript 1 et 2, nous pouvions définir un plan d'occurrence élevé, ce qui avait pour effet de placer l'instance au premier plan, mais aussi et surtout de générer des trous dans la liste des numéros d'index des occurrences sur la scène. Cela n'est plus possible en AS3.

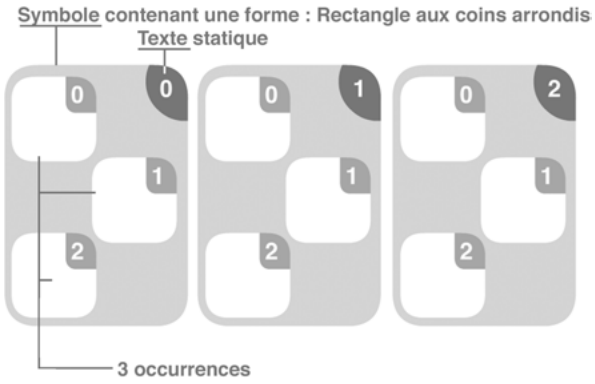
Indexation des instances dans un conteneur d'objets d'affichage

L'indexation des occurrences contenues dans un conteneur d'objets d'affichage redémarre toujours à 0.

Prenons l'exemple suivant pour mieux comprendre : imaginez que vous ayez trois conteneurs d'objets d'affichage sur la scène (par exemple des occurrences de Clip qui contiennent un rectangle gris aux coins arrondis). Vous placez, à l'intérieur de chacun d'entre eux, trois occurrences d'un symbole de votre bibliothèque ainsi qu'un texte statique.

Figure 4-29

L'index d'une occurrence est propre à chaque conteneur d'objets d'affichage.



Votre animation possédera donc trois occurrences ayant comme numéros d'index 0, 1 et 2 (les trois grands rectangles de la figure 4-29), mais chacune d'elles possédera des occurrences dont les numéros d'index débuteront à 0. Si nous cherchons à savoir quelle occurrence possède les index 0, 1, 2, 3 et 4, nous obtenons dans les trois cas le même résultat :

```
[object Shape]
[object StaticText]
[object MovieClip]
[object MovieClip]
[object MovieClip]
```

Pour cela, nous avons exécuté les scripts suivants :

```
for ( var i=0 ;i<grand1.numChildren;i++ ) {
  trace(grand1.getChildAt(i));
}
```

et

```
for ( var i=0 ;i<grand2.numChildren;i++ ) {
  trace(grand2.getChildAt(i));
}
```

et

```
for ( var i=0 ;i<grand3.numChildren;i++ ) {
  trace(grand3.getChildAt(i));
}
```

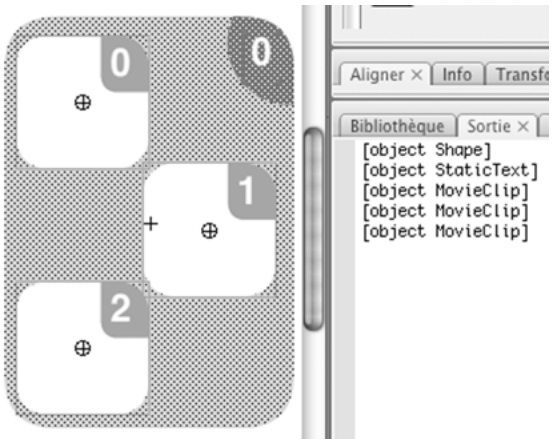


Figure 4-30
Voici le contenu du premier symbole.

Si vous ajoutez ou supprimez une occurrence contenue dans un conteneur d'objets d'affichage, la renumérotation des index des occurrences se produira automatiquement.

Connaître le nombre d'occurrences dans un conteneur d'objets d'affichage

Nous allons très rapidement avoir besoin de connaître le nombre d'occurrences contenues sur la scène ou dans un conteneur d'objets d'affichage ; vous allez donc devoir faire appel à la propriété `numChildren`. Son utilisation est très simple car il suffit de saisir uniquement ce terme avec le nom du conteneur en préfixe (aucun préfixe n'est nécessaire pour connaître le nombre d'occurrences contenues sur la scène). Effectuez une mise en pages à base d'occurrences, de formes et de textes puis saisissez la ligne d'instruction ci-dessous.

```
trace(numChildren);
```

Dans la fenêtre Sortie, vous découvrirez un nombre qui correspond au nombre d'occurrences contenues sur la scène.

> **Remarque**
>
> Toutes les formes tracées à l'aide des outils de dessin qui n'ont pas été transformées en symbole ne sont pas, par définition, des occurrences, mais elles sont tout de même comptabilisées.

Lorsque vous utiliserez la valeur renvoyée par cette propriété pour régler le niveau d'une occurrence, gardez toujours à l'esprit que le numéro d'index de la première occurrence sur la scène ou dans un conteneur d'objets d'affichage porte le numéro 0. De ce fait, l'occurrence qui se trouve au premier plan portera, par exemple, le numéro d'index 6 s'il y a sept occurrences.

Pour placer une occurrence au premier plan sur la scène, il faut donc utiliser la ligne d'instruction suivante :

```
setChildIndex(nomOccurrence,numChildren-1);
```

La méthode setChildIndex()

Revenons sur le fonctionnement de la méthode `setChildIndex()`, qui est extrêmement simple (il suffit de préciser le nom de l'objet d'affichage que vous souhaitez placer sur la scène, comme une occurrence de type Clip, ainsi que son niveau, défini par un indice).

```
setChildIndex(nomOccurrence,3);
```

Pour information, lorsque vous placez un symbole sur la scène, que ce soit manuellement par un glisser-déposer ou dynamiquement à l'aide de lignes d'instructions en ActionScript, la gestion des plans est automatique. La dernière occurrence placée sur la scène vient toujours se placer au premier plan, au-dessus de toutes les autres.

À la lecture de l'unique ligne d'instruction ci-dessus, nous serions tentés de croire que cette action est simple mais, dans certains cas, elle peut s'avérer plus complexe. Nous aborderons ce problème dans le développement suivant, Cibler une occurrence cliquée.

Concentrons-nous d'abord sur un premier script qui permet de placer une carte à jouer au premier plan. Trois occurrences intitulées `bt1`, `bt2` et `bt3` ont été placées sur la scène (sous

les trois cartes de la figure 4-28) et lorsque nous cliquons sur l'une d'entre elles, elle se place au premier plan.

Fichier de référence : `Chapitre4/gestionplans1.fla`

```
bt1.addEventListener(MouseEvent.MOUSE_DOWN,premierPlan1);
bt2.addEventListener(MouseEvent.MOUSE_DOWN,premierPlan2);
bt3.addEventListener(MouseEvent.MOUSE_DOWN,premierPlan3);

function premierPlan1(evt:MouseEvent) {
  setChildIndex(carte1,numChildren-1);
}
function premierPlan2(evt:MouseEvent) {
  setChildIndex(carte2,numChildren-1);
}
function premierPlan3(evt:MouseEvent) {
  setChildIndex(carte3,numChildren-1);
}
```

Comme vous pouvez le constater, ce script est simple, mais il pourrait s'avérer très rapidement fastidieux à écrire si nous avions davantage d'occurrences à traiter.

Passage au premier plan ou changement de plan d'une occurrence

Que se produit-il lorsqu'une occurrence passe au premier plan ou change de plan ? La réponse à cette question est extrêmement importante et nous permet d'illustrer la réindexation automatique de toutes les occurrences d'un conteneur. L'animation suivante vous permet de vérifier en images ce comportement.

Fichier de référence : `Chapitre4/gestionplans2.fla`

Aidez-vous de cette animation pour vérifier en image les propos suivants :

Sept occurrences se trouvent sur la scène. Elles sont indexées par défaut de 0 à 6. Nous décidons de placer l'occurrence portant l'index 1 au niveau 4, en écrivant l'appel `setChildIndex(carte1,4)`. Les index de chaque occurrence avant et après le changement de niveau sont présentés dans le tableau 4-2.

Tableau 4-2 Index de chaque occurrence avant et après le déplacement de l'occurrence Carte1 au niveau 4

Occurrence	Index avant le changement de niveau	Index après le changement de niveau
Carte0	0	0
Carte1	1	4
Carte2	2	1
Carte3	3	2
Carte4	4	3
Carte5	5	5
Carte6	6	6

Après le déplacement de l'occurrence `Carte1` au niveau 4 :

- L'occurrence qui se trouvait avant l'index 1 n'a pas changé de place.
- Les occurrences dont l'index était supérieur à 4 n'ont pas été modifiées.

Les occurrences situées entre les index 2 et 4 ont été déplacées d'un niveau, comblant ainsi l'espace laissé vide dans la liste d'indexation par l'occurrence `Carte1`.

À titre d'exemple, le tableau 4-3 présente les index de chaque occurrence avant et après le déplacement de l'occurrence `Carte4` au niveau 1.

Tableau 4-3 Index de chaque occurrence avant et après le déplacement de l'occurrence Carte4 au niveau 1

Occurrence	Index avant le changement de niveau	Index après le changement de niveau
Carte0	0	0
Carte1	1	2
Carte2	2	3
Carte3	3	4
Carte4	4	1
Carte5	5	5
Carte6	6	6

Cibler une occurrence cliquée

Comme nous l'évoquions précédemment, il est parfois difficile de faire référence au nom d'un objet d'affichage lors de l'appel à la méthode `setChildIndex()`.

Considérons le script ci-dessous et rappelons que le paramètre `evt.currentTarget` permet de faire référence à l'occurrence cliquée.

```
curseur.addEventListener(MouseEvent.CLICK,deplacerDroite);
function deplacerDroite(evt:Event) {
  evt.currentTarget.x+=3;
}
```

Dans un premier temps, adaptons ce script pour pouvoir utiliser la méthode `setChildIndex()` sans `evt.currentTarget` :

```
curseur.addEventListener(MouseEvent.CLICK,deplacerDroite);
function deplacerDroite(evt:Event) {
  setChildIndex(curseur,3);
}
```

Dans cet exemple, l'occurrence intitulée `curseur` vient se placer au niveau de l'index 3 comme le montre la figure 4-31. Nous avons fait clairement référence au nom d'occurrence `curseur`, mais pas à `evt.currentTarget`.

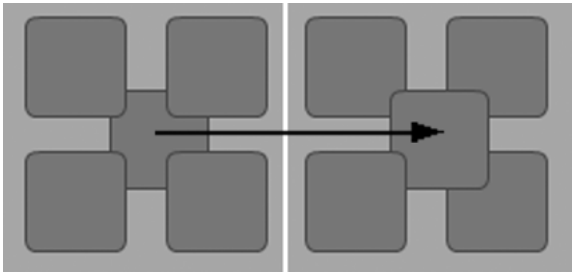


Figure 4-31

L'occurrence située en arrière-plan s'intitule curseur et se trouve à l'index 0 avant l'exécution du script. Elle se situe à l'index 3 après l'exécution du script.

Reprenons à présent le code initial en faisant appel à l'objet `evt.currentTarget` pour ne pas avoir à faire référence au nom d'une occurrence et ainsi pouvoir réutiliser la fonction de rappel.

```
curseur.addEventListener(MouseEvent.CLICK,deplacerDroite);
function deplacerDroite(evt:Event) {
  setChildIndex(evt.currentTarget,3);
}
```

À la publication de l'animation, nous obtenons un message d'erreur !

```
1118: Contrainte implicite d'une valeur du type statique Object vers un type
peut-être sans rapport flash.display:DisplayObject.
```

Ce message d'erreur signifie que `evt.currentTarget` n'est pas un `DisplayObject`, c'est-à-dire un type d'objet auquel s'attend la méthode `setChildIndex()`. Nous devons donc modifier notre code de la façon suivante :

```
curseur.addEventListener(MouseEvent.CLICK,deplacerDroite);
function deplacerDroite(evt:Event) {
  setChildIndex(DisplayObject(evt.currentTarget),3);
}
```

L'ajout de la fonction `DisplayObject()` permet de convertir le type d'un objet en `DisplayObject`. Elle est donc nécessaire pour combiner `evt.currentTarget` et la méthode `setChildIndex()`.

Information

Essayez de tracer `evt.currentTarget` en exécutant la ligne d'instruction `trace(evt.currentTarget)` et vous découvrirez qu'il s'agit d'un objet de type MovieClip. Il est surprenant de découvrir qu'un MovieClip n'est pas, par défaut, un objet d'affichage. Rappelons peut-être qu'un MovieClip possède une classe éponyme qui hérite de la classe `DisplayObjectContainer` (qui hérite elle-même de la classe `DisplayObject`).

Voici un dernier exemple qui met en évidence l'aspect pratique de ne disposer que d'une seule fonction pour plusieurs objets d'écoute.

Fichier de référence : `Chapitre4/gestionsplans3.fla`

```
carte1.addEventListener(MouseEvent.MOUSE_DOWN,premierPlan);
carte2.addEventListener(MouseEvent.MOUSE_DOWN,premierPlan);
carte3.addEventListener(MouseEvent.MOUSE_DOWN,premierPlan);

function premierPlan(evt:MouseEvent) {
  setChildIndex(DisplayObjectContainer(evt.currentTarget),numChildren-1);
}
```

Remarque

Attention : afin de mieux pouvoir gérer les plans des occurrences, il est fortement conseillé de les placer dans des instances de conteneur telles que des Sprite, par exemple. Consultez le chapitre 2 pour mieux comprendre ce dernier propos.

Connaître le numéro d'index d'une occurrence

Fichier de référence : `Chapitre4/gestionsplans4.fla`

Vous aurez parfois besoin de connaître le numéro d'index d'une occurrence afin de pouvoir la manipuler. Vous devrez alors utiliser la méthode `getChildIndex()` en précisant un nom d'objet d'affichage en paramètre.

L'exemple ci-dessous permet d'afficher sur la scène le numéro d'index de l'occurrence sur laquelle on clique. L'affichage s'effectue dans un texte dont le nom d'instance est `numeroIndex`.

```
carte1.addEventListener(MouseEvent.MOUSE_DOWN,premierPlan);
carte2.addEventListener(MouseEvent.MOUSE_DOWN,premierPlan);
carte3.addEventListener(MouseEvent.MOUSE_DOWN,premierPlan);

function premierPlan(evt:MouseEvent) {
  numeroIndex.text = getChildIndex(DisplayObject(evt.currentTarget)).toString();
}
```

Vous observerez que la méthode `getChildIndex()` s'attend également à un paramètre de type `DisplayObject`. Nous devons donc faire appel à la méthode `DisplayObject()` pour convertir le type de `evt.currentTarget`.

Faire référence à une occurrence à partir de son index

Rappelons tout d'abord que pour obtenir le nom d'une occurrence sur laquelle vous cliquez, il suffit de faire référence à la propriété `name` accessible à partir de la propriété `currentTarget`.

```
carte1.addEventListener(MouseEvent.MOUSE_DOWN,premierPlan);
function premierPlan(evt:MouseEvent) {
  trace(evt.currentTarget.name);
}
```

Dans certains cas, vous aurez besoin de préciser le numéro d'index pour rechercher ou faire référence à une occurrence ; vous utiliserez alors la méthode `getChildAt()` en précisant un index.

Faire référence au nom d'une occurrence par concaténation

Lorsque vous souhaiterez faire référence à des occurrences par concaténation d'une chaîne de caractères et d'un nombre croissant, vous utiliserez la méthode `getChildByName()`. Dans l'exemple suivant, nous réglons la transparence de plusieurs occurrences, intitulées `carte1` à `carte10`, à l'aide d'une boucle en faisant appel à la méthode `getChildByName()`.

Fichier de référence : `Chapitre4/gestionsplans5.fla`

```
for (var i:Number=1; i<=10; i++) {
  getChildByName("carte"+i).alpha = 0.5;
}
```

Le script est très court car notre besoin est simple, mais si vous ne comprenez pas cette structure de code, consultez le chapitre 10 de ce livre qui traite de la boucle `for()`.

Désactiver la détection d'événement sur une occurrence

Fichier de référence : `Chapitre4/desactiveroccurrence1.fla`

Lorsque vous souhaitez qu'une occurrence ne réagisse plus aux événements de la souris, vous disposez de deux solutions :

- Désactiver temporairement la capacité de cette occurrence à déceler un événement de type souris avec la propriété `mouseEnabled`.
- Annuler le gestionnaire d'événement qui lui a été attribué avec la méthode `removeEventListener()`.

Abordons tout de suite un premier exemple qui montre comment rendre deux occurrences inactives après un clic sur un interrupteur.

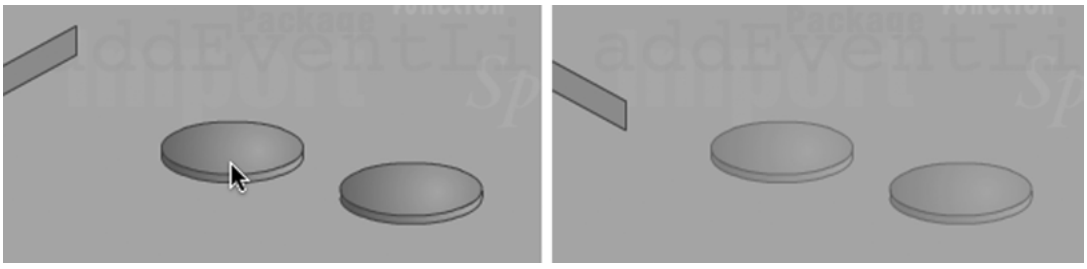


Figure 4-32

Les deux occurrences de la copie d'écran de droite ne peuvent plus être saisies pour être déplacées car l'interrupteur, à gauche de la scène, a été abaissé.

Nous avons placé sur la scène un interrupteur représenté par un rectangle incliné et deux pions. Ces derniers peuvent être déplacés par glisser-déposer.

Si vous cliquez sur l'interrupteur, son échelle verticale (`scaleY`) bascule à -1, donnant ainsi l'impression qu'il s'abaisse. Par la même occasion, nous rendons les pions insaisissables et légèrement transparents. Si vous cliquez à nouveau sur l'interrupteur, les pions redeviennent opaques et mobiles.

```
declencheur.addEventListener(MouseEvent.CLICK,desactiverPions);

function desactiverPions(evt:MouseEvent) {
  evt.currentTarget.scaleY*=-1;
  if (evt.currentTarget.scaleY==-1) {
    pion1.mouseEnabled = false;
    pion1.alpha = 0.5;
    pion2.mouseEnabled = false;
    pion2.alpha = 0.5;
  } else {
    pion1.mouseEnabled = true;
    pion1.alpha = 1;
    pion2.mouseEnabled = true;
    pion2.alpha = 1;
  }
}

pion1.addEventListener(MouseEvent.MOUSE_DOWN,deplacerPion);
pion2.addEventListener(MouseEvent.MOUSE_DOWN,deplacerPion);
pion1.addEventListener(MouseEvent.MOUSE_UP,fixerPion);
pion2.addEventListener(MouseEvent.MOUSE_UP,fixerPion);

function deplacerPion(evt:MouseEvent) {
  evt.currentTarget.startDrag();
}

function fixerPion(evt:MouseEvent) {
  stopDrag();
}
```

Remarque

Si vous aviez davantage d'occurrences à gérer, il serait préférable d'utiliser une boucle `for()` pour placer les pions sur la scène et d'écrire ensuite les scripts de leurs comportements (lignes contenues dans l'instruction `if()`).

Lorsque nous réglons la propriété `mouseEnabled` à `false` pour une occurrence donnée, les événements souris se rapportant à cette dernière ne peuvent plus être détectés.

Dans l'exemple suivant, nous souhaitons qu'une occurrence ne puisse être déplacée qu'une seule fois.

Fichier de référence : `Chapitre4/desactiveroccurrence2.fla`

```
pion1.addEventListener(MouseEvent.MOUSE_DOWN,deplacerPion);
pion1.addEventListener(MouseEvent.MOUSE_UP,fixerPion);

function deplacerPion(evt:MouseEvent) {
  evt.currentTarget.startDrag();
  evt.currentTarget.removeEventListener(MouseEvent.MOUSE_DOWN,deplacerPion);
}

function fixerPion(evt:MouseEvent) {
  stopDrag();
  evt.currentTarget.removeEventListener(MouseEvent.MOUSE_UP,fixerPion);
}
```

Pour interdire un deuxième déplacement éventuel, nous devons tout simplement annuler les gestionnaires d'événements. Nous commençons par appeler la fonction `removeEventListener()` pour l'événement `MOUSE_DOWN` si l'utilisateur clique sur l'occurrence. Au moment où il relâche le bouton de la souris après un déplacement de l'occurrence, nous appelons à nouveau la méthode `removeEventListener()` pour annuler également l'événement `MOUSE_UP`. Précisons, tout de même, que si ces occurrences doivent être à nouveau mobiles, il est tout à fait possible de leur redéfinir un écouteur en faisant appel à la méthode `addEventListener()`.

Déplacer la tête de lecture du scénario

Fichier de référence : `Chapitre4/deplacerTete.fla`

Il existe une problématique récurrente lorsqu'une personne doit réaliser ses premières animations en Flash (soulevée dans le chapitre 15) : quelle méthode faut-il utiliser pour afficher différents contenus sur la scène d'une animation, en réaction à un événement de type souris ou clavier, déclenché par l'utilisateur ? Il existe de nombreuses techniques. L'une d'entre elles, qui ne présente pas de réelle difficulté, mais qui est aussi la moins dynamique, consiste à déplacer la tête de lecture du scénario sur des images-clés.

La construction des différents écrans d'une animation peut en effet être réalisée très simplement à partir de plusieurs images disposées le long du scénario. Il suffit ensuite de programmer des boutons pour qu'ils réagissent au clic et déclenchent un déplacement de la tête de lecture. Une telle technique signifie donc que la mise en forme des écrans passe par une manipulation intensive des outils de Flash et de la Timeline (le scénario). Dans le cas d'une animation ayant un contenu très important, cette méthode s'avère peu efficace et fait perdre beaucoup de temps.

Dans la copie d'écran de la figure 4-33, vous découvrez une animation avec de nombreuses images (et images-clés). Chacune d'entre elles possède un contenu très différent d'une image à l'autre. Cette technique est conseillée aux personnes qui ne savent pas très bien programmer ou ne le souhaitent pas.

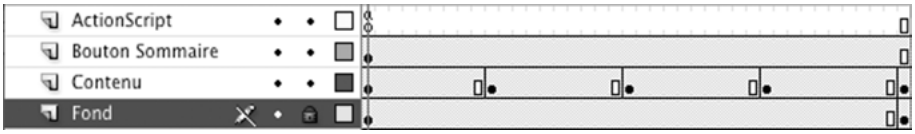


Figure 4-33

L'animation correspondant à cette copie d'écran possède cinq écrans. Un sommaire (figure 4-34) et quatre autres images (figure 4-35).

Une telle construction à partir du scénario se conduit très facilement car il suffit de connaître la technique de création d'une image-clé (une simple pression sur la touche de fonction F6 du clavier) pour pouvoir construire autant d'écrans que vous le désirez.

Arrêter la tête de lecture

Si vous construisez une animation comme celle que nous vous proposons au téléchargement (`deplacerTete.fla`) ou qui présente au moins deux images-clés, vous devrez bloquer la tête de lecture afin qu'elle s'arrête sur l'image 1 de l'animation qui contient le sommaire (figures 4-33 et 4-34).

> **Note**
>
> Le sommaire d'une animation ne se trouve pas obligatoirement sur l'image 1, notamment lorsque vous utilisez une séquence animée d'introduction.

Dans ce cas, placez l'instruction `stop()` sur l'image-clé où vous souhaitez voir cesser la lecture. Cette fonction bloquera la tête de lecture lorsque celle-ci rencontrera l'image-clé.

Déplacer la tête de lecture dans le scénario d'une animation

Pour illustrer cette technique, nous avons réalisé une animation (`deplacerTete.fla`) qui met en évidence l'utilisation de la méthode `gotoAndStop()` et qui permet de placer la tête de lecture sur une image précise du scénario.

L'image-clé 1 de notre animation contient, sur la scène, les éléments de la copie d'écran de la figure 4-34 : un titre qui introduit le sujet de l'animation (La couleur) et cinq boutons de navigation. Quatre d'entre eux vont nous mener aux différents écrans contenus aux images-clés 10, 20, 30 et 40, et le cinquième (le bouton Sommaire) nous ramènera à l'image 1.

Figure 4-34

Écran du sommaire de l'animation deplacerTete.fla

Figure 4-35

L'un des quatre écrans (hormis celui du sommaire) de l'animation deplacerTete.fla.

Voici l'unique script qui se trouve sur l'image-clé 1 : il décrit les actions à réaliser en réaction aux différents clics effectués sur les boutons de l'animation.

```
stop();

btRVB.image = 10;
btCMJN.image = 20;
btHexa.image=30;
btTSL.image=40;
```

```
btRVB.addEventListener(MouseEvent.CLICK,deplacerTete);
btCMJN.addEventListener(MouseEvent.CLICK,deplacerTete);
btHexa.addEventListener(MouseEvent.CLICK,deplacerTete);
btTSL.addEventListener(MouseEvent.CLICK,deplacerTete);

function deplacerTete(evt:Event) {
  gotoAndStop(evt.currentTarget.image);
  btSommaire.visible=true;
}
//
btSommaire.visible=false;
btSommaire.addEventListener(MouseEvent.CLICK,retourSommaire);

function retourSommaire(evt:MouseEvent) {
  gotoAndStop(1);
}
```

L'analyse de ce script est simple, mais néanmoins intéressante sur un point précis par lequel nous terminerons nos explications.

- La fonction `stop()` sert à bloquer la tête de lecture sur l'image 1, celle du sommaire.
- Cinq écouteurs permettent de déceler le clic sur les occurrences que nous évoquions précédemment.
- L'appel à la propriété `visible` sert à masquer le bouton Sommaire sur la première image-clé de l'animation.
- La fonction `deplacerTete()`, qui est chargée de déplacer la tête de lecture sur les images 10, 20, 30 et 40, contient l'instruction `btSommaire.visible=true` pour afficher cette occurrence lorsque l'utilisateur quitte l'écran du Sommaire.

Attardons-nous à présent sur la présence du mot `image` dans notre script.

Il s'agit d'une propriété que nous ajoutons dynamiquement à chaque instance de bouton de navigation (`btRVB`, `btCMJN`, `btHexa` et `btTSL`). De cette façon, nous stockons une valeur que nous rappelons dans la fonction `deplacerTete()` en faisant référence à la propriété `gotoAndStop(evt.currentTarget.image)`. Cette valeur correspond au numéro d'image vers laquelle la tête de lecture doit se déplacer.

> **Remarque**
>
> Il est en effet impossible de passer un argument à une fonction de rappel. L'appel à une propriété rattachée à une occurrence de clip permet donc de pallier ce problème.

Les autres méthodes

Nous avons, jusqu'à présent, utilisé les méthodes `gotoAndStop()` et `stop()` pour contrôler la position de la tête de lecture. Voici quelques méthodes supplémentaires qui proposent une navigation différente au sein d'une animation.

`play()` : fonction qui permet de relancer la lecture préalablement interrompue avec la méthode `stop()` ou `gotoAndStop()`.

`gotoAndPlay()` : fonction qui déplace la tête de lecture sur une image indiquée en paramètre (entre les parenthèses de la fonction) et relance la lecture. Exemple : `gotoAndPlay(20)`.

`nextFrame()` : fonction qui déplace la tête de lecture sur l'image suivante.

`prevFrame()` : fonction qui déplace la tête de lecture sur l'image précédente.

Déplacer la tête de lecture d'un clip

Fichier de référence : `Chapitre4/deplacerTete2.fla`

Observez bien la suite d'images de la figure 4-37 : il s'agit des images-clés contenues dans un symbole de type clip que nous avons disposé sur la scène. L'occurrence obtenue a d'ailleurs été nommée, à cette occasion, `plante`.

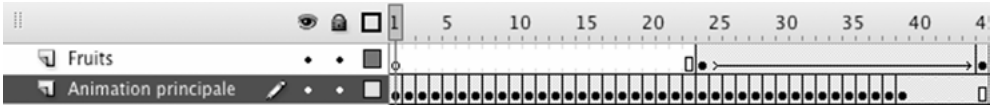


Figure 4-36
Scénario d'un clip

Figure 4-37
Cette animation résulte d'une succession d'images-clés contenues dans le clip que nous évoquions à la figure 4-36.

Rappelons qu'un clip possède un scénario interne qui peut contenir des images et des images-clés utilisées alors pour générer une animation. Toutefois, il sera parfois plus pratique de réaliser une animation dans un symbole plutôt que de la créer sur le scénario principal d'une animation.

Dans cet exemple, nous avons tout d'abord réalisé une animation dans un clip. Nous souhaiterions maintenant la contrôler pour pouvoir visualiser chaque image, l'une après l'autre, dans un sens ou un autre. Nous aurions pu commencer par placer la fonction `stop()` sur la première image-clé du clip afin que la lecture de son occurrence ne soit pas automatique, mais nous l'avons fait dans le script principal, sur la première image-clé de l'animation, en faisant référence au nom de l'occurrence `plante` : `plante.stop()`. Au lancement de l'animation par le raccourci Ctrl-Entrée (Windows) ou Commande-Entrée (Mac), le clip est donc figé sur la première image-clé de son scénario. Il ne nous reste plus qu'à programmer deux occurrences de clip pour contrôler la position de la tête de lecture.

Les deux occurrences, intitulées `btPlusTot` et `btPlusTard`, qui ont été placées sur la scène sont utilisées pour déplacer la tête de lecture de l'occurrence qui contient notre animation constituées d'images-clés (figure 4-36). Voici le script qui les gère.

```
plante.stop();

btPlusTot.addEventListener(MouseEvent.CLICK,reculerTete);
btPlusTard.addEventListener(MouseEvent.CLICK,avancerTete);


function reculerTete(evt:MouseEvent) {
  plante.prevFrame();
}
function avancerTete(evt:MouseEvent) {
  plante.nextFrame();
}
```

Comme expliqué précédemment, la première instruction bloque la tête de lecture du scénario du clip qui contient l'animation.

Ensuite, deux gestionnaires, accompagnés de leurs écouteurs, ont pour fonction de déplacer la tête de lecture sur l'image suivante ou précédente.

Remarque

La fonction est bien `prevFrame()` et non `previousFrame()`.

Si nous souhaitions relancer la lecture du clip, il faudrait utiliser l'instruction suivante :

```
plante.play();
```

Note

Les fonctions `gotoAndStop()` et `gotoAndPlay()` fonctionnent également sous la même condition, c'est-à-dire en les préfixant du nom de l'occurrence `plante.gotoAndStop`.

5

Les mouvements d'une occurrence sur la scène

Flash est avant tout un logiciel d'animation. La programmation a été implémentée dans l'IDE de Flash progressivement, et il est donc tout à fait naturel que nous puissions écrire des scripts d'animation d'occurrences.

D'un autre côté, il est tout à fait possible de réaliser des interpolations dans Flash, sans recourir à la moindre ligne de code. C'est ce que font les utilisateurs qui ne possèdent pas de notions de programmation.

Quels sont donc les avantages de réaliser une animation à partir d'un programme écrit en ActionScript ? La réponse est simple : une animation issue d'un script sera plus facile à configurer et à modifier. De plus, il sera parfois impossible de réaliser des animations avec un mouvement précis à partir du scénario, donc sans programmation.

Pourquoi alors existe-t-il autant de techniques pour animer une occurrence ?

La réponse est encore une fois très simple : tout dépend de ce que vous souhaitez animer.

Si vous devez simplement faire tourner une occurrence en continu, sans contrôler le lancement de cette animation, ni même son arrêt, vous pouvez utiliser l'événement `ENTER_FRAME`. Par ailleurs, si vous disposez déjà d'un tel événement dans un script, pourquoi faire appel à un autre gestionnaire, alors que celui-ci peut recevoir une ligne d'instruction supplémentaire chargée de faire tourner l'occurrence.

En revanche, dans certains cas, il sera indispensable de contrôler le lancement et/ou l'arrêt d'un mouvement d'occurrence. Vous devrez alors utiliser la classe `Timer()` qui possède des méthodes facilitant ces opérations.

Troisième et dernière question : dans quel cas privilégier l'utilisation de la classe `Tween()` alors que des propriétés d'occurrences combinées avec la classe `Timer()` et/ou l'événement `ENTER_FRAME` aboutissent au même résultat ?

La réponse à apporter à cette interrogation est plus longue car elle nécessite de développer certains arguments.

Vous allez découvrir que vous pouvez programmer une interpolation de différentes façons. L'utilisation de l'événement `ENTER_FRAME` est plus facile à comprendre et à maîtriser en comparaison de la classe `Tween()`, mais cette dernière est plus précise pour configurer un mouvement. Prenons l'exemple des variations de vitesse dans une animation : pour donner une impression d'élasticité ou de rebond, il faudrait écrire de nombreuses lignes de code en utilisant la première solution, alors que la classe `Tween()` constitue une solution plus efficace.

Par ailleurs, lorsqu'un mouvement doit être mis en pause ou reproduit dans le sens inverse, la classe `Tween()` propose des méthodes prêtes à l'emploi, alors qu'avec l'événement `ENTER_FRAME` et l'appel de propriétés d'occurrences, tout doit être programmé.

Nous pourrions encore vous exposer plus longuement les avantages de la classe `Tween()`, mais vous les découvrirez dans les exemples de ce chapitre, qui présente l'ensemble des solutions disponibles pour gérer les mouvements d'une occurrence sur la scène.

Utilisation de l'événement ENTER_FRAME

Dans la mesure où la classe `Tween()` s'avère plus puissante que l'utilisation de l'événement `ENTER_FRAME`, pourquoi faire appel à ce dernier dans un gestionnaire ?

Dans certains cas, vous disposerez déjà d'une fonction appelée par un écouteur faisant référence à cet événement. Vous pourrez alors ajouter facilement les quelques lignes de codes nécessaires pour obtenir un mouvement d'occurrence. Par ailleurs, rappelons que tout le monde n'a pas le même niveau en programmation : si la classe `Tween()` vous paraît complexe à employer, l'événement `ENTER_FRAME` dans un gestionnaire reste une solution accessible.

Fichier de référence : `Chapitre5/enterframe.fla`

Commençons par un premier exemple, celui d'une balle qui se déplace de la gauche vers la droite de la scène, à vitesse constante.

```
balle1.addEventListener(Event.ENTER_FRAME,deplacerBalle1);

function deplacerBalle1(evt:Event) {
  balle1.x+=1.5;
}
```

Ce script devrait vous être familier car il a été utilisé dans le chapitre 3 dédié à la gestion des événements. Nous allons maintenant étudier comment faire varier la vitesse de déplacement d'une occurrence.

Ralentir un mouvement

Dans l'exemple précédent, où le mouvement s'effectue à vitesse constante, le pas est fixé à une valeur de 1,5 pixel. Pour ralentir un mouvement, il suffit alors de faire varier le pas de déplacement de l'occurrence.

Une solution simple consiste à déplacer l'occurrence d'une valeur proportionnelle à la distance qu'il lui reste à parcourir. Par exemple, avec un coefficient de proportionnalité de 10 %, s'il faut la déplacer de 200 pixels, on commence par ajouter 20 pixels à sa position (10 % de 200). Il reste alors 180 pixels à parcourir. Au deuxième passage, on ajoutera 18 pixels à sa position (10 % de 180) et ainsi de suite, jusqu'à ce qu'il ne reste que quelques pixels avant la position définitive de l'occurrence. Le script suivant met en œuvre ce principe.

```
balle2.addEventListener(Event.ENTER_FRAME,deplacerBalle2);

function deplacerBalle2(evt:Event) {
  balle2.x+=(480-balle2.x)*0.07;
}
```

Le calcul `480-balle2.x` représente la distance que l'occurrence doit encore parcourir avant d'atteindre la position située à 480 pixels du bord gauche de l'écran. Nous déplaçons ensuite l'occurrence de 7 % de la distance restant à parcourir.

Pour être optimisé, ce script devrait contenir un test (dans la fonction `deplacerBalle()`) qui vérifierait si la distance restant à parcourir est inférieure à 1 pixel, pour éviter au lecteur Flash de calculer des valeurs proches de 0.

```
if(480-balle2.x<1){
  balle2.removeEventListener(Event.ENTER_FRAME,deplacerBalle2);
  balle2.x=480;
}
```

Cette technique présente tout de même l'avantage de pouvoir modifier facilement le pourcentage, et ainsi d'augmenter ou de diminuer la vitesse de déplacement. Il est également très facile de déployer une telle méthode.

Accélérer un mouvement

De façon symétrique au ralentissement, il suffit d'augmenter le pas de déplacement d'une occurrence pour obtenir un mouvement accéléré.

Nous allons incrémenter progressivement une variable.

```
balle3.addEventListener(Event.ENTER_FRAME,deplacerBalle3);
var pasInitial:Number = 0;
function deplacerBalle3(evt:Event) {
  pasInitial+=0.1;
  balle3.x+=pasInitial;
}
```

Les deux paramètres que vous pouvez modifier pour obtenir un résultat différent sont :

- `pasInitial` : définissez une valeur différente de `0` pour obtenir une occurrence déjà en mouvement au moment de l'appel au gestionnaire. Si vous définissez une valeur initiale négative, vous obtiendrez un recul de l'occurrence avant qu'elle ne se mette à avancer.
- `pasInitial+=0.1` : en augmentant ou en diminuant la valeur `0.1`, vous obtiendrez un mouvement plus ou moins accéléré.

Cette technique présente l'avantage d'être simple à déployer, mais il est conseillé d'ajouter un test conditionnel pour bloquer la vitesse lorsqu'elle atteint une certaine valeur.

```
function deplacerBalle3(evt:Event) {
  pasInitial+=0.1;
  if(pasInitial>3) pasInitial = 3;
  balle3.x+=pasInitial;
}
```

Pour optimiser notre code et ne pas exécuter en continu un test inutile, il serait préférable d'utiliser cette fonction :

```
function deplacerBalle3(evt:Event) {
  pasInitial+=0.1;
  if(pasInitial>3) {
    balle3.removeEventListener(Event.ENTER_FRAME,deplacerBalle3);
    balle3.addEventListener(Event.ENTER_FRAME,deplacerBalle3Bis);
  }
  balle3.x+=pasInitial;
}
```

Il ne vous resterait plus qu'à redéfinir une fonction intitulée `deplacerBalle3Bis` qui ne contiendrait qu'une seule ligne d'instruction : `balle3.x+=pasInitial`.

En modifiant le script initial, comme vous le propose l'exemple ci-dessous, vous obtiendrez un mouvement saccadé :

```
function deplacerBalle3(evt:Event) {
  pasInitial+=0.1;
  if(pasInitial>3) pasInitial = 0.5;
  balle3.x+=pasInitial;
}
```

Saccader un mouvement

Le script ci-dessous permet d'obtenir un mouvement saccadé, de façon plus ou moins analogue à celle que nous venons de voir à la fin du développement précédent, mais nous faisons appel à une fonction mathématique plus complexe. Référez-vous éventuellement à la section Utilisation des fonctions `Math.sin()` et `Math.cos()` de ce chapitre pour une explication plus détaillée.

```
balle4.addEventListener(Event.ENTER_FRAME,deplacerBalle4);
var coef:Number = 0;
function deplacerBalle4(evt:Event) {
  coef+=0.07;
  balle4.x+=Math.abs(Math.cos(coef))*2;
}
```

En modifiant la valeur d'incrémentation de la variable `coef`, vous pourrez augmenter ou ralentir le mouvement. La valeur `2` de la dernière ligne permet, elle, d'allonger la distance entre deux saccades.

Obtenir un mouvement sinueux

Considérons le script suivant.

```
balle5.addEventListener(Event.ENTER_FRAME,deplacerBalle5);
var valeur:Number = 0;
function deplacerBalle5(evt:Event) {
  valeur+=0.1;
  balle5.x+=2;
  balle5.y=227.8+Math.cos(valeur)*15;
}
```

En modifiant la valeur d'incrémentation de la variable `valeur` (par exemple `0.3` au lieu de `0.1`), vous pourrez augmenter le nombre d'oscillations verticales. Plus cette valeur tendra vers 0, plus l'ondulation obtenue sera allongée.

La dernière valeur de ce script (`15`) permet de définir l'amplitude de l'oscillation. Plus cette valeur sera grande, plus les déplacements verticaux de l'occurrence seront importants.

Utilisation de la classe Tween()

Quelques repères

Nous allons utiliser à plusieurs reprises différents termes qui font tous référence à la même notion : la programmation d'une interpolation de mouvement. Nous parlerons donc de tweening, d'interpolation, ou bien encore de tween.

Nous allons découvrir, dans cette partie du livre, qu'il est possible de réaliser des animations d'occurrences avec peu de code. En revanche, la teneur des lignes d'instructions est plus abstraite en comparaison des scripts que nous avons eu à gérer jusqu'à présent.

Avant de vous lancer dans la rédaction de vos premières lignes d'instructions, vous devez être conscient de la nécessité d'importer les classes `easing` et `transitions`. Sans l'exécution préalable des deux lignes ci-dessous, vous ne pourrez pas faire appel à la classe `Tween()`, car le lecteur Flash ne les intègre pas par défaut.

Membres d'une classe

Il s'agit des différentes méthodes et propriétés qui appartiennent à une classe.

```
import fl.transitions.easing.*;
import fl.transitions.*;
```

Par ailleurs, notez que toutes les propriétés avec lesquelles vous avez l'habitude de travailler pour manipuler vos occurrences vont pouvoir être utilisées dans une animation basée sur la classe `Tween()` (propriétés `x`, `y`, `scaleX`, `scaleY`, `width`, `height`, `alpha` et `rotation`).

Découvrons à présent l'unique ligne d'instruction nécessaire au déroulement d'un menu.

```
new Tween(balle,"y",Regular.easeOut,20,180,2,true);
```

Au premier abord, cette ligne ne ressemble pas à celle que nous pourrions utiliser pour régler une simple propriété d'occurrence, mais rassurez-vous, toutes les parties de cette instruction sont très explicites.

Voici le rôle de chaque paramètre placé à l'intérieur des parenthèses :

- `balle` : le nom de l'occurrence concernée par l'interpolation à base de la classe `Tween()`.
- `"y"` : nom de la propriété qui va être utilisée pour réaliser l'interpolation. Rappelons que vous pouvez utiliser les propriétés `x`, `y`, `scaleX`, `scaleY`, `width`, `height`, `alpha` et `rotation`.
- `Regular.easeOut` : mode d'animation pour obtenir un effet. Nous reviendrons sur ce paramètre après les explications relatives aux paramètres de la classe `Tween()`.
- `20` : valeur de départ pour l'interpolation de mouvement à base de la classe `Tween()`. Dans notre exemple, cette valeur correspond notamment à la position du point d'alignement (appelé aussi point d'ancrage) de l'occurrence.
- `180` : valeur d'arrivée de l'interpolation (c'est également la position du point d'alignement).
- `2` : durée de l'interpolation exprimée en secondes.
- `true` : paramètre servant à préciser que la valeur précédente (le chiffre `2`) est exprimée en secondes et non en images. En ayant une cadence de 50 images par seconde dans une animation, nous pourrions utiliser la syntaxe suivante pour préciser une durée de 2 seondes : `new Tween(balle,"y",Regular.easeOut,20,200,100,false)`.

La figure 5-1 montre l'effet de ces paramètres sur la scène.

Dans l'état actuel du script, le mouvement peut être exécuté une fois, mais il est difficile de le contrôler davantage. Imaginons en effet que nous souhaitions réaliser le mouvement inverse (pour ré-enrouler le menu) ou réaliser une action lorsque l'interpolation est terminée. Pour que cela soit possible, il faudrait compléter le script de la façon suivante :

```
Var deroulement = new Tween(balle,"y",Regular.easeOut,20,180,2,true);
```

Ainsi, il devient possible de prévoir d'autres actions et interactions avec le mouvement. C'est la variable `deroulement` (qui est une instance de la classe `Tween()`) que nous allons pouvoir utiliser pour compléter notre script.

Avant d'aborder le développement suivant, consultez le fichier ci-dessous pour découvrir un exemple où le déclenchement d'un tweening se fait par un clic sur une occurrence.

Fichier de référence : `Chapitre5/tween.fla`

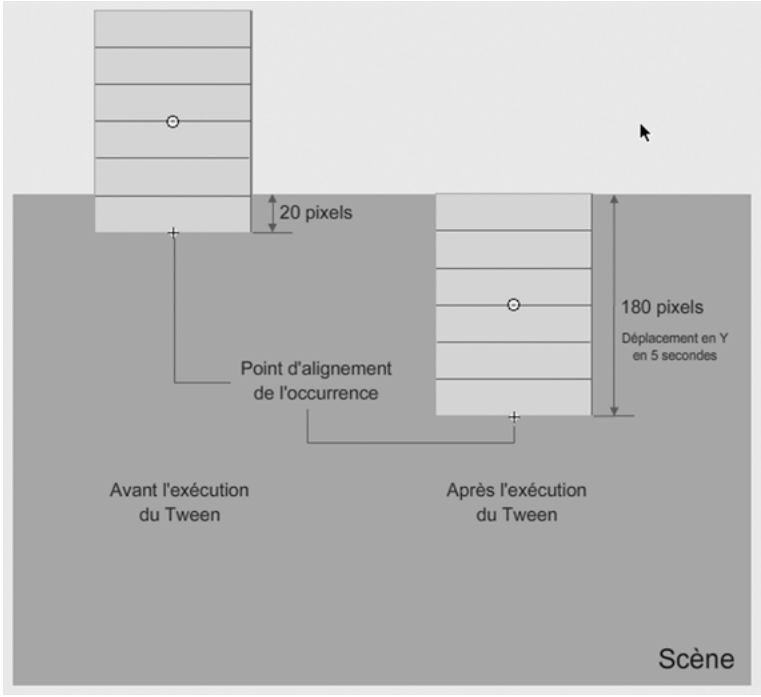


Figure 5-1

Effet des paramètres de la classe Tween sur la scène

Exécuter une instruction à la fin d'une interpolation

Dans l'exemple qui va suivre, nous allons déplacer une occurrence sur la scène en démontrant qu'avant et après l'interpolation il est possible d'exécuter une ou plusieurs lignes d'instructions.

Avant l'interpolation, il suffit tout simplement de placer le code dans la fonction de rappel qui contient l'instanciation de la classe `Tween()`. Dans notre exemple, il s'agit de l'instruction `balle2.scaleX = balle2.scaleY = 0.6`, qui permet de régler l'échelle de l'occurrence à 60 % de sa taille d'origine.

Après l'interpolation, nous ajoutons un écouteur qui gère l'événement `TweenEvent.MOTION_FINISH`.

```
var bouge:Tween;
stage.addEventListener(MouseEvent.CLICK,deplacerBalle2);

function deplacerBalle2(evt:MouseEvent) {
  balle2.scaleX = balle2.scaleY = 0.6;
  bouge = new Tween(balle2,"x",Strong.easeOut,balle2.x,mouseX,1.5,true);
  new Tween(balle2,"y",Strong.easeOut,balle2.y,mouseY,1.5,true);
```

```
  bouge.addEventListener(TweenEvent.MOTION_FINISH ,augmenterTailleOccurrence);
}

function augmenterTailleOccurrence(evt:TweenEvent) {
  balle2.scaleX=1;
  balle2.scaleY=1;
}
```

Pièges classiques

Le déploiement de lignes d'instructions gérant un tween dans un programme est souvent source d'erreurs pour deux raisons principales : la portée des variables locales et la confusion entre déclaration et initialisation.

Fichier de référence : `Chapitre5/tween3.fla`

Une variable locale

Les développeurs néophytes oublient souvent qu'une variable a une portée limitée à la fonction dans laquelle elle a été déclarée. Ainsi, en générant le code ci-dessous, vous ne pouvez faire référence à l'instance de la classe `Tween()` qu'à l'intérieur de la fonction de rappel `deplacerBalle()`.

```
import fl.transitions.easing.*;
import fl.transitions.*;
//
btLancer.addEventListener(MouseEvent.CLICK,deplacerBalle);

function deplacerBalle(evt:MouseEvent) {
  var bouge:Tween = new Tween(balle,"x",Strong.easeOut,60,400,2,true);
  bouge.addEventListener(TweenEvent.MOTION_FINISH ,augmenterTailleOccurrence);
}

function augmenterTailleOccurrence(evt:TweenEvent) {
  trace("Mouvement terminé");
}

balle.addEventListener(MouseEvent.CLICK,replacerBalle);

function replacerBalle(evt:MouseEvent) {
  bouge.yoyo();
}
```

L'exécution de ce script génère une erreur car vous faites référence à l'instance `bouge` dans la dernière fonction du programme alors qu'elle n'a pas été déclarée. Vous devez donc déplacer la déclaration de la variable pour qu'elle puisse être accessible de n'importe quel endroit du programme.

Pour corriger votre script et pouvoir faire référence à l'instance `bouge` à partir d'autres fonctions, effectuez le changement suivant :

```
import fl.transitions.easing.*;
import fl.transitions.*;
//
var bouge:Tween;

btLancer.addEventListener(MouseEvent.CLICK,deplacerBalle);

function deplacerBalle(evt:MouseEvent) {
  bouge = new Tween(balle,"x",Strong.easeOut,60,400,2,true);
  bouge.addEventListener(TweenEvent.MOTION_FINISH ,augmenterTailleOccurrence);
}

function augmenterTailleOccurrence(evt:TweenEvent) {
  trace("Mouvement terminé");
}

balle.addEventListener(MouseEvent.CLICK,replacerBalle);

function replacerBalle(evt:MouseEvent) {
  bouge.yoyo();
}
```

Nous avons placé la déclaration de l'instance `bouge` en dehors de la fonction et nous pouvons ainsi y faire référence, notamment dans la dernière fonction `replacerBalle()`.

Déclarer n'est pas initialiser

Dans l'exemple ci-dessous, vous pouvez constater que l'écouteur qui gère l'événement, `TweenEvent.MOTION_FINISH`, se trouve en dehors de la fonction de rappel `deplacerBalle()`. Cette ligne d'instruction sera donc exécutée avant même que l'instance `bouge` ne soit initialisée, ce qui génère une erreur.

Si vous replacez cette ligne d'instruction dans la fonction de rappel `deplacerBalle()` ou dans une autre qui ne sera appelée qu'à partir du moment ou la variable `bouge` aura été initialisée, cela ne posera plus de problème.

```
var bouge:Tween;

btLancer.addEventListener(MouseEvent.CLICK,deplacerBalle);

function deplacerBalle(evt:MouseEvent) {
  bouge = new Tween(balle,"x",Strong.easeOut,60,400,2,true);

}
bouge.addEventListener(TweenEvent.MOTION_FINISH ,augmenterTailleOccurrence);
```

Le script correct est le suivant :

```
var bouge:Tween;

btLancer.addEventListener(MouseEvent.CLICK,deplacerBalle);

function deplacerBalle(evt:MouseEvent) {
  bouge = new Tween(balle,"x",Strong.easeOut,60,400,2,true);
  bouge.addEventListener(TweenEvent.MOTION_FINISH ,augmenterTailleOccurrence);
}
```

Les modes d'animation

Au début de ce développement dédié à la classe `Tween()`, nous vous précisions que nous reviendrions sur le paramètre `Regular` ; voyons en détail l'intérêt de ce réglage.

Pour obtenir des effets dans l'interpolation, vous pouvez remplacer le paramètre `Regular` (appelé aussi fonction d'accélération) par l'un des quatre autres de la liste ci-dessous :

- `Back`
- `Bounce`
- `Elastic`
- `Strong`

À cela s'ajoutent trois paramètres supplémentaires qui définissent l'application de l'effet en début ou en fin de trajectoire, ou bien même les deux :

- `easeIn`
- `easeOut`
- `easeInOut`

Des effets particuliers

Il est important de signaler qu'une combinaison des paramètres ci-dessus avec les propriétés `x` et `y` permet d'obtenir des mouvements aux trajectoires sinueuses très variables. Essayez, par exemple, ces quelques combinaisons :

> **Remarque**
>
> Configurez une animation de 500 pixels de largeur par 300 de hauteur. Positionnez une occurrence intitulée `balle` à 20 pixels du haut et du bord gauche de la scène. Saisissez le script ci-dessous, puis remplacez la paire de lignes d'instructions contenues dans la fonction de rappel par celles qui sont proposées en exemple.

```
import fl.transitions.easing.*;
import fl.transitions.*;

balle.addEventListener(MouseEvent.CLICK,lancer);
```

```
function lancer(evt:MouseEvent) {
  new Tween(balle,"x",Regular.easeOut,balle.x,420,2,true);
  new Tween(balle,"y",Bounce.easeIn,balle.y,150,2,true);
}
```

Exemples de mouvements :

```
new Tween(balle,"x",Regular.easeIn,balle.x,420,2,true);
new Tween(balle,"y",Bounce.easeIn,balle.y,150,2,true);

new Tween(balle,"x",Elastic.easeIn,balle.x,420,3,true);
new Tween(balle,"y",Bounce.easeInOut,balle.y,150,2,true);

new Tween(balle,"x",Bounce.easeIn,balle.x,420,3,true);
new Tween(balle,"y",Bounce.easeOut,balle.y,150,2,true);

new Tween(balle,"x",Bounce.easeIn,balle.x,420,1,true);
new Tween(balle,"y",Bounce.easeOut,balle.y,150,3,true);

new Tween(balle,"x",Bounce.easeOut,balle.x,420,4,true);
new Tween(balle,"y",Bounce.easeInOut,balle.y,150,1,true);

new Tween(balle,"x",Bounce.easeOut,balle.x,420,4,true);
new Tween(balle,"y",Elastic.easeInOut,balle.y,150,3,true);

new Tween(balle,"x",Bounce.easeOut,balle.x,420,2,true);
new Tween(balle,"y",Elastic.easeIn,balle.y,150,5,true);
```

Éditez l'unique symbole de votre animation pour remplacer la forme rond par un carré. Replacez l'occurrence sur la scène, puis essayez les lignes d'instructions ci-dessous.

```
new Tween(balle,"scaleX",Bounce.easeOut,1,3,2,true);
new Tween(balle,"scaleY",Elastic.easeIn,1,3,2,true);

new Tween(balle,"scaleX",Bounce.easeOut,0.5,3,1,true);
new Tween(balle,"scaleY",Bounce.easeIn,1,3,2,true);

new Tween(balle,"scaleX",Bounce.easeOut,0.5,3,1,true);
new Tween(balle,"scaleY",Elastic.easeInOut,1,3,3,true);

new Tween(balle,"scaleX",Bounce.easeIn,0.5,3,3,true);
new Tween(balle,"scaleY",Elastic.easeOut,1,3,3,true);
```

Notez que le temps alloué aux durées des différentes interpolations donne des résultats très différents. Il ne suffit pas de combiner les effets (`Bounce`, `Elastic`, `Regular`, `Strong` et `Back`), ni même leurs applications (`easeIn`, `easeOut` et `easeInOut`), mais de régler de façon appropriée la durée de l'interpolation.

Propriétés et méthodes complémentaires

Nous avons, jusqu'à présent, expliqué comment réaliser une interpolation par programmation ActionScript grâce à la classe `Tween()`, mais nous n'avons pas détaillé toutes les possibilités de réglages complémentaires. Voici deux tableaux qui vous présentent les propriétés et méthodes que vous pouvez utiliser pour un meilleur contrôle de vos interpolations.

Nous utiliserons l'exemple ci-dessous pour commenter le tableau 5-1.

```
bouge = new Tween(balle,"x",Strong.easeOut,60,400,2,true);
```

Tableau 5-1 Les propriétés de la classe Tween()

Nom de la propriété	Rôle de la propriété
`begin`	Valeur de départ de l'interpolation. 60 dans notre exemple.
`duration`	Durée de l'interpolation. 2 dans notre exemple.
`finish`	Valeur de fin de l'interpolation. 400 dans notre exemple.
`FPS`	Frames/seconde ou IPS pour images/seconde.
`func`	Permet de connaître la fonction d'accélération utilisée.
`isPlaying`	Paramètre permettant de savoir si une interpolation est en cours d'exécution.
`looping`	Paramètre servant à préciser si une interpolation doit être exécutée en boucle. Cette propriété peut également être lue.
`obj`	Nom de l'objet (ou occurrence) concerné par l'interpolation. Valeur `balle` dans notre exemple.
`position`	Permet d'obtenir la valeur de la propriété utilisée dans l'interpolation. Dans notre exemple, l'occurrence va se déplacer de 60 à 400 pixels ; la propriété `position` ne peut donc nous renvoyer qu'une valeur comprise dans cet intervalle.
`prop`	Permet de connaître la propriété utilisée dans l'interpolation. Dans notre exemple, nous obtiendrions `x`.
`time`	Permet d'obtenir le temps écoulé depuis le début de l'interpolation.
`useSeconds`	Permet de savoir si le dernier paramètre de l'interpolation est exprimé en secondes.

Tableau 5-2 Les méthodes de la classe Tween()

Nom de la méthode	Rôle de la méthode
`continueTo()`	Permet de poursuivre l'interpolation en cours vers une nouvelle valeur. Cette méthode ne contient pas de valeur de départ car elle utilise la valeur courante. Seules les valeurs d'arrivée et la nouvelle durée doivent être précisées.
`fforward()`	Termine l'interpolation sans exécuter les interpolations intermédiaires.
`nextFrame()`	Permet d'afficher l'image suivante lorsqu'une interpolation est arrêtée.
`prevFrame()`	Permet d'afficher l'image précédente lorsqu'une interpolation est arrêtée.
`resume()`	Relance une interpolation lorsqu'elle a été mise en pause à l'aide de la méthode `stop()`.
`rewind()`	Réaffecte la valeur de départ à l'occurrence interpolée.
`start()`	Lit une interpolation depuis sa valeur de départ.
`stop()`	Permet d'interrompre l'interpolation.
`yoyo()`	Méthode très pratique permettant d'effectuer l'interpolation dans le sens inverse.

Utilisation de la classe Timer()

Il est intéressant de faire appel à la classe `Timer()` lorsqu'il est nécessaire de maîtriser le rythme de l'effet.

Avec l'événement `ENTER_FRAME` et la classe `Tween()`, il est impossible de contrôler la cadence d'un mouvement. La classe `Timer()` permet de gérer cet aspect-là d'une animation.

Prenons l'exemple d'une horloge, où le mouvement d'une aiguille doit se produire de façon régulière, à chaque seconde. Voici le code qui permet d'obtenir ce comportement.

> **Remarque**
>
> Si vous ne connaissez pas la classe `Timer()`, consultez la fin du chapitre 3 qui traite de cette technique.

```
var mecanisme:Timer = new Timer(1000);
var instant_T:Date;
var coefHoraire:Number;

function tournerAiguilles(evt:TimerEvent) {
  instant_T = new Date();
  coefHoraire = instant_T.getHours()>=13 ? 15 : 30;
  aigHeures.rotation = instant_T.getHours()*coefHoraire;
  aigMinutes.rotation =instant_T.getMinutes()*6;
  aigSecondes.rotation = instant_T.getSeconds()*6;
}

mecanisme.addEventListener(TimerEvent.TIMER,tournerAiguilles);
mecanisme.start();
```

Ce script est intéressant car il présente une technique d'animation totalement différente de toutes celles que nous avons utilisées dans ce chapitre. Alors que la classe `Tween()` nous permettait de définir une durée de mouvement, mais sans nous permettre de contrôler la cadence, ici, nous nous appuyons sur une temporisation très précise.

Par ailleurs, ce script nous permet de découvrir la gestion de la classe `Date()`, également abordée au chapitre 8, consacré aux tableaux, en gérant les jours de la semaine.

L'exemple suivant va combiner les classes `Tween()` et `Timer()` pour démontrer qu'on peut exécuter un mouvement d'une durée précise à intervalle régulier.

```
import fl.transitions.*;
import fl.transitions.easing.*;

var compteur:Timer = new Timer(2500);
var mouvementBille:Tween;

function avancerBille(evt:TimerEvent) {
  new Tween(billeMobile,"x",Regular.easeIn,billeMobile.x,Math.random()*500,1.5,true);
  mouvementBille = new Tween(billeMobile,"y",Strong.easeOut,billeMobile.y,Math.
  ➥random()*280,2,true);
  mouvementBille.addEventListener(TweenEvent.MOTION_FINISH,placerBille);
}
```

```
compteur.addEventListener(TimerEvent.TIMER,avancerBille);
compteur.start();

function placerBille(evt:TweenEvent) {
  var nouvelleBille:Bille = new Bille();
  nouvelleBille.x=billeMobile.x;
  nouvelleBille.y=billeMobile.y;
  addChild(nouvelleBille);
}
```

Dans un premier temps, nous mettons en place un timer avec un intervalle régulier de 2,5 secondes (`new Timer(2500)`). Puis nous définissons le mouvement de la bille, qui se déplace sur la scène en écrivant deux lignes d'instructions faisant appel à la classe `Tween()`. Comme vous l'aurez peut-être remarqué, nous faisons référence aux mouvements sinueux abordés précédemment lors de la présentation de la classe `Tween()` en combinant, dans cet exemple, les fonctions d'accélération `Regular.easeIn` et `Strong.easeOut`.

En conclusion, c'est la répétition à intervalle plus ou moins long mais régulier qui nous permet d'obtenir, dans ces deux cas et d'une façon générale avec la classe `Timer()`, un mouvement.

Utilisation des fonctions Math.sin() et Math.cos()

Pour obtenir un mouvement pendulaire, circulaire ou ondulatoire, vous allez découvrir que le code à écrire fait appel aux fonctions `Math.sin()` et `Math.cos()`. D'une façon plus générale, nous pouvons dire qu'elles vont nous permettre d'obtenir des mouvements cycliques.

À la lecture de ces deux termes, cosinus et sinus, vous êtes peut-être déjà réticent à l'idée de devoir travailler avec de telles notions, car elles vous rappellent de mauvais souvenirs de trigonométrie qui datent de la classe de troisième, au collège !

Essayons de vous rappeler et/ou de vous faire comprendre ce que nous permettent d'obtenir les fonctions trigonométriques cosinus et sinus.

Commencez par essayer d'effectuer les opérations ci-dessous :

- $3 + 4 = ?$
- $5 - 3 = ?$
- $8 \div 2 = ?$
- $3 \times 4 = ?$

Ces calculs vous semblent simples car ce sont des opérations que vous êtes amenés à faire au quotidien. Essayons d'aller plus loin, dans un tout autre registre :

- $2 \times \pi = ?$
- $\sqrt{9} = ?$

Même si ces deux questions peuvent paraître plus complexes au premier abord, notre mémoire prend vite le dessus. Pour le calcul avec pi, nous nous rappelons de façon automatique que ces deux lettres sont associées à la valeur 3,14. Dans la cas de la racine carrée, nous nous souvenons que le nombre situé sous le signe √ doit être le résultat de la multiplication d'un nombre par lui-même. La racine carrée et les calculs faisant appel à pi vous semblent donc plus difficiles qu'une opération à base d'addition, de soustraction, de division ou de multiplication. Mais les opérations ci-dessous vous paraissent-elles plus faciles que √16 ou √25 ?

- 234 786 × 765 432 = ?
- 4 537 365 / 38 = ?

Quel serait votre réflexe dans ce cas présent ? Vous allez très probablement vous tourner vers la calculatrice !

Même si vous savez faire une multiplication et une division, et que vous avec un ordre de grandeur du résultat obtenu (un nombre plus grand pour une multiplication, plus petit pour une division), vous utilisez une calculatrice pour aller plus vite, mais aussi parce que vous faites confiance au résultat obtenu. Eh bien pour le calcul des sinus et cosinus c'est pareil, il suffit que vous connaissiez l'ordre de grandeur du résultat, même si vous n'êtes pas capable de le calculer.

Quel peut donc être le résultat du calcul de cosinus 17 par exemple, mais surtout comment l'exploiter dans le calcul d'un mouvement ?

En tant que non mathématicien, la seule réponse que je puisse vous apporter est que le résultat sera un nombre compris entre -1 et 1.

Mais ces valeurs sont-elles pour autant aléatoires ? Non, et vous pouvez le constater sur la figure 5-2 qui associe un tableau de valeurs avec un graphique des fonctions sinus et cosinus.

+	0.00	0.30	0.60	0.90	1.20	1.50	1.80	2.10	2.40	2.70	3.00	3.30	3.60	3.90	4.20	4.50	4.80	5.10	5.40	5.70	6.00	6.30
Cosinus □	1.00	0.96	0.83	0.62	0.36	0.07	-0.23	-0.50	-0.74	-0.90	-0.99	-0.99	-0.90	-0.73	-0.49	-0.21	0.09	0.38	0.63	0.83	0.96	1.00
Sinus ●	0.00	0.30	0.56	0.78	0.93	1.00	0.97	0.86	0.68	0.43	0.14	-0.16	-0.44	-0.69	-0.87	-0.98	-1.00	-0.93	-0.77	-0.55	-0.28	0.02

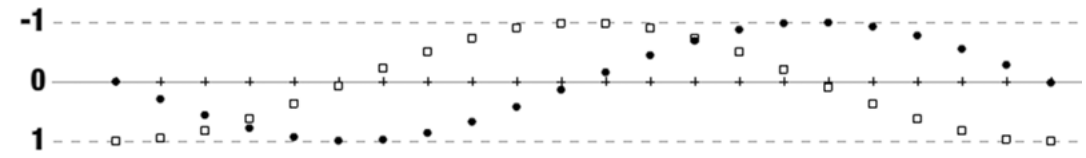


Figure 5-2

Les valeurs des cosinus et sinus de quelques nombres, avec le graphique associé.

Rappel

Nous utiliserons également dans nos exemples, la fonction `Math.abs()` qui permet d'obtenir la valeur absolue d'un nombre, c'est-à-dire une valeur toujours positive (la valeur absolue de –5 est 5 par exemple).

Remarque

Dans le graphique de la figure 5-2, les valeurs -1 et 1 sont inversés car nous avons utilisé Flash pour réaliser cette illustration. Rappelons que, dans ce logiciel, le coin supérieur gauche de la scène correspond aux coordonnées (x,y) = (0,0). De ce fait les valeur positives sont dans la scène et les valeurs négatives au-dessus.

Vous pouvez constater sur le graphique de la figure 5-2 que le résultat des fonctions sinus et cosinus est bien toujours compris entre –1 et 1 en passant par 0. Nous avons ici choisi des valeurs comprises entre 0 et 6,3, mais cela aurait été vrai pour toute autre suite de nombres. Ajoutons que le cosinus de 0 vaut 1 et le sinus de 0 vaut 0, nous avons donc un décalage initial de 1.

Nous avons positionné des carrés et des ronds sur le graphique de la figure 5-2 pour illustrer les valeurs obtenues lors du calcul des fonctions sinus et cosinus. Cela nous permet d'effectuer un deuxième constat, le calcul des sinus et cosinus renvoie des valeurs périodiques : le graphique obtenu sera le même pour des valeurs situées entre 0 et 6,3 que pour des valeurs comprises entre 6,3 et 12,6.

Essayez de réaliser une première animation avec le code ci-dessous ; vous constaterez que le calcul du cosinus ou du sinus vous permet de faire bouger une occurrence en x ou en y d'un point à un autre.

Ajoutons simplement que pour obtenir la trajectoire que vous allez découvrir, il nous a fallu multiplier le résultat obtenu par le calcul du cosinus par 50 pour obtenir un déplacement sur 100 pixels : de –50 à +50 pixels (rappelons que la valeur du cosinus varie entre –1 et 1).

Fichier de référence : `Chapitre5/cossin1.fla`

```
var valeurCroissante:Number = 0;

planete.addEventListener(Event.ENTER_FRAME,mouvementLineaire);

function mouvementLineaire(evt:Event) {
  valeurCroissante+=0.1;
  evt.currentTarget.x=255+(Math.sin(valeurCroissante)*50);
}
```

Le mouvement de cette occurrence est un déplacement horizontal variant d'une position située à 205 pixels du bord gauche de la scène (255 - 50) jusqu'à celle située à 305 pixels (255 + 50). La valeur 255 de la ligne précédente correspond à l'axe autour duquel l'occurrence se déplace.

Remarque

Si vous observez un saut de l'image au lancement de l'animation, réglez manuellement la position x de votre de occurrence à 255 pixels (via la palette Propriétés).

Mouvement circulaire

Fichier de référence : `Chapitre5/cossin2.fla`

Dans ce deuxième exemple, nous réussissons à obtenir un mouvement circulaire car nous combinons en x et en y, le calcul d'un cosinus et d'un sinus, en partant de la même valeur. Consultez le lien ci-dessous pour vous rendre compte de l'obtention d'un mouvement circulaire qui résulte de la combinaison de deux positions dans un repère orthonormé (le cosinus est représenté par le point qui se déplace sur l'axe horizontal, le sinus par celui qui se déplace sur l'axe vertical). Il est en effet très difficile d'expliquer par écrit, sans possibilité de schéma animé, la démonstration suivante :

`http://www.yazo.net/index.php?option=com_content&task=view&id=32&Itemid=47`

```
var valeurCroissante:Number = 0;

planete.addEventListener(Event.ENTER_FRAME,mouvementCirculaire);

function mouvementCirculaire(evt:Event) {
  valeurCroissante+=0.1;
  evt.currentTarget.x=250+(Math.cos(valeurCroissante)*100);
  evt.currentTarget.y=130+(Math.sin(valeurCroissante)*100);
}
```

Mouvement plus elliptique

Dans l'exemple précédent, nous obtenons une trajectoire circulaire car l'occurrence se déplace du même nombre de pixels dans les directions horizontale et verticale. Si elle se déplaçait plus loin horizontalement ou verticalement, nous obtiendrions un mouvement circulaire aplati, c'est-à-dire une trajectoire elliptique. Changez donc l'une des deux valeurs (`100`) par un nombre plus grand ou plus petit et vous obtiendrez une animation différente.

Accélérer ou ralentir le mouvement

Si vous avez utilisé un gestionnaire `ENTER_FRAME` pour exécuter votre code en continu (notamment l'incrémentation de la valeur de la variable `valeurCroissante` de notre exemple), la vitesse d'exécution de votre animation dépend du nombre d'images par seconde de votre fichier `.fla`. Si vous utilisez la classe `Timer()`, vous pouvez préciser l'intervalle temps entre deux répétitions.

Il existe une autre technique qui consiste à effectuer un calcul plus précis du cosinus et/ou du sinus afin d'obtenir plus de valeurs pour positionner votre occurrence. Plus le pas de

la variable servant de base au calcul du cosinus et/ou du sinus (+ 0.1 dans notre exemple) est important, plus rapide sera le mouvement ; en revanche, moins il sera précis.

Mouvement pendulaire

Fichier de référence : `Chapitre5/cossin3.fla`

Un mouvement pendulaire est très proche d'une trajectoire circulaire. Ce qui différencie ces deux trajectoires est le traitement appliqué au calcul de la position verticale.

En observant l'animation de l'adresse ci-dessous, vous pouvez constater que les deux occurrences noire et rouge passent toutes les deux au-dessus de l'axe des abscisses.

`http://www.yazo.net/index.php?option=com_content&task=view&id=32&Itemid=47`

Nous devons réussir à les conserver sous cet axe, ce qui peut être obtenu facilement en utilisant la fonction mathématique qui renvoie une valeur absolue : `Math.abs()`. Par exemple, `Math.abs(-23,5)` donne 23,5.

```
var valeurCroissante:Number = 0;

planete.addEventListener(Event.ENTER_FRAME,mouvementLineaire);

function mouvementLineaire(evt:Event) {
  valeurCroissante+=0.05;
  evt.currentTarget.x=250+(Math.sin(valeurCroissante)*50);
  evt.currentTarget.y=150+(Math.abs(Math.cos(valeurCroissante))*50);
}
```

Le fait de placer `Math.cos()` dans la fonction `Math.abs()`, nous permet d'obtenir une valeur positive.

Pour obtenir un mouvement pendulaire au-dessus de l'axe des abscisses, il suffit de rendre négatives toutes les valeurs calculées. Il suffit donc de les multiplier par -1. Dans la mesure où nous effectuons déjà une multiplication dans notre calcul, remplaçons simplement 50 par -50.

Effet pulse

Fichier de référence : `Chapitre5/cossin3.fla`

Qui n'a jamais vu un Macintosh avec sa veille lumineuse qui respire ? Pour obtenir un effet d'occurrence qui devient de plus en plus transparente avant de redevenir opaque puis de recommencer perpétuellement ce cycle, donnant ainsi l'impression de pulsions, il vous suffit de partir du script du mouvement pendulaire. La combinaison d'une fonction sinus ou cosinus avec une valeur absolue permet d'obtenir une série de nombres que vous pouvez alors utiliser avec la propriété alpha.

Fichier de référence : `Chapitre5/cossin3.fla`

```
var valeurCroissante:Number = 0;

spot.addEventListener(Event.ENTER_FRAME,mouvementLineaire);

function mouvementLineaire(evt:Event) {
  valeurCroissante+=0.05;
  evt.currentTarget.alpha = Math.abs(Math.cos(valeurCroissante));
}
```

Ajoutons que vous pouvez également utiliser les propriétés `scaleX` et `scaleY` pour obtenir le même effet en faisant varier l'échelle d'une occurrence et non plus sa transparence.

Remarque

Consultez également, au début de ce chapitre, le développement dédié à l'événement `ENTER_FRAME` qui fait référence à deux autres exemples à travers les points Obtenir un mouvement sinueux et Saccader un mouvement.

Mouvement d'une planète

Après avoir analysé ensemble tous les scripts liés aux calculs de cosinus et sinus dans les exemples précédents, voici un script que vous devriez comprendre sans difficulté.

Fichier de référence : `Chapitre5/cossin3.fla`

```
var valeurCroissante:Number = 0;

planete.addEventListener(Event.ENTER_FRAME,mouvementCirculaire);

function mouvementCirculaire(evt:Event) {
  valeurCroissante+=0.07;
  evt.currentTarget.x=250+(Math.cos(valeurCroissante)*200);
  evt.currentTarget.y=130+(Math.sin(valeurCroissante)*50);

  evt.currentTarget.scaleX=0.5+((evt.currentTarget.y-80)/130);
  evt.currentTarget.scaleY=0.5+((evt.currentTarget.y-80)/130);

  evt.currentTarget.alpha=0.3+((evt.currentTarget.y-80)/130);
}
```

Il s'agit d'un mouvement classique qui n'est pas uniquement utilisé pour la représentation des mouvements des planètes du système solaire, mais également pour afficher plusieurs images sur la scène d'une animation.

Pour obtenir l'effet escompté, nous avons simplement ajouté deux lignes d'instructions qui permettent de régler l'échelle de l'occurrence qui tourne en fonction de sa position verticale ; nous avons également revu à la baisse le rayon vertical afin d'obtenir un mouvement elliptique.

Déplacement d'une occurrence d'un point à un autre de la scène

Pour le dernier développement de ce chapitre, nous avons souhaité regrouper sur cette page, deux scripts qui permettent de déplacer une occurrence à l'endroit d'un clic de la souris sur la scène.

Il s'agit de lignes de code que vous avez déjà rencontrées dans ce livre dans des exemples plus ou moins similaires.

Nous devons commencer par gérer le clic sur la scène avec le gestionnaire suivant :

```
stage.addEventListener(MouseEvent.CLICK,deplacerOccurrence);
function deplacerOccurrence (evt:MouseEvent) {

}
```

Remarque

Dans cet exemple, nous faisons référence à la propriété `stage` de l'animation (`this`) et non à une quelconque instance intitulée `stage`. Rappelons que chaque occurrence sur la scène possède cette propriété.

Nous devons ensuite écrire le code chargé de déplacer de façon continue l'occurrence en faisant appel aux propriétés `x` et `y`. Deux possibilités s'offrent à vous.

Première possibilité, vous utilisez un gestionnaire qui fait appel à l'événement `ENTER_FRAME` ou vous faites appel à la classe `Timer()`. Dans les deux cas vous ferez référence aux propriétés `x` et `y` de l'occurrence à déplacer.

Deuxième possibilité, vous faites appel à la classe `Tween()` ; précisons que cette dernière solution est préférable dans la mesure où le choix d'effet dans le mouvement est facilement paramétrable.

Script avec la classe `Tween()` :

```
import fl.transitions.easing.*;
import fl.transitions.*;

var mouvementX:Tween;
var mouvementY:Tween;

stage.addEventListener(MouseEvent.CLICK,deplacerPuce);

function deplacerPuce(evt:MouseEvent) {
  mouvementX = new Tween(puce,"x",Strong.easeOut,puce.x,mouseX,1.5,true);
  mouvementY =new Tween(puce,"y",Strong.easeOut,puce.y,mouseY,1.5,true);
}
```

Script avec un gestionnaire de type `Event` et l'événement `ENTER_FRAME` :

```
var destX:Number;
var destY:Number;
```

```
stage.addEventListener(MouseEvent.CLICK,deplacerPuce);

function deplacerPuce(evt:MouseEvent) {
  destX=mouseX;
  destY=mouseY;
  puce.addEventListener(Event.ENTER_FRAME,avancerBalle);

}

function avancerBalle(evt:Event) {
  puce.x+=(destX-puce.x)*0.1;
  puce.y+=(destY-puce.y)*0.1;
  if (Math.abs(destX-puce.x) + Math.abs(destY-puce.y) < 2 ) {
    puce.removeEventListener(Event.ENTER_FRAME,avancerBalle);
  }
}
```

Si vous faites appel à un gestionnaire de type `Event` avec l'événement `ENTER_FRAME`, pensez à exécuter la fonction `removeEventListener()` lorsque votre occurrence a terminé sa trajectoire. Vous noterez que pour évaluer la proximité de l'occurrence à son point de destination, nous vérifions si elle se trouve à moins d'un pixel de distance du clic de la souris. Dans la mesure où la soustraction risque de renvoyer une valeur négative (`destX-puce.x`), nous en prenons la valeur absolue.

Adaptons une dernière fois le script ci-dessus afin que l'occurrence suive perpétuellement le curseur de la souris.

```
stage.addEventListener(MouseEvent.CLICK,deplacerPuce);

function deplacerPuce(evt:MouseEvent) {

  puce.addEventListener(Event.ENTER_FRAME,avancerBalle);

}

function avancerBalle(evt:Event) {
  puce.x+=(mouseX-puce.x)*0.1;
  puce.y+=(mouseY-puce.y)*0.1;
}
```

Retirez éventuellement le gestionnaire `MouseEvent` afin que l'occurrence `puce` suive le curseur de la souris dès le lancement de l'animation.

```
puce.addEventListener(Event.ENTER_FRAME,avancerBalle);

function avancerBalle(evt:Event) {
  puce.x+=(mouseX-puce.x)*0.1;
  puce.y+=(mouseY-puce.y)*0.1;
}
```

Dans ce dernier exemple, il est préférable d'utiliser ce script plutôt que des lignes d'instructions faisant appel à la classe `Tween()`.

6

La construction d'une interface sur la scène

Quel que soit votre niveau en ActionScript, au début de chaque nouvelle production, vous vous interrogerez toujours sur la manière de concevoir une animation. Vous pouvez par exemple simplement glisser-déposer des symboles sur la scène et des images-clés tout au long du scénario, mais plus vos besoins en matière d'interactivité seront grands, plus il sera nécessaire d'éviter une telle méthode.

Lorsque vous ne connaissiez pas l'ActionScript, le seul moyen de réaliser la mise en page des différentes parties de votre animation était de faire appel à l'IDE de Flash. Vous utilisiez alors principalement les règles de la scène, la palette Aligner, les copier-coller d'occurrences, la manipulation d'images et d'images-clés, en résumé, tout ce qui vous permettait de manipuler l'interface sans avoir à utiliser l'ActionScript.

À l'opposé, certains développeurs chevronnés vont jusqu'à se passer de l'interface de Flash et n'utilisent que des fichiers externes sans manipuler le moindre symbole. Outre le plaisir éprouvé à ne manier que des lignes de code, ce mode de développement s'avère très rapidement plus efficace et plus optimisé (gestion du temps de production et des mises à jour d'un projet).

Démonstration

Pour illustrer notre propos, nous avons réalisé une série de trois animations qui répondent toutes au même besoin : utiliser une barre de navigation latérale pour pouvoir changer le contenu central de l'animation (un simple texte dans notre exemple).

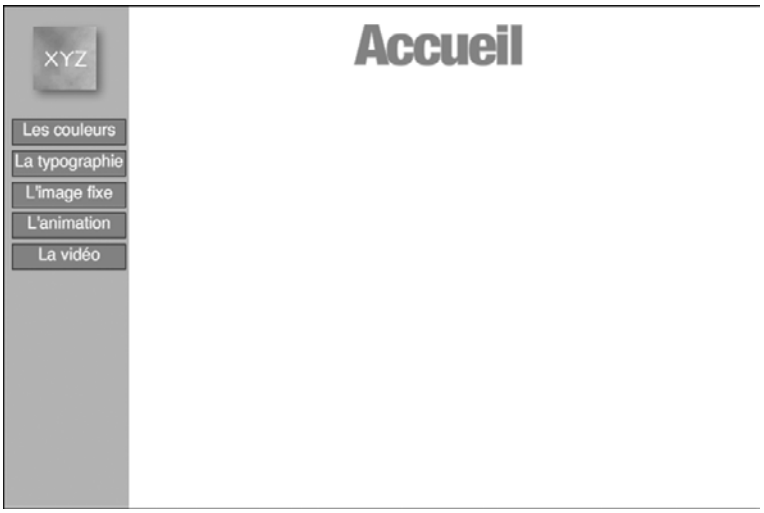


Figure 6-1

Cette interface va être obtenue à partir de méthodes toutes très différentes les unes des autres.

Pour chacun de ces trois exemples, nous allons présenter l'état de la scène avant la publication, la bibliothèque, le scénario et le script principal qui gère l'interactivité de l'animation. Vous allez découvrir que les méthodes employées sont réellement toutes très différentes.

Construction à base de symboles glissés sur la scène

Fichier de référence : `Chapitre6/ConstructionStatique.fla`

État de la scène : tous les éléments qu'elle contient ont été placés en saisissant des symboles de la bibliothèque pour venir les déposer à différents endroits sur la scène, c'est-à-dire par une technique de glisser-déposer. Les textes des boutons ont été créés directement sur la scène à l'aide de l'outil texte. L'ombre du logo a été ajoutée à partir de l'interface de Flash en faisant appel à la palette Filtres (fig. 6-2).

Contenu de la bibliothèque : tous les éléments utilisés dans la construction de l'interface sont présents dans la bibliothèque. Seuls les textes des boutons n'y figurent pas car ils ont été créés sur la scène, comme nous le précisions ci-dessus (fig. 6-3).

Aspect du scénario : notre animation possède cinq boutons sur l'écran du sommaire. Avec cette technique de construction d'une interface et, plus globalement, d'une animation, le scénario possède de ce fait six images-clés espacées les unes des autres (fig. 6-4).

> **Remarque**
>
> Les images-clés sont espacées les unes des autres afin que vous puissiez, sans difficulté, en insérer une nouvelle, lorsque le besoin se présente.

Figure 6-2

Les éléments que contient la scène montrent déjà l'apparence qu'aura l'animation après publication.

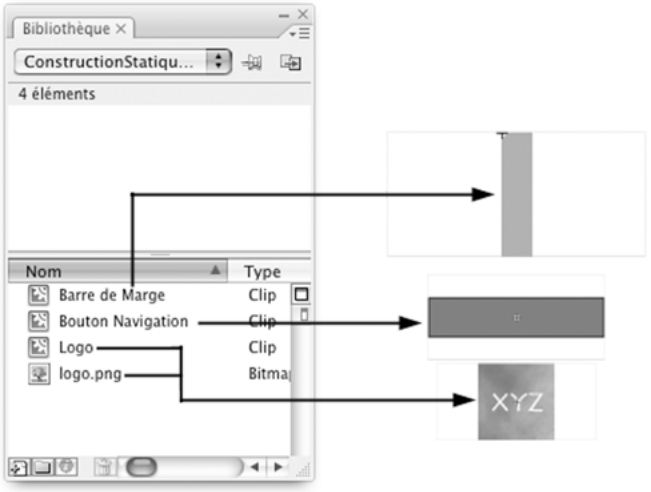


Figure 6-3

La bibliothèque possède tous les symboles utilisés dans l'animation avant sa publication.

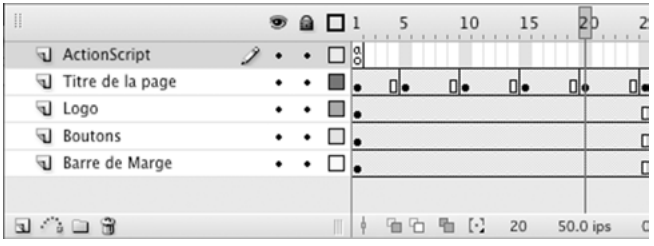


Figure 6-4

Chaque écran de l'animation correspond à une image-clé.

Script de l'image-clé 1 : il contient uniquement des lignes d'instructions relatives à la navigation au sein de l'animation.

```
stop();

btCouleurs.addEventListener(MouseEvent.CLICK,allerImage5);
btTypographie.addEventListener(MouseEvent.CLICK,allerImage10);
btImageFixe.addEventListener(MouseEvent.CLICK,allerImage15);
btAnimation.addEventListener(MouseEvent.CLICK,allerImage20);
btVideo.addEventListener(MouseEvent.CLICK,allerImage25);

function allerImage5(evt:MouseEvent) {
  gotoAndStop(5);
}
function allerImage10(evt:MouseEvent) {
  gotoAndStop(10);
}
function allerImage15(evt:MouseEvent) {
  gotoAndStop(15);
}
function allerImage20(evt:MouseEvent) {
  gotoAndStop(20);
}
function allerImage25(evt:MouseEvent) {
  gotoAndStop(25);
}
```

Conclusion : cet exemple tend à démontrer qu'il devient très rapidement fastidieux de construire une interface par glisser-déposer de symboles et par création d'images-clés. L'exemple aurait été encore plus flagrant si nous avions eu à gérer une quinzaine de boutons. En revanche, cela démontre qu'il est facile de réaliser une animation contenant plusieurs écrans à partir de multiples images-clés, dès lors que vous connaissez un minimum de manipulations dans l'IDE de Flash.

Symbole avec liaison et utilisation de la Timeline

Fichier de référence : `Chapitre6/ConstructionDynamique.fla`

Dans ce deuxième fichier, l'animation est en partie construite dynamiquement. L'interface est en effet obtenue par l'exécution de lignes d'instructions qui génèrent les boutons en marge gauche. L'affichage des différents écrans de l'animation est obtenue, comme dans l'exemple précédent, par navigation entre les différentes images-clés du scénario.

État de la scène : comme vous pouvez le constater, à la différence du premier fichier, les boutons de navigation ne figurent plus dans le panneau latéral gauche, car ils vont être placés dynamiquement sur la scène.

Figure 6-5

La construction du sommaire est obtenue, en partie, grâce à l'exécution de lignes d'instructions.

Contenu de la bibliothèque : seul le symbole intitulé Bouton Navigation a été modifié par rapport à l'exemple précédent. Nous lui avons ajouté un texte dynamique.

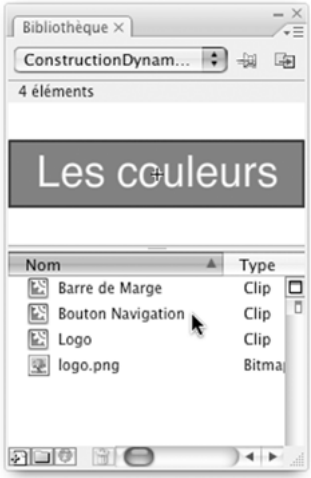


Figure 6-6

La bibliothèque possède toujours tous les symboles utilisés dans l'animation avant sa publication.

Aspect du scénario : dans ce deuxième exemple, nous ne cherchons toujours pas à construire dynamiquement le contenu des différents écrans de l'animation ; nous utilisons donc encore les images-clés pour en construire les différentes parties. Le scénario est le même que celui de l'exemple précédent (figure 6-4).

Script de l'image-clé 1 : le script principal de l'animation est chargé de gérer la navigation entre les différentes images, mais il sert également à construire une partie de l'interface de l'animation : la barre de navigation.

```
stop();

var serieBoutons=["btCouleurs","btTypographie","btImageFixe","btAnimation","btVideo"];
var numeroEntree:Number;

for (var entree:String in serieBoutons) {
  numeroEntree = Number(entree);
  var bouton:BoutonNavigation = new BoutonNavigation();
  bouton.name=entree;
  bouton.x=50;
  bouton.y=100+(numeroEntree*30);
  bouton.etiquette.text=serieBoutons[numeroEntree].substr(2,20);
  addChild(bouton);
  bouton.numeroImage=5+(numeroEntree*5);
  bouton.addEventListener(MouseEvent.CLICK,deplacerTete);
}

function deplacerTete(evt:MouseEvent) {
  gotoAndStop(evt.currentTarget.numeroImage);
}
```

Conclusion : cet exemple illustre qu'il est souvent intéressant de combiner manipulation de l'interface de Flash et gestion du code lorsqu'on ne maîtrise pas suffisamment le langage ActionScript. Le codage de la construction de la barre de navigation est un gain de temps notable.

Construction d'une animation à partir d'un fichier externe

Fichiers de référence : `Chapitre6/ConstructionFichierAS.fla`, `logo.png`, `main.as`, `importImage.as` et `TracerRectangles.as`

Dans ce troisième et dernier fichier, l'animation est intégralement construite dynamiquement. Cela signifie que l'interface est obtenue par l'exécution de lignes d'instructions en ActionScript, contenues dans des fichiers externes d'extension `.as`.

État de la scène : elle ne contient rien.

Aspect du scénario : il ne contient rien

Contenu de la bibliothèque : elle contient un symbole et une police de caractères, mais cette dernière aurait pu être importée dans l'animation à partir de l'exécution d'une ligne d'instruction. Par ailleurs, le seul et unique symbole de l'animation aurait pu ne pas se trouver dans la bibliothèque : sa construction aurait pu être obtenue à partir d'un script en

AS. Nous avons simplement souhaité montrer qu'il était possible de faire appel à un symbole de la bibliothèque à partir d'un fichier externe.

Figure 6-7

Ces éléments sont présents dans la bibliothèque dans le seul but de démontrer qu'un symbole peut être appelé à partir d'un fichier externe.

Script de l'image-clé 1 : il n'y en a pas. En revanche, nous avons créé trois fichiers AS dont voici le contenu.

Fichier de référence : `Chapitre6/main.as`

```
package {

  import flash.display.Sprite;
  import flash.display.MovieClip;
  import flash.events.MouseEvent;
  import flash.text.TextField;
  import flash.text.TextFieldAutoSize;
  import flash.text.TextFormat;
  import flash.text.TextFormatAlign;

  public class main extends Sprite {

    public var serieBoutons=["btCouleurs","btTypographie","btImageFixe",
    ➥"btAnimation","btVideo"];
    public var numeroEntree:Number;
    public var titre:TextField;
    public var styleEnTitre:TextFormat;
    public var marge:Sprite;
    public var corps:Sprite
```

```
public var barreMarge:TracerRectangle;
public var logo:ImportImage;

function main() {

  marge = new Sprite();
  corps = new Sprite();
  corps.x=100;

  addChild(marge); // Placé après la fonction ConstruireBarreLateralePrincipale
  addChild(corps); // cela fonctionne quand même car les objets sont créés,
                   mais pas encore affichés


  ConstruireBarreLateralePrincipale();

}
private function ConstruireBarreLateralePrincipale() {

  barreMarge = new TracerRectangle(0,0,100,400,"0xE29366");
  marge.addChild(barreMarge);

  logo = new ImportImage("logo.png");
  logo.x=25;
  logo.y=10;
  marge.addChild(logo);

  for (var entree:String in serieBoutons) {
    numeroEntree=Number(entree);
    var bouton:BoutonNavigation=new BoutonNavigation();
    bouton.name=entree;
    bouton.x=50;
    bouton.y=100 + numeroEntree * 30;
    bouton.etiquette.text=serieBoutons[numeroEntree].substr(2,20);
    marge.addChild(bouton);
    bouton.numeroImage=5 + numeroEntree * 5;
    bouton.addEventListener(MouseEvent.CLICK,chargerEcran);
  }
}
public function chargerEcran(evt:MouseEvent) {
  placerTitre(evt.currentTarget.etiquette.text);
}
public function placerTitre(nomTitre) {

  //removeChild(marge) Pour faire disparaître la barre avec les boutons

  if (titre != null) {
    corps.removeChild(titre);
  }

  titre=new TextField();
  titre.name="zoneDeTitre";
  titre.text=nomTitre;
```

```
        titre.x=50;
        titre.y=30;
        titre.selectable=false;
        titre.width=350;
        titre.autoSize=TextFieldAutoSize.LEFT;
        titre.embedFonts=true;

        corps.addChild(titre);

        styleEnTitre=new TextFormat();
        styleEnTitre.color="0xC44F10";
        styleEnTitre.size=24;
        styleEnTitre.align=TextFormatAlign.RIGHT;
        styleEnTitre.font="Helvetica Neue Black Condensed";
        titre.setTextFormat(styleEnTitre);

      }
    }
}
```

Fichier de référence : `Chapitre6/ImportImage.as`

```
package {

  import flash.display.Sprite;
  import flash.net.URLRequest;
  import flash.display.Loader;

  public class ImportImage extends Sprite {

    private var nomImage:String;
    private var chargeur:Loader;
    private var adresseImage:URLRequest;

    function ImportImage(nomImage) {
      adresseImage = new URLRequest(nomImage);
      chargeur = new Loader();
      chargeur.load(adresseImage);
      addChild(chargeur);
    }
  }
}
```

Fichier de référence : `Chapitre6/TracerRectangle.as`

```
package {

  import flash.display.Sprite;
  import flash.geom.Rectangle;
  import flash.display.Shape;

  public class TracerRectangle extends Sprite {

    private var origineX:Number;
    private var origineY:Number;
```

```
    private var largeur:Number;
    private var hauteur:Number;
    private var couleur:String;

    function TracerRectangle(origineX,origioneY,largeur,hauteur,couleur) {
      var square:Shape=new Shape();
      square.graphics.beginFill(couleur,1);
      square.graphics.drawRect(origineX,origineY,largeur,hauteur);
      addChild(square);
    }
  }
}
```

Conclusion : cette animation met en évidence que la construction d'une interface et d'une animation peuvent être conduites sans utiliser l'IDE de Flash. Il est vrai que pour faire appel à une telle méthode, il est nécessaire de bien connaître l'ActionScript et/ou un langage de programmation orienté objet. Cela dit, en possession de ces connaissances, vous découvrirez très rapidement que la manipulation de l'interface du logiciel ralentit la production de vos projets. Par ailleurs, la mise à jour d'un projet qui utilise ce mode de développement est plus souple. Afin que cet exemple puisse être accessible à un plus grand nombre, nous n'avons pas voulu faire appel à un fichier XML, ce qui aurait été plus judicieux pour stocker des informations relatives à chacun des 6 écrans, c'est-à-dire le contenu du sommaire et celui des cinq parties de notre animation.

Symbole glissé vers la scène (environnement auteur)

Glisser-déposer un symbole sur la scène est la méthode la plus accessible aux néophytes en programmation ! Ils peuvent très facilement construire différents écrans (sur des images-clés) tout au long du scénario d'une animation sans qu'il soit nécessaire de saisir la moindre ligne d'instruction. Leur besoin en matière d'interactivité s'arrête généralement au contrôle de la tête de lecture d'une animation pour visualiser ces différents écrans.

Pour pouvoir glisser-déposer des symboles sur la scène, il faut avoir effectué, au préalable, un travail d'importation de médias. Ceci a pour conséquence d'alourdir plus ou moins le poids des fichiers .fla et .swf. Même si aujourd'hui les débits d'Internet permettent le transfert de plusieurs méga-octets en quelques secondes ou minutes, il est souvent inutile d'imposer à un utilisateur le chargement intégral d'une animation, surtout si celle-ci comporte de nombreux médias.

Nous avons évoqué, au début de ce chapitre, la notion de mise à jour. Or, il s'avère impossible de mettre à jour une animation construite par glisser-déposer de symboles. Relativisons notre propos : rien ne vous empêche d'ouvrir un fichier Flash (dont l'extension est .fla) que vous auriez réalisé quelques temps auparavant, pour importer de nouveaux médias et/ou changer la dispositions des occurrences sur la scène. Cependant, si les mises à jour doivent se répéter fréquemment, cette opération devient très vite fastidieuse. L'avantage de faire appel à des lignes de code en ActionScript pour gérer la construction et l'interactivité d'une animation est justement d'éviter la réouverture d'un fichier Flash qui doit être mis à jour.

La méthode de construction d'une animation qui consiste donc à faire principalement appel à la timeline (le scénario) pour créer les différents écrans d'un projet à base de glisser-déposer de symboles sur la scène a ses limites ! Comme nous l'évoquions ci-dessus, dès que vous devrez gérer des médias tels que la diffusion de sons ou la consultation de vidéos au sein d'une animation, il sera indispensable d'utiliser la programmation.

En conclusion, nous ne conseillerions cette méthode de développement d'animation qu'à des personnes dont les besoins en matière de programmation sont très limités. Elle peut également constituer une solution pour un développement rapide d'animations relativement simples ou bien pour les personnes qui ne parviennent pas à programmer en dehors des notions de base.

Méthodologie de développement

Supposons que vous n'ayez pas le temps d'apprendre en détail la programmation en AS3, mais que vous souhaitiez très rapidement être capable de réaliser une animation interactive comme celles que nous vous avons présentées (le premier exemple, `Chapitre6/Construction-Statique.fla`). Voici une procédure pour vous aider dans votre première réalisation.

Pour commencer, effectuez une arborescence rapide de votre animation ou un *storyboard*, pour en déterminer le nombre d'écrans total.

1. Créez tout d'abord un nouveau document Fichier Flash (AS 3.0).
2. Enregistrez votre animation.
3. Réglez les paramètres de votre animation (largeur et hauteur de la scène, couleur et réglez la cadence à 50 ips).
4. Créez les symboles dont vous aurez besoin dans votre animation.
5. Importez également les différents médias que vous utiliserez dans les différentes images-clés de votre animation.
6. Au niveau du scénario (la timeline), créez, sur un ou plusieurs calques, les éléments du décor, tout du moins les parties de l'interface graphique qui se répètent d'un écran à l'autre. Ces calques devront se situer sous les autres.

> **Remarque**
>
> Pensez à étendre vos calques de fond sur autant d'images que nécessaire. Prenez exemple sur la copie d'écran de la figure 6-4 (Logo, Boutons et Barre de Marge).

7. Nommez les occurrences sur lesquelles l'utilisateur final devra cliquer pour déplacer la tête de lecture dans l'animation. Rappelons que le nom d'une occurrence ne doit pas commencer par une majuscule et ne peut contenir de caractères accentués ou spéciaux. N'utilisez pas non plus d'espace.
8. Placez l'instruction `stop()` sur l'image-clé de votre animation qui contient la page sommaire.

9. Utilisez ensuite le script ci-dessous, que vous devez répéter autant de fois que vous avez de boutons, pour déplacer la tête de lecture sur les différentes images-clés de votre animation.

```
btEcran1.addEventListener(MouseEvent.CLICK,allerImage10);

function allerImage10(evt:MouseEvent) {
  gotoAndStop(5);
}
```

Nous vous rappelons qu'il est fortement conseillé de laisser de nombreuses images vides entre les images-clés afin de pouvoir insérer d'éventuelles images-clés.

Conseil

Utilisez le raccourci clavier F6 pour pouvoir insérer des images-clés.

Symbole avec liaison placé sur la scène en AS (environnement auteur)

Pour réaliser la mise en pages d'une animation dans l'interface de Flash, vous devez construire tous les écrans, l'un après l'autre, faire glisser des symboles sur la scène et utiliser des outils de création. Mais dans certains cas, le placement de nombreuses occurrences sur la scène peut s'avérer fastidieux voire même impossible.

Pour pallier ce problème, la solution est extrêmement simple : utilisez des symboles avec liaison, accompagnés de l'opérateur `new` et de la méthode `addChild()`.

Une fois n'est pas coutume, sans même introduire cette notion ou vous la présenter, commençons par une démonstration, que nous vous conseillons de suivre en même temps en créant préalablement un nouveau fichier Flash.

1. Réalisez dans un clip l'animation d'une taupe qui sort de son trou (placez l'instruction `stop()` sur la dernière image-clé).
2. Effectuez un clic droit sur ce symbole dans la bibliothèque et sélectionnez la commande Liaisons.
3. Cochez la case Exporter pour ActionScript et donnez un nom de Classe (débutez ce nom par une majuscule sans utiliser de caractères accentués ou spéciaux, ni même d'espace).
4. À présent, nous devons placer ce clip sur la scène. Pour cela, nous utilisons le script ci-dessous sur l'image-clé 1 de notre animation, afin de générer dynamiquement une occurrence sur la scène.

Fichier de référence : `Chapitre6/TaupeLand.fla`

```
var taupe1 = new TaupeAnimee();
taupe1.x=300;
taupe1.y=200;
addChild(taupe1);
```

Sans le savoir, vous venez de suivre la procédure qui permet de placer un symbole dynamiquement sur la scène !

Grâce à ce code, à chaque lancement de l'animation, la taupe vient se placer au centre de la scène.

Revenons sur cette procédure, qu'il est indispensable de bien connaître, car c'est celle que vous devrez utiliser lorsque vous devrez placer un ou plusieurs symboles dynamiquement sur la scène. Prenons donc un nouvel exemple, celui d'une barre de menus, pour découvrir que la technique est toujours la même.

Avant de nous lancer dans une procédure pas à pas, il est important de comprendre qu'un symbole doit posséder un nom de classe pour pouvoir être placé sur la scène à l'aide de l'opérateur `new` et de la méthode `addChild()`. Dans l'exemple précédent, nous avions défini un nom de classe après avoir créé le symbole ; dans celui-ci, nous allons le faire au moment de la création.

Fichier de référence : `Chapitre6/ConstructionDynamique2`

1. Pour créer un nouveau symbole de type Clip, commencez par tracer sur la scène un trait réalisé au crayon (ou tout autre élément graphique). Il s'agit d'un trait de soulignement.
2. Ajoutez ensuite un texte dynamique au-dessus de ce trait et incorporez-y les caractères à utiliser. Employez le bouton Intégrer dans la palette Propriétés (consultez le chapitre 14 dédié au texte dans Flash). Nous donnons alors un nom d'occurrence au texte dynamique, comme le nom etiquette dans cet exemple.

> **Rappel**
>
> Incorporer les caractères à un texte dynamique ou de saisie ne sert qu'à s'assurer que la police que vous avez choisie sera bien celle que verra l'utilisateur final.

3. Sélectionnez le trait et le texte dynamique et pressez la touche de fonction F8.
4. Nommez le symbole et, point le plus important, cochez la case Exporter pour ActionScript puis donnez un nom de classe. Attention, ce nom doit, de préférence et par convention, commencer par une majuscule et ne peut surtout pas contenir de caractères spéciaux, accentués ou d'espace. Dans notre exemple, nous choisissons le nom de classe `BoutonMenuPrincipal`.

Pour finir, voici le script que nous avons placé sur l'image-clé 1 du scénario principal de l'animation.

```
var boutonParis:BoutonMenuPrincipal;
var boutonMilan:BoutonMenuPrincipal;
var boutonLondres:BoutonMenuPrincipal;

boutonParis = new BoutonMenuPrincipal ();
boutonMilan = new BoutonMenuPrincipal ();
boutonLondres = new BoutonMenuPrincipal ();
```

```
boutonParis.etiquette.text="Paris";
boutonMilan.etiquette.text="Milan";
boutonLondres.etiquette.text="Londres";

addChild(boutonParis);
addChild(boutonMilan);
addChild(boutonLondres);

boutonParis.x = 50;
boutonMilan.x = 210;
boutonLondres.x = 370;

boutonParis.y = boutonMilan.y = boutonLondres.y = 15;
```

Nous commençons par déclarer trois noms d'instances de la classe `BoutonMenuPrincipal` puis nous procédons à leur instanciation.

Souvenez-vous que le symbole contient un texte dynamique intitulé `etiquette` ; nous renseignons donc la propriété `text` de cette instance.

Pour finir, nous faisons appel à la méthode `addChild()` pour placer ces instances dans la liste d'affichage et nous définissons les valeurs des propriétés `x` et `y` de chaque instance.

Remarque

Si vous ne souhaitez pas ou ne savez pas importer une police dans une animation Flash, la technique que nous venons d'utiliser peut donc vous aider, car l'encapsulation d'une police se fait lors de l'étape d'intégration des caractères.

Script dans un fichier AS

Fichiers de référence : `Chapitre6/ConstructionDynamique3AS.fla` et `Chapitre6/mainDynamique.as`

Le script ci-dessous se trouve dans le fichier texte nommé `mainDynamique.as`. Il s'agit de la classe du document `ConstructionDynamique3AS.fla`

```
package {

  import flash.display.Sprite;

  public class mainDynamique extends Sprite {

    var boutonParis:BoutonMenuPrincipal;
    var boutonMilan:BoutonMenuPrincipal;
    var boutonLondres:BoutonMenuPrincipal;

    function mainDynamique() {

      boutonParis=new BoutonMenuPrincipal();
      boutonMilan=new BoutonMenuPrincipal();
      boutonLondres=new BoutonMenuPrincipal();
```

```
            boutonParis.etiquette.text="Paris";
            boutonMilan.etiquette.text="Milan";
            boutonLondres.etiquette.text="Londres";

            addChild(boutonParis);
            addChild(boutonMilan);
            addChild(boutonLondres);

            boutonParis.x=50;
            boutonMilan.x=210;
            boutonLondres.x=370;

            boutonParis.y=boutonMilan.y=boutonLondres.y=15;
        }
    }
}
```

Comme vous pouvez une fois de plus le constater, le code contenu dans ce fichier AS est identique à celui que nous utilisons dans le script de l'image-clé 1 du scénario principal de l'animation.

Supprimer une occurrence créée dynamiquement

Si vous souhaitez retirer de la scène une occurrence que vous avez placée dynamiquement à l'aide de la méthode `addChild()`, nous vous rappelons que vous devez utiliser `removeChild()`. Consultez la dernière partie du chapitre 2 qui présente cette méthode.

Création de tracés vectoriels et de textes dynamiques

Pour construire une interface, nous avons examiné, depuis le début de ce chapitre, de nombreux procédés. Voici deux nouvelles techniques qui vont venir compléter celles que nous connaissons déjà :

- La création et la gestion d'un texte à partir de classes dédiées (`TextField()`, `TextFormat()` et bien d'autres).
- La création de formes vectorielles à partir de classes dédiées (`Graphics()`, `Shape()` et bien d'autres).

Le chapitre 14 de ce livre traite du texte dans sa globalité ; référez-vous donc aux explications qui y sont données pour apprendre à maîtriser ce média.

Qu'est-il possible de réaliser à base de tracés vectoriels ?

La réponse à cette question et très simple : des formes vectorielles ! Cela sous-entend donc aussi bien de simples droites que des tracés plus complexes à base d'une multitude de courbes et de droites. Pour tracer des carrés, des rectangles, des rectangles à coins arrondis, des cercles et des ellipses, il existe déjà des méthodes prédéfinies (de la classe `Graphics()`), vous n'aurez donc pas besoin d'écrire de scripts et/ou de calculs de cosinus pour ces formes, comme nous le faisions en AS1/2.

Remarque

Celles et ceux qui connaissent les méthodes `moveTo()` et `lineTo()` en AS1/2, vont être surpris de découvrir que les techniques de manipulation des formes vectorielles n'ont pas changé en AS3. Vous allez simplement devoir ajouter la propriété `graphics` devant les méthodes que vous aviez l'habitude d'utiliser.

Définition

Un tracé vectoriel peut prendre n'importe quelle forme, mais il se caractérise avant tout par le fait qu'il possède un fond et/ou un contour. Tous deux peuvent être de couleur unie plus ou moins transparente, mais seul le fond peut être rempli d'un dégradé, d'un motif ou d'une image.

Il est souvent plus simple de tracer une forme à l'aide des outils de dessin prévus à cet effet dans la barre d'outils de l'interface de Flash, mais dans certains cas, il est impossible d'anticiper la position et l'aspect d'un tracé. Sans même parler de graphiques de représentation (diagrammes, histogrammes, camembert, etc.), il sera parfois nécessaire de tracer une simple droite ou une courbe entre deux points, d'entourer une zone de l'interface d'un cercle ou d'une autre forme, ou bien même d'encadrer une image sélectionnée. Vous avez peut-être déjà dû effectuer ces opérations et la seule solution que vous avez trouvée a été de créer un symbole placé dynamiquement sur la scène. Vous allez découvrir qu'avec quelques lignes de code, il est très simple de tracer une droite.

Tracer une droite

Fichier de référence : `Chapitre6/trace1.fla`

Sur la copie d'écran de la figure 6-8, nous vous présentons l'interface habituelle des exercices de ce livre, mais celle-ci est traversée par une ligne droite. Il s'agit d'un tracé vectoriel obtenu à l'aide du script ci-dessous :

Figure 6-8

Le trait vertical central sur la scène de cette animation a été tracé en ActionScript.

```
var ligneSeparation:Shape = new Shape();

ligneSeparation.graphics.lineStyle(4,0xFFFFFF,1);
ligneSeparation.graphics.moveTo(250,0);
ligneSeparation.graphics.lineTo(250,277);

addChild(ligneSeparation);
```

La création d'une forme vectorielle débute par l'instanciation de l'une des trois classes suivantes : `Shape()`, `MovieClip()` ou `Sprite()`. Cela vous permet ensuite d'y rattacher un tracé vectoriel. Dans notre exemple, dans la mesure où il s'agit d'une simple droite, nous ne pouvons définir qu'un style de trait mais pas d'attributs de fond.

```
ligneSeparation.graphics.lineStyle(4,0xFFFFFF,1);
```

Cette première méthode, qui définit les caractéristiques d'un trait, contient trois paramètres qui servent à configurer respectivement les trois points suivants :

- L'épaisseur du trait, exprimée en pixels.
- La couleur du trait, exprimée en valeur hexadécimale.
- Le pourcentage de transparence de la couleur du trait.

Remarque

Le pourcentage doit être défini à partir d'une base décimale : cela signifie que 1 représente 100 %, 0.1 représente 10 % et 0.5 représente 50 %.

Une fois l'aspect du trait défini, nous devons indiquer à Flash son point de départ. Pour cela, nous utilisons la méthode `moveTo()`. Nous vous rappelons que l'origine de la scène, les coordonnées `x` = 0 pixels et y = 0 pixels, se trouve dans le coin supérieur gauche.

Nous devons ensuite définir le point d'arrivée de la droite en faisant appel à la méthode `lineTo()`.

Attention

La méthode `lineTo()` nécessite deux paramètres qui spécifient les coordonnées d'un point (toujours par rapport au coin supérieur gauche de la scène) et non une distance.

Le mot `graphics` est une propriété commune aux deux classes `Sprite()` et `Shape()`, qui permet de faire référence à la classe `Graphics()`. Celle-ci possède les méthodes et propriétés qui permettent de gérer le dessin vectoriel sur la scène d'une animation (ou dans un conteneur). Précisons que la classe `MovieClip()` hérite des méthodes et propriétés de la classe `Sprite()`. Elle possède donc, par héritage, la propriété `graphics`.

Pour pouvoir faire référence aux méthodes `lineStyle()`, `moveTo()` et `lineTo()`, vous devez les préfixer du terme `graphics`.

Ce n'est pas terminé !

En effet, nous avons commencé notre script par la création d'une instance, mais nous ne l'avons pas encore ajoutée à la liste d'affichage : nous devons donc le faire à l'aide de la méthode `addChild()`.

Tracer une courbe

Fichier de référence : `Chapitre6/trace10.fla`

> **Remarque**
> Si ce n'est pas déjà fait, consultez les explications précédentes relatives à la construction d'une droite, avant de poursuivre.

La technique pour tracer une courbe est simple a mettre en œuvre mais un peu plus complexe à comprendre, surtout si vous ne maîtrisez pas les courbes de Bézier ; commençons donc par vous en rappeler sommairement le principe.

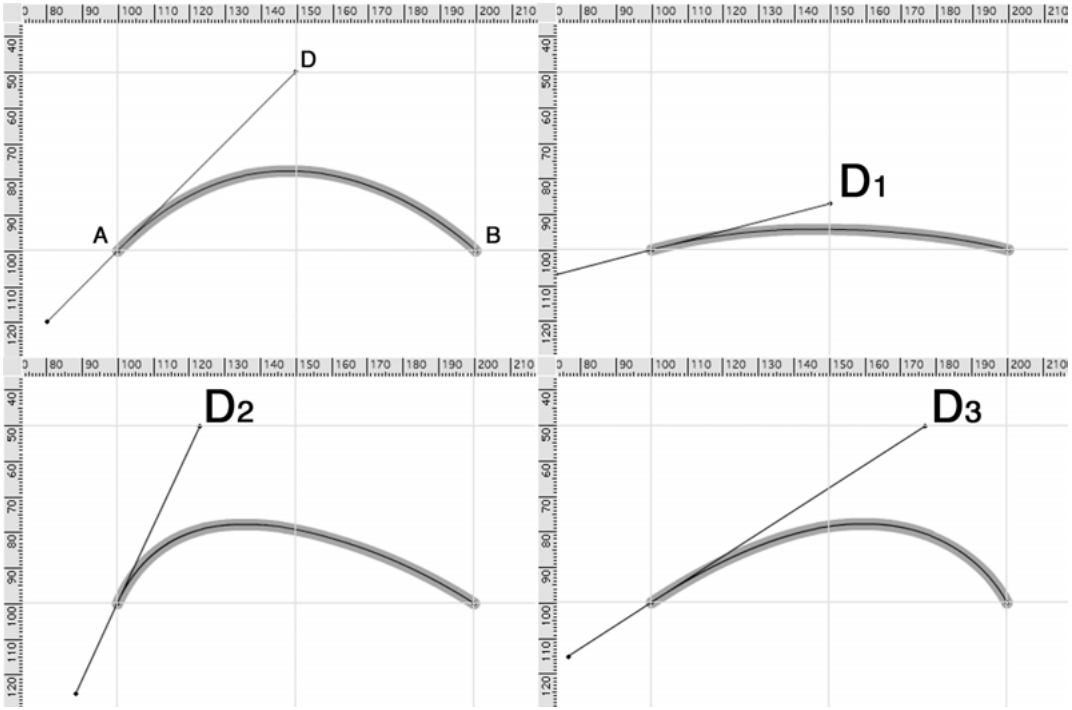


Figure 6-9

La méthode curveTo(), utilisée en ActionScript, fait appel aux courbes de Bézier pour construire un tracé qui n'est pas une droite.

Remarque

Aidez-vous du fichier `tracebeziers.fla` pour mettre en pratique les explications ci-dessous. Manipulez le point D de la figure 6-9 avec l'outil flèche blanche. Cliquez préalablement sur le point A pour faire apparaître la ligne directrice (définie par le segment allant du point A à D).

Pour obtenir une courbe, vous devez définir un point de départ et un point d'arrivée (nommés respectivement A et B sur la figure 6-9), puis préciser la direction de la courbure (D, D1, D2 et D3).

Premier constat : en fonction de la position du point D (appelé point directeur), la courbe change d'apparence. Vous devez simplement comprendre que plus ce point est éloigné du segment qui relie les points A et B, plus l'amplitude de la courbe est importante (comparez, sur la figure 6-9 les courbes avec les lettres D et D1).

Deuxième constat : pour préciser la direction de la courbe, vous devez rapprocher ou éloigner ce même point D de A ou B. Comparez les trois courbes qui possèdent les lettres D, D2 et D3. La courbe qui possède une courbure penchant vers la gauche a obligatoirement son point D2 qui se trouve lui-même plutôt vers la gauche (par rapport au point D).

L'éloignement du point directeur et sa position par rapport aux points A et B suffisent à définir la courbure d'un segment. Comment va-t-on pouvoir à présent traduire cette logique en ActionScript ? En utilisant la méthode `curveTo()`.

```
var courbe:Shape = new Shape();
courbe.graphics.lineStyle(1,0,1);
courbe.graphics.moveTo(100,100);
courbe.graphics.curveTo(150,75,200,100);
addChild(courbe);
```

Cette méthode possède quatre paramètres. Voici deux lignes d'instructions, basées sur la figure 6-9, pour vous aider à en comprendre la logique.

```
courbe.graphics.moveTo(A.x,A.y);
courbe.graphics.curveTo(D.x,D.y,B.x,B.y);
```

La méthode `moveTo()` fonctionne de la même façon que nous l'avons vu pour tracer une droite. En revanche, la méthode `curveTo()` possède, en quelque sorte, deux points. Les deux premières valeurs définissent les coordonnées x et y du point directeur, alors que les deux dernières définissent le point d'arrivée de la courbe.

Tracer un carré ou un rectangle

Fichier de référence : `Chapitre6/trace2.fla`

Comme nous l'évoquions précédemment, pour tracer un rectangle ou un carré, il n'est plus nécessaire de faire appel à plusieurs reprises à la méthode `lineTo()`, pour tracer une série de droites consécutives formant un quadrilatère. Il vous suffit tout simplement d'utiliser la méthode `drawRect()`.

Remarque

Si vous n'avez pas lu la technique qui permet de tracer une droite, référez-vous à ces explications pour comprendre le début du script ci-dessous.

```
var cadre:Shape = new Shape();

cadre.graphics.lineStyle(4,0xFFFFFF,1);
cadre.graphics.drawRect(70,50,250,125);

addChild(cadre);
```

La méthode parle d'elle-même il suffit d'indiquer des coordonnées, une largeur et une hauteur et le rectangle est tracé.

Rappelons que les coordonnées (0, 0) en x et en y se trouvent en haut à gauche de la scène. Ainsi, dans notre exemple, nous créons un rectangle de 250 pixels de largeur sur 125 pixels de hauteur, à 70 pixels du bord gauche et 50 pixels du haut de la scène.

Attention au piège !

Tant que vous n'avez pas besoin de contrôler ou connaître les coordonnées d'un rectangle que vous avez tracé sur la scène, vous ne rencontrerez aucun problème. En revanche, si vous souhaitez modifier sa position, il faudra adapter votre code de la façon suivante :

```
var cadre:Shape = new Shape();

cadre.graphics.lineStyle(4,0xFFFFFF,1);
cadre.graphics.drawRect(0,0,250,125);

addChild(cadre);

cadre.x=70;
cadre.y=50;
```

Si cela vous paraît illogique, essayons de comprendre ce qui se passe.

Lorsque nous plaçons dynamiquement un symbole ou un texte sur la scène, nous utilisons l'un des deux scripts suivants :

```
var monNom:Cartouche = new Cartouche ();
addChild(monNom);
```

ou

```
var monNom:TextField = new TextField();
monNom.text = "David";
addChild(monNom);
```

Quelles sont généralement les lignes que vous écrivez ensuite ? Ne serait-ce pas `monNom.x = une valeur` et `monNom.y = une valeur` pour positionner précisément votre création ? Dans cet exemple, nous n'avons pas placé ces lignes d'instructions au début du script ; voilà pourquoi il est nécessaire d'indiquer les valeurs `0` et `0` pour définir les coordonnées du tracé d'un rectangle.

Recentrer un carré ou un rectangle

Comme nous venons de vous le faire remarquer, la création d'un carré ou d'un rectangle se fait par rapport au conteneur ciblé avec la méthode `addChild()`. Afin de mieux comprendre cette dernière affirmation, essayez de faire tourner la forme que vous venez de créer. Vous visualiserez ainsi l'origne x et y du conteneur dans lequel vous ajoutez l'occurrence.

```
var cadre:Shape = new Shape();

cadre.graphics.lineStyle(4,0xFFFFFF,1);
cadre.graphics.drawRect(70,50,250,125);

addChild(cadre);

cadre.addEventListener(Event.ENTER_FRAME,tourner);

function tourner(evt:Event) {
  evt.currentTarget.rotation+=1;
}
```

Le constat est flagrant ! Lorsque la ligne d'instruction `addChild(cadre)` est exécutée, le conteneur d'objet d'affichage ciblé est, par défaut, l'animation. Cette dernière ayant ses coordonnées (0, 0) en haut à gauche de la scène, la forme est excentrée dès sa création. Voici le script que vous devriez utiliser pour créer un rectangle (ou tout autre forme fermée) dont le point d'alignement est au centre.

```
var cadre:Shape = new Shape();
cadre.graphics.lineStyle(1,0,1);
cadre.graphics.drawRect(-50,-50,100,100);
addChild(cadre);
cadre.x=150;
cadre.y=150;

cadre.addEventListener(Event.ENTER_FRAME,tourner);

function tourner(evt:Event) {
  evt.currentTarget.rotation+=1;
}
```

Tracer un cercle ou une ellipse

La technique pour tracer un cercle ou une ellipse, est aussi simple que celle pour créer un carré ou un rectangle. Il vous suffit de faire appel aux méthodes `drawCircle()` et `drawEllipse()`. Dans le cas de la première, vous devez définir des coordonnées x et y pour positionner le cercle, puis un troisième paramètre pour définir le rayon du cercle. Dans le cas d'une ellipse, vous devez définir deux rayons de tailles différentes.

```
var boutonRond:Shape = new Shape();

boutonRond.graphics.beginFill(0x666666,0.7);
boutonRond.graphics.drawCircle(300,200,50);

addChild(boutonRond);
```

Dans cet exemple, nous traçons un cercle gris foncé à 70 % d'opacité de 100 pixels de diamètre, soit 50 pixels de rayon, dont le centre est à 300 pixels du bord gauche de la scène et 200 du haut.

Ellipse

À la différence du cercle, la méthode `drawEllipse()` ne gère pas les coordonnées x et y de la même façon. Elles correspondent en effet au coin supérieur gauche du rectangle invisible dans lequel vient s'inscrire l'ellipse un fois tracée.

Régler les attributs d'une forme

Parmi les différents réglages d'une forme, on trouve l'épaisseur et le style du trait, mais également son remplissage. Commençons par découvrir qu'un trait ne se limite pas à une représentation d'un simple tracé d'un pixel de largeur de couleur noire.

Attributs de trait

Fichier de référence : `Chapitre6/trace9.fla`

Dans le premier exemple présenté, voici la ligne d'instruction que nous avions utilisée :

```
ligneSeparation.graphics.lineStyle(4,0xFFFFFF,1);
```

La méthode `lineStyle()` contient trois paramètres qui permettent de mettre en forme l'apparence d'un trait, mais il en existe d'autres qui sont utiles lorsque vous utilisez des épaisseurs plus importantes.

Modifier l'apparence des extrémités d'un tracé

```
trait.graphics.lineStyle(20,0xFFFFFF,0.7,false,LineScaleMode.NORMAL,CapsStyle.ROUND,
➥JointStyle.MITER);
```

Il s'agit de l'avant-dernier paramètre dans l'exemple ci-dessus ; il peut posséder trois valeurs :

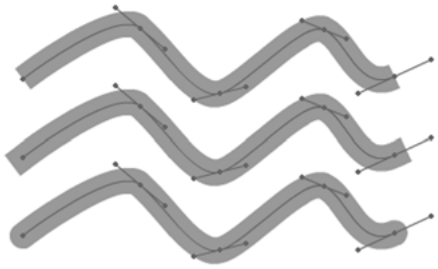
- `CapsStyle.NONE` (pas d'extrémité)
- `CapsStyle.SQUARE` (extrémité carré)
- `CapsStyle.ROUND` (extrémité ronde)

Nous nous permettons de vous le préciser de nouveau : ces attributs ne sont visibles qu'à partir du moment où vous avez défini une épaisseur de trait importante.

Par ailleurs, quelle différence fait-on précisément entre les constantes `NONE` et `SQUARE`. Voici une copie d'écran qui démontre la continuité (`CapsStyle.SQUARE`) du tracé au-delà du point directeur de la deuxième courbe de Bézier. En fonction de la position du point d'ancrage d'une extrémité par rapport à une autre, le paramètre `CapsStyle` permet d'effectuer des raccords proprement sans qu'il n'y ait un débordement ou un blanc.

Figure 6-10

Les trois types d'extrémités disponibles pour un trait : CapsStyle.NONE, CapsStyle.SQUARE et CapsStyle.ROUND.



Dans le fichier `trace9.fla`, le script ci-dessous permet de définir l'extrémité des deux raquettes du fameux jeu Pong (qui n'est pas fonctionnel dans notre exemple).

```
var raquette1:Shape = new Shape();
raquette1.graphics.lineStyle(20,0xFFFFFF,0.7,false,LineScaleMode.NORMAL,CapsStyle.
➥ROUND,JointStyle.MITER);
raquette1.graphics.moveTo(30,100);
raquette1.graphics.lineTo(30,180);
addChild(raquette1);

var raquette2:Shape = new Shape();
raquette2.graphics.lineStyle(20,0xFFFFFF,0.7,false,LineScaleMode.NORMAL,CapsStyle.
➥SQUARE,JointStyle.MITER);
raquette2.graphics.moveTo(470,100);
raquette2.graphics.lineTo(470,180);
addChild(raquette2);
```

Modifier l'apparence des angles d'un tracé ouvert ou fermé

Comme vous pouvez le constater sur la copie d'écran de la figure 6-11, l'extrémité des flèches diffère car nous avons utilisé trois valeurs différentes pour le dernier paramètre de la ligne d'instruction ci-dessous :

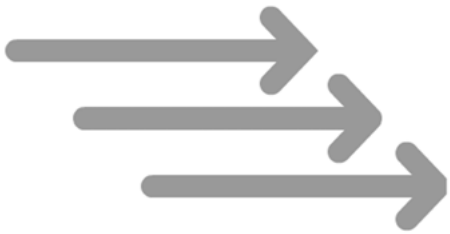
```
trait.graphics.lineStyle(20,0xFFFFFF,0.7,false,LineScaleMode.NORMAL,CapsStyle.ROUND,
➥JointStyle.MITER);
```

Vous avez le choix entre ces trois constantes :

- `JointStyle.MITER`
- `JointStyle.ROUND`
- `JointStyle.BEVEL`

Figure 6-11

Trois types d'extrémités permettent de définir l'apparence d'un trait.



Dans le fichier `trace9.fla`, vous découvrirez trois tracés relativement épais qui possèdent des angles comparables aux extrémités des flèches de la figure 6-11. Voici les scripts que nous avons utilisés.

```
var trajectoire1:Shape = new Shape();
trajectoire1.graphics.lineStyle(30,0xFFFFFF,0.7,false,LineScaleMode.NORMAL,CapsStyle.
➥SQUARE,JointStyle.ROUND);
trajectoire1.graphics.moveTo(100,50);
trajectoire1.graphics.lineTo(150,90);
trajectoire1.graphics.lineTo(200,50);
trajectoire1.graphics.lineTo(250,90);
trajectoire1.graphics.lineTo(300,50);
addChild(trajectoire1);

var trajectoire2:Shape = new Shape();
trajectoire2.graphics.lineStyle(30,0xFFFFFF,0.7,false,LineScaleMode.NORMAL,CapsStyle.
➥SQUARE,JointStyle.MITER);
trajectoire2.graphics.moveTo(100,125);
trajectoire2.graphics.lineTo(150,165);
trajectoire2.graphics.lineTo(200,125);
trajectoire2.graphics.lineTo(250,165);
trajectoire2.graphics.lineTo(300,125);
addChild(trajectoire2);

var trajectoire3:Shape = new Shape();
trajectoire3.graphics.lineStyle(30,0xFFFFFF,0.7,false,LineScaleMode.NORMAL,CapsStyle.
➥SQUARE,JointStyle.BEVEL);
trajectoire3.graphics.moveTo(100,200);
trajectoire3.graphics.lineTo(150,240);
trajectoire3.graphics.lineTo(200,200);
trajectoire3.graphics.lineTo(250,240);
trajectoire3.graphics.lineTo(300,200);
addChild(trajectoire3);
```

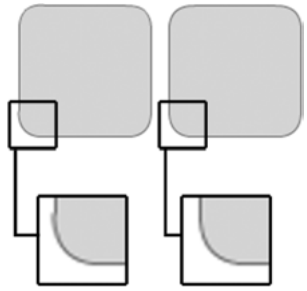
Une imperfection enfin corrigée

Pendant de nombreuses années, certaines courbes présentaient des imperfections de lissage ! Il nous est à présent possible de corriger ce défaut en réglant le quatrième paramètre à `true`.

```
carte.graphics.lineStyle(3,0xFFFFFF,0.7,true,LineScaleMode.NORMAL,CapsStyle.SQUARE,
➥JointStyle.BEVEL);
```

Figure 6-12

Dans certains cas, le lissage d'un contour peut présenter des imperfections.



Remplissage

Fichier de référence : `Chapitre6/trace3.fla`

Il existe une méthode très simple pour remplir une forme fermée qui s'intitule `beginFill()`. Vous devez indiquer deux paramètres : la valeur hexadécimale correspondant à la couleur et le niveau de transparence (valeur comprise entre 0 et 1, comme 0,65 pour 65 %).

Notons que cela fonctionne également pour des formes ouvertes, mais le remplissage se limite alors dans ce cas aux extrémités (le programme imagine une droite transparente entre ces dernières pour fermer la forme).

En partant de la création de formes (rectangle et cercle), voici comment vous pouvez remplir une forme :

```
var tapisJeu:Shape = new Shape();

tapisJeu.graphics.lineStyle(4,0xFFFFFF,1);
tapisJeu.graphics.beginFill(0x33aa33,1);
tapisJeu.graphics.drawRect(20,20,250,230);

addChild(tapisJeu);


var carteAJouer:Shape = new Shape();

carteAJouer.graphics.lineStyle(4,0xFFFFFF,1);
carteAJouer.graphics.beginFill(0xaa3333,0.7);
carteAJouer.graphics.drawRoundRect(250,40,120,180,20);

addChild(carteAJouer);


var boutonRond:Shape = new Shape();

boutonRond.graphics.beginFill(0x3333aa,0.7);
boutonRond.graphics.drawCircle(400,130,15);

addChild(boutonRond);
```

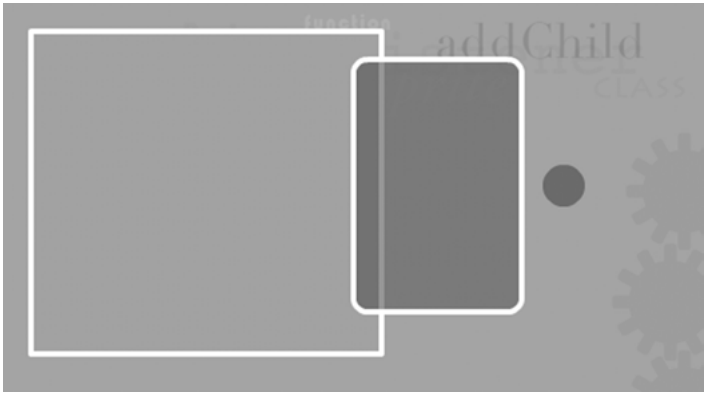


Figure 6-13

Une forme peut posséder un fond plus ou moins opaque, avec ou sans contour.

Vous noterez qu'une forme contenant une couleur n'est plus obligée de posséder de contour ; c'est le cas du cercle à droite de la figure 6-13.

Script dans un fichier AS

Fichiers de référence : `Chapitre6/trace5.fla` et `TracerMain.as`

Une fois encore, le cœur du code que nous utilisons dans le fichier AS externe est le même que celui que nous avons saisi dans la fenêtre Actions. Nous vous le présentons tout de même pour vous proposer un exemple et mettre en évidence la nécessité d'importer la classe `Graphics()` qui possède toutes les méthodes que nous n'avons pas cessé d'utiliser (`drawCircle()`, `lineTo()`, `moveTo()`, etc.).

```
package {

  import flash.display.Sprite;
  import flash.display.Graphics;


  public class TracerMain extends Sprite {

    public var carteAJouer:Sprite;

    function TracerMain() {

      carteAJouer = new Sprite();

      carteAJouer.graphics.lineStyle(4,0xFFFFFF,1);
      carteAJouer.graphics.beginFill(0xFFFFFF,0.2);
      carteAJouer.graphics.drawRoundRect(250,40,120,180,20);

      addChild(carteAJouer);
    }
  }

}
```

Tracer des formes en fonction des événements souris

Fichier de référence : `Chapitre6/trace6.fla`

Jusqu'à présent, toutes les formes que nous avons créées ont été tracées sur la scène sans que nous ayons besoin d'intervenir à l'aide du clavier ou de la souris. Nous souhaiterions à présent pouvoir tracer un cercle ou un rectangle au moment de l'exécution du fichier SWF et non au moment de la réalisation de l'animation.

Prenons l'exemple d'un cercle ou d'un rectangle que vous auriez besoin de tracer à l'aide de la souris. Vous devez non seulement prévoir l'utilisation d'un gestionnaire, mais aussi

définir l'instant de son déclenchement. Regardons tout de suite cet exemple qui est particulièrement intéressant pour de nombreuses raisons.

```
var cadre:Shape = new Shape();
addChild(cadre);

var pointDepartX:Number;
var pointDepartY:Number;

var ecartLargeur:Number;
var ecartHauteur:Number;

stage.addEventListener(MouseEvent.MOUSE_DOWN,creerRectangle);
stage.addEventListener(MouseEvent.MOUSE_UP,annulerMouseMove);

function annulerMouseMove(evt:MouseEvent) {
  stage.removeEventListener(MouseEvent.MOUSE_MOVE,tracerRectangle);
}

function creerRectangle(evt:MouseEvent) {

  pointDepartX = mouseX;
  pointDepartY = mouseY;

  stage.addEventListener(MouseEvent.MOUSE_MOVE,tracerRectangle);
}

function tracerRectangle(evt:MouseEvent) {

  ecartLargeur=mouseX-pointDepartX;
  ecartHauteur=mouseY-pointDepartY;

  cadre.graphics.clear();
  cadre.graphics.lineStyle(2,0x222222,0.6);

  cadre.graphics.drawRect(pointDepartX,pointDepartY,ecartLargeur,ecartHauteur);
}
```

Nous commençons par créer une instance de la classe `Shape()` pour pouvoir définir un style de trait et lui appliquer la méthode `drawRect()` ultérieurement dans le script.

Les quatre variables que nous déclarons vont nous servir, comme leurs noms l'indiquent, à mémoriser l'emplacement du clic au moment où l'utilisateur s'apprête à tracer la forme et stocker le calcul de la largeur et de la hauteur du quadrilatère.

Attention : vous ne devez pas demander au lecteur Flash de chercher à tracer la forme (rectangle ou carré) dès que la souris bouge. Cela ne doit être effectué qu'à partir du moment où un clic a été défini sur la scène. C'est la raison pour laquelle l'objet d'écoute qui fait appel à l'événement `MOUSE_MOVE` se trouve dans la fonction de rappel de l'objet d'écoute qui fait lui-même appel à l'événement `MOUSE_DOWN`. Afin de terminer le tracé du

quadrilatère, l'écouteur gérant l'événement `MOUSE_MOVE` doit être détruit au moment où l'utilisateur relâche le bouton de la souris.

La réalisation d'un rectangle qui change de taille en continu lorsque le curseur de la souris bouge est possible grâce à la méthode `clear()`, qui efface le dernier tracé (réalisé à l'aide de la méthode `drawRect()`) avant qu'un nouveau soit redéfini (les deux dernières lignes du script).

Pour obtenir une création de forme en partant du centre (depuis l'emplacement du clic de la souris), adaptez la fonction `tracerRectangle()` comme ceci :

```
function tracerRectangle(evt:MouseEvent) {

  decalCentreX = 0;
  decalCentreY = 0;

  ecartLargeur=mouseX-pointDepartX;
  ecartHauteur=mouseY-pointDepartY;

  if ( evt.shiftKey) {
      decalCentreX = Math.abs(ecartLargeur/2);
      decalCentreY = Math.abs(ecartHauteur/2);
  }
    cadre.graphics.clear();
    cadre.graphics.lineStyle(2,0x222222,0.6);

    cadre.graphics.drawRect(pointDepartX-decalCentreX,pointDepartY-decalCentreY,
    ➥ecartLargeur,ecartHauteur);

}
```

Sprite, Shape ou MovieClip ?

La principale différence entre ces trois classes réside dans l'importance de leurs propriétés et de leurs méthodes. Prenons un exemple précis : vous souhaiteriez créer un rectangle aux coins arrondis, comme une carte à jouer ; vous devez, dans ce cas, créer une instance de la classe `Sprite()` ou `MovieClip()`, car la classe `Shape()` ne possède pas la méthode `startDrag()`.

D'une façon générale, nous pouvons dire qu'il est conseillé d'utiliser la classe `Shape()` pour tracer des formes vectorielles dans le but d'habiller la scène. En revanche, si vous souhaitez créer un élément graphique pour pouvoir le contrôler, nous vous conseillons d'utiliser les classes `Sprite()` ou `MovieClip()`.

Rappel

Une occurrence de Sprite ne possède pas de timeline contrairement à celle d'un MovieClip.

Utiliser un tableau

Fichier de référence : `Chapitre6/trace7.fla`

Pour réaliser des formes bien plus complexes que celles que nous avons pu découvrir dans ces explications, vous pouvez stocker les coordonnées de points à joindre pour former un tracé. Dans la pratique, le mieux serait de stocker des données dans un fichier XML. Dans notre exemple, nous avons utilisé un tableau, ce qui a le mérite d'être plus facile à créer.

> **Remarque**
> Ouvrez le fichier `trace7.fla` pour découvrir l'ensemble des données réellement contenues dans le tableau `coord`.

```
var coord=[248.25,38.1,…,248.25,38.55];

var contourFrance:Shape = new Shape();
addChild(contourFrance);

contourFrance.graphics.lineStyle(1,0x474763,1);
contourFrance.graphics.moveTo(coord[0],coord[1]);

for (var i:Number=2; i<coord.length-1; i+=2) {
  contourFrance.graphics.lineTo(coord[i],coord[i+1]);
}

contourFrance.y = -20;
contourFrance.scaleX = contourFrance.scaleY = 0.75;
```

Le résultat de ce script aboutit au tracé de la carte de France.

Figure 6-14

Cette forme est obtenue à partir d'une multitude de points stockés dans un tableau et d'une série de lineTo().

Réaliser un tracé progressivement

Fichier de référence : `Chapitre6/trace8.fla`

Nous sommes partis du script du fichier `trace7.fla` que nous avons adapté. Pour la temporisation du tracé des nombreuses droites qui composent le contour de la France, nous avions principalement deux solutions : ajouter une fonction de rappel appelée par un écouteur gérant l'événement `ENTER_FRAME` ou bien créer une instance de la classe `Timer()`. C'est cette dernière solution que nous avons retenue.

```
var contourFrance:Shape = new Shape();
addChild(contourFrance);

contourFrance.graphics.lineStyle(1,0x474763,1);
contourFrance.graphics.moveTo(coord[0],coord[1]);

contourFrance.y = -20;
contourFrance.scaleX = contourFrance.scaleY = 0.75;

var compteur:Timer = new Timer(10,coord.length/2);
compteur.addEventListener(TimerEvent.TIMER,tracerFrance);
var entree:Number = 2;

function tracerFrance (evt:Event) {
  contourFrance.graphics.lineTo(coord[entree],coord[entree+1]);
  entree+=2;
  if (entree==coord.length) compteur.stop();

}

compteur.start();
```

Conclusion

Ne négligez surtout pas le travail des formes vectorielles pour réaliser certaines parties de vos interfaces. Vous gagnerez non seulement en poids de fichier (aspect plus ou moins négligeable selon la nature de vos animations), mais surtout en temps de développement et en souplesse de déploiement de certaines fonctionnalités.

Importation d'images

Nous n'allons pas aborder tout de suite la technique d'importation d'une image sur la scène à partir de lignes d'instructions en AS. Mais, comme ce chapitre explique clairement les avantages et les inconvénients des différentes méthodes de construction d'une interface, nous nous devions, tout de même, de consacrer un petit paragraphe aux avantages d'une importation dynamique de photos sur la scène.

Jusqu'à présent, vous utilisiez peut-être la commande Importer du menu Fichier pour placer des images sur la scène. En dehors du fait que cela augmente très rapidement le poids de vos fichiers SWF, vous avez peut-être pu remarquer, ou vous remarquerez, qu'il n'est pas très facile de mettre à jour des animations gérant les photos de cette façon.

Pour pallier ces deux inconvénients, il vous suffit de charger des images sur la scène à partir de lignes d'instructions. Cela présente l'avantage de pouvoir substituer une image à une autre pour mettre à jour une animation qui charge une photo à partir des classes `Loader()` et `URLRequest()`.

N'avez-vous jamais eu besoin de placer des photos sur la scène sous forme de quadrillage ? Un petit tableau de 36 photos (six lignes de six colonnes) ! En ActionScript, il vous suffit de quelques lignes de code, alors qu'à partir de l'interface de Flash, il vous faudra quelques très longues minutes !

Consultez le chapitre 12 pour découvrir le code nécessaire au chargement d'une image.

Instanciation de classes personnalisées

Comme vous avez pu le lire rapidement au début de ce chapitre, il est possible de créer le contenu d'une animation en faisant appel à des fichiers externes. Les lignes d'instructions ci-dessous le prouvent et permettent d'obtenir un texte sur la scène sans avoir besoin de le créer à partir de l'outil dédié disponible dans l'interface de Flash.

```
var zoneTexte=new TextField();
zoneTexte.text="Bonjour";
addChild(zoneTexte);
```

Comment pourrions-nous éviter de copier-coller ces trois lignes si nous souhaitions créer de nombreux textes, comme pour générer des titres, des paragraphes, etc. ?

La solution est tout aussi simple à comprendre qu'à mettre en œuvre : vous devez créer une classe ! Dans le chapitre 16, nous proposons une première approche du travail avec les classes à travers deux exemples précis ; le premier présente une classe qui permet d'effectuer une conversion de nombre, alors que le deuxième rattache une classe à un symbole. Ni l'un ni l'autre ne génère de contenu sur la scène. Dans les lignes qui vont suivre, nous allons apprendre à créer une classe que nous allons ensuite instancier dans le but d'obtenir une occurrence sur la scène.

Rappelons que la programmation d'une animation qui fait appel à des classes présente de nombreux avantages, dont le principal est de pouvoir réutiliser le code contenu dans un fichier AS. Au chapitre 1, dans la partie La programmation orientée objet, nous avions découvert le principe et la méthode pour développer proprement. Allons à présent plus loin et réalisons ensemble une classe chargée de placer un texte sur la scène.

Fichiers de référence : `Chapitre6/creationTexte.fla`, `Chapitre6/CreationTexteMain.as` et `Chapitre6/CreationTexte.as`

Pour commencer, examinez en détail la construction de cette interface.

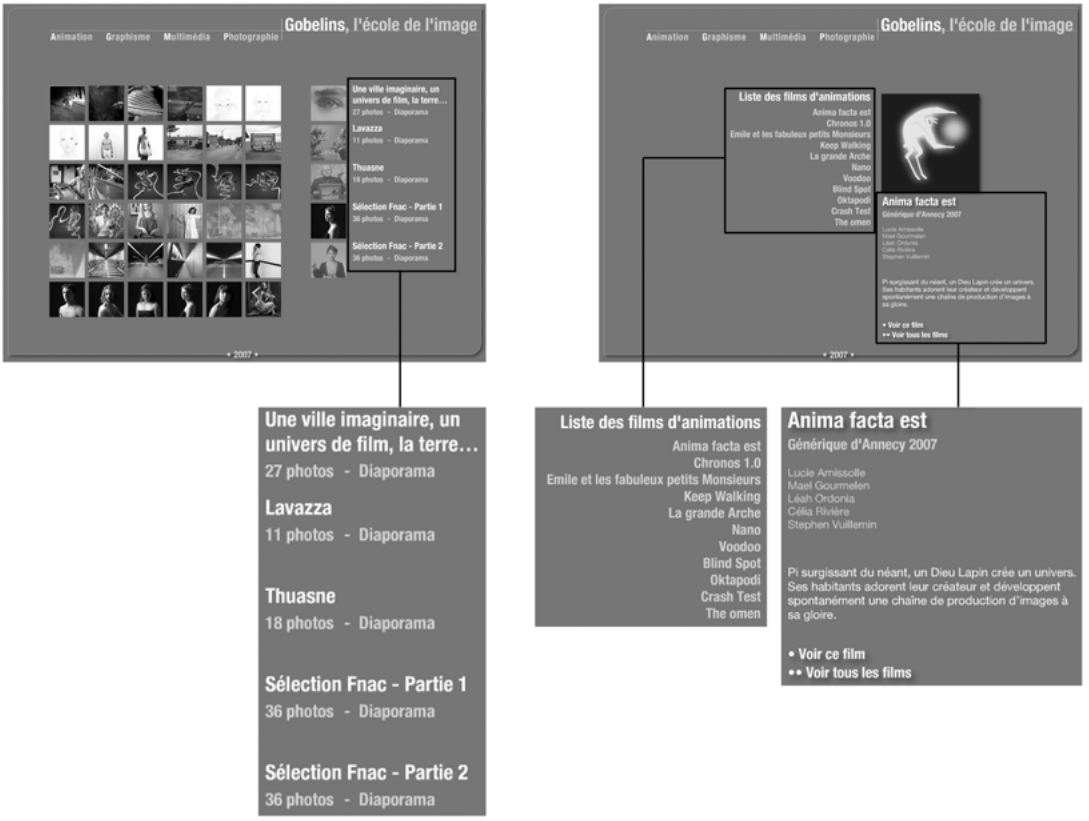


Figure 6-15

Textes obtenus à partir d'une seule classe.

Les textes, tous différents les uns des autres, ont été placés sur la scène à partir d'une seule ligne d'instruction.

```
titre = new CreationTexte("Liste des films d'animations" ,1);
```

Dans cet exemple, nous avons deux paramètres ; nous éviterons pour le moment de commenter le deuxième, car il sera traité lors de l'analyse du dernier exemple. Si vous observez bien les copies d'écran de la figure 6-15, vous noterez que certains textes n'ont sûrement pas été saisis en paramètre de la classe `CreationTexte()`. C'est effectivement le cas pour les titres des films, les noms des étudiants, les pitchs et les noms des séries de photos. Ces informations ont été extraites d'un fichier XML et non renseignées directement.

```
var nomFilm:String=donneesXML.Animation.Film[numeroFilm].@nom.toString();
titreLegendeVignette=new CreationTexte(nomFilm,300,3);
```

Ne tenons pas compte de cela et commençons tout de suite.

La structure des fichiers

Nous avons commencé par créer un fichier Flash (.fla), puis nous l'avons enregistré (`creationTexte.fla`). Nous avons ensuite créé la classe du document : il s'agit d'un fichier AS enregistré sous le nom `CreationTexteMain.as` (nous n'avons pas pu le nommer `main.as` car un autre fichier portant le même nom se trouvait déjà dans le dossier intitulé `Chapitre6`). Nous avons terminé notre première manipulation en saisissant le mot `CreationTexteMain` dans la zone de saisie (Classe du document) située en bas à droite de la palette Propriété du fichier `creationTexte.fla`.

À partir de là, nous pouvons dire que nous avons une construction basique et standard d'un projet Flash.

Figure 6-16
Un fichier Flash et sa classe du document

Nous terminons cette première phase en créant un deuxième fichier AS qui va justement contenir notre classe `CreationTexte()` et nous le nommons `CreationTexte.as`. Il ne nous reste plus qu'à saisir les lignes d'instructions contenues dans ces fichiers.

Contenu des fichiers AS

Le fichier `CreationTexteMain.as` contient le script ci-dessous :

```
package {

  import flash.display.Sprite;

  public class CreationTexteMain extends Sprite {

    public static  var titre1:CreationTexte;
    public static  var titre2:CreationTexte;

    function CreationTexteMain() {
      titre1 = new CreationTexte("Paris");
      titre2 = new CreationTexte("Londres");

      addChild(titre1);
      titre1.y = 50;

      addChild(titre2);
      titre2.y = 100;

      titre1.x = titre2.x = 25;
    }
  }

}
```

Comme vous pouvez le constater, le script est extrêmement simple : il se limite à la création de deux instances de la classe `CreationTexte()`. Pour les néophytes en matière de programmation orientée objet, nous vous rappelons qu'il est vivement conseillé de lire la fin du chapitre 1 de ce livre pour mieux appréhender la structure d'un fichier AS. Revenons tout de même sur cette petite analyse.

Nous importons la classe `Sprite()` car la classe `CreationTexteMain()` est étendue à partir de celle-ci.

Rappel

Étendre une classe permet d'en créer une nouvelle (`CreationTexteMain()`) sans être obligé de redéfinir toutes ses propriétés et méthodes, car elle hérite déjà de celle que vous étendez (`Sprite()`). Ainsi, dans notre exemple, la classe `CreationTexteMain()` dispose déjà de toutes les propriétés et méthodes de la classe `Sprite()`.

Nous vous rappelons ensuite que le nom de la classe doit être précédé du terme `public`, car cela nous permet justement de faire référence à ce mot (cette classe et son contenu) à partir d'autres fichiers AS. Avant de créer le constructeur (`function CreationtexteMain(){}`), nous déclarons deux nouvelles instances de la classe `CreationTexte()`, c'est-à-dire que nous informons le programme de l'existence prochaine de deux instanciations de la classe `CreationTexte()`.

Maintenant, dans le constructeur, qui doit porter le même nom que celui de la classe, nous instancions tout simplement la classe `CreationTexte()`, puis nous ajoutons ces deux instances à la liste d'affichage.

Ce premier fichier AS est très simple pour un développeur dont les connaissances en POO sont acquises, car nous instancions tout simplement la classe `CreationTexte()` dont voici le contenu (fichier `CreationTexteMain.as`).

```
package {

  import flash.display.Sprite;
  import flash.text.*;

  public class CreationTexte extends Sprite {

    public var zoneTexte:TextField;

    public function CreationTexte(expression:String) {

      zoneTexte=new TextField();
      zoneTexte.text=expression;
      addChild(zoneTexte);

    }

  }
}
```

Vous noterez que le script est court car il fait très peu de choses : il se contente de créer un texte dynamique. Observez bien le cœur de la classe. Il s'agit des mêmes lignes d'instructions que celles que nous avions utilisées en introduction à ce développement, Instanciation de classes personnalisées.

Dans cette classe, nous importons (`import flash.text.*`) toutes les sous-classes du package `text`, car nous allons ajouter des fonctionnalités dans l'exemple qui va suivre.

Dans le constructeur, nous commençons donc par déclarer une instance intitulée `zoneTexte` de type `TextField`, que nous initialisons.

Nous ne devons pas oublier de faire appel à la méthode `addChild()` qui va afficher l'instance `zoneTexte` dans les conteneurs d'objet d'affichage `titre1` et `titre2`.

Voilà ! En exécutant le fichier `creationTexte.fla`, vous découvrirez que le fichier AS `CreationTexteMain.as` fait appel à celui qui s'intitule `CreationTexte.as`. Vous constaterez tout de même que le texte n'est pour l'instant pas mis en forme, ce à quoi nous allons tenter de remédier.

Deuxième exemple

Fichiers de référence : `Chapitre6/creationTitre.fla`, `Chapitre6/CreationTitreMain.as` et `Chapitre6/CreationTitre.as`

Dans ce deuxième exemple, nous avons tout simplement reproduit le code que nous avions écrit dans le premier exemple pour générer deux fichiers AS et un .fla. Nous avons également adapté le nom de nos classes puis changé le contenu des fichiers AS.

> **Remarque**
>
> Si, pour gagner du temps, vous dupliquez les fichiers du premier exemple, n'oubliez pas de changer le nom de la classe du document dans le fichier `creationTitre.fla`.

Voici les lignes d'instructions contenues dans le fichier `CreationTitreMain.as`

```
package {

  import flash.display.Sprite;

  public class CreationTitreMain extends Sprite {

    public static  var titre1:CreationTitre;
    public static  var titre2:CreationTitre;

    function CreationTitreMain() {
      titre1 = new CreationTitre("La ville Lumière");
      titre2 = new CreationTitre("se trouve aux pays des peintres …");

      addChild(titre1);
      titre1.y = 50;
```

```
        addChild(titre2);
        titre2.y = 100;

        titre1.x = titre2.x = 25;
      }
    }

}
```

Vous noterez qu'il n'y a aucun changement majeur : nous avons simplement modifié le nom des classes. En revanche, nous avons ajouté de nombreuses lignes dans le fichier `CreationTitre.as` :

```
package {


  import flash.display.Sprite;
  import flash.text.*;

  public class CreationTitre extends Sprite {

    public var zoneTexte:TextField;
    private var styleEnTitre:TextFormat;


    public function CreationTitre(expression:String) {

      zoneTexte=new TextField();
      zoneTexte.name="zoneTexte";

      zoneTexte.selectable=false;
      zoneTexte.text=expression;
      zoneTexte.autoSize=TextFieldAutoSize.LEFT;
      zoneTexte.embedFonts=true;

      styleEnTitre=new TextFormat();
      styleEnTitre.color="0x474763";
      styleEnTitre.size=24;
      styleEnTitre.align=TextFormatAlign.RIGHT;
      styleEnTitre.font="Helvetica Neue Bold Condensed";
      zoneTexte.setTextFormat(styleEnTitre);

      addChild(zoneTexte);

    }

  }
}
```

Le début de la classe est inchangé, à l'exception de l'import des sous-classes du package `text`.

Note

Si vous ne connaissez pas encore les propriétés et autres méthodes liées à la mise en forme du texte, consultez le chapitre 14 ; toutes ces notions y sont expliquées.

Nous avons ensuite instancié la classe `TextFormat()` pour définir les valeurs de toutes les propriétés utilisées dans notre style (`color`, `size`, `align` et `font`).

Comme vous pouvez le remarquer, la méthode `setTextFormat()` sert à appliquer la mise en forme à l'instance de type `TextField()`, mais une fois encore, cela ne suffit peut-être pas.

Si vous ne devez créer qu'un seul type de texte dans une animation, cette classe peut vous suffire. En revanche, dans le cas où vous souhaiteriez gérer plusieurs styles différents, il faudrait la développer davantage.

Il serait très simple d'ajouter des paramètres dans le constructeur de notre classe, mais nous risquerions d'aboutir à des mises en forme de textes irrégulières.

Revenons sur ce dernier propos et tentons de l'expliquer. Lorsque vous effectuez une mise en pages, vous devez respecter une charte graphique qui se définit par l'usage d'un code couleur, l'utilisation d'une certaine iconographie et le choix d'une typographie accompagnée de différents styles. Comme vous pouvez le constater sur la copie d'écran de la figure 6-15, les textes utilisent tous la même police de caractères, mais pas la même graisse ni le même corps. Chaque style correspond à une définition particulière. Voici d'ailleurs les lignes d'instructions qui ont permis d'aboutir à ce résultat (elles sont sorties de leur contexte, mais l'objectif de cette présentation est de vous faire prendre conscience de la nécessité de créer des styles et non une mise en forme ponctuelle à l'occasion d'un nouveau besoin).

```
public function styleTitre() {

      styleEnTitre=new TextFormat();
      styleEnTitre.color="0xFFFFFF";
      styleEnTitre.size=24;
      styleEnTitre.align = TextFormatAlign.RIGHT;
      styleEnTitre.font="Helvetica Neue Bold Condensed";
      zoneTexte.setTextFormat(styleEnTitre);

    }
    public function styleListe() {

      styleEnTitre=new TextFormat();
      styleEnTitre.color="0xCCCCCC";
      styleEnTitre.size=19;
      styleEnTitre.align = TextFormatAlign.RIGHT;
      styleEnTitre.font="Helvetica Neue Bold Condensed";
      zoneTexte.setTextFormat(styleEnTitre);

      zoneTexte.addEventListener(MouseEvent.CLICK,ligneCliquee);
    }
```

```
public function styleTitreLegende() {

  styleEnTitre=new TextFormat();
  styleEnTitre.color="0xFFFFFF";
  styleEnTitre.size=24;
  styleEnTitre.align = TextFormatAlign.LEFT;
  styleEnTitre.font="Helvetica Neue Bold Condensed";
  zoneTexte.setTextFormat(styleEnTitre);
}
public function styleCategorieLegende() {

  styleEnTitre=new TextFormat();
  styleEnTitre.color="0xCCCCCC";
  styleEnTitre.size=16;
  styleEnTitre.align = TextFormatAlign.LEFT;
  styleEnTitre.font="Helvetica Neue Bold Condensed";
  zoneTexte.setTextFormat(styleEnTitre);
}
public function styleEtudiantsLegende() {

  styleEnTitre=new TextFormat();
  styleEnTitre.color="0xCCCCCC";
  styleEnTitre.size=12;
  styleEnTitre.align = TextFormatAlign.LEFT;
  styleEnTitre.font="Helvetica Neue";
  zoneTexte.setTextFormat(styleEnTitre);
}

public function stylePitchFilm() {

  styleEnTitre=new TextFormat();
  styleEnTitre.color="0xFFFFFF";
  styleEnTitre.size=13;
  styleEnTitre.align = TextFormatAlign.LEFT;
  styleEnTitre.font="Helvetica Neue";
  zoneTexte.wordWrap = true;
  zoneTexte.setTextFormat(styleEnTitre);
}

public function styleVoirLeFilm() {

  styleEnTitre=new TextFormat();
  styleEnTitre.color="0xFFFFFF";
  styleEnTitre.size=16;
  styleEnTitre.align = TextFormatAlign.CENTER;
  styleEnTitre.font="Helvetica Neue Bold Condensed";
  zoneTexte.setTextFormat(styleEnTitre);
}

public function styleListePhoto() {

  styleEnTitre=new TextFormat();
  styleEnTitre.color="0xFFFFFF";
```

```
        styleEnTitre.size=19;
        styleEnTitre.align = TextFormatAlign.LEFT;
        styleEnTitre.font="Helvetica Neue Bold Condensed";
        zoneTexte.multiline = true;
        zoneTexte.wordWrap = true;
        zoneTexte.setTextFormat(styleEnTitre);

      zoneTexte.addEventListener(MouseEvent.CLICK,ligneCliquee);
      }
```

Lorsqu'une personne doit faire appel à une classe qu'elle n'a pas conçue, elle ne peut alors choisir qu'entre les différents styles prédéfinis dans la charte graphique. Cela permet de garder une certaine cohérence dans la mise en forme d'un écran.

Nous allons donc devoir modifier une troisième et dernière fois notre classe `CreationTexte()` ou `CreationTitre()` afin de la compléter.

Troisième exemple

Fichiers de référence : `Chapitre6/texteEnForme.fla`, `Chapitre6/TexteEnFormeMain.as` et `Chapitre6/TexteEnForme.as`

Pour cette dernière série de trois fichiers, nous sommes de nouveau partis des fichiers précédents que nous avons modifiés comme auparavant (changement du nom des fichiers et des classes, sans oublier le nom de la classe de document dans la palette Propriétés du fichier `texteEnForme.fla`).

Voici donc le contenu des deux fichiers AS.

```
package {

  import flash.display.Sprite;

  public class TexteEnFormeMain extends Sprite {

    public static  var titre1:TexteEnForme;
    public static  var titre2:TexteEnForme;

    function TexteEnFormeMain() {
      titre1 = new TexteEnForme("Le soleir de la mer",1);
      titre2 = new TexteEnForme("par Paul GreenBegmann",2);

      addChild(titre1);
      titre1.y = 50;

      addChild(titre2);
      titre2.y = 80;

      titre1.x = titre2.x = 25;
    }
  }

}
```

et

```
package {


  import flash.display.Sprite;
  import flash.events.MouseEvent;
  import flash.text.*;

  public class TexteEnForme extends Sprite {

    public   var zoneTexte:TextField;
    private var styleEnTitre:TextFormat;


    public  function TexteEnForme(expression:String,numeroStyle:Number) {

      zoneTexte=new TextField();
      zoneTexte.name = "zoneTexte";
      zoneTexte.selectable = false;
      zoneTexte.text=expression;
      zoneTexte.autoSize= TextFieldAutoSize.LEFT;
      zoneTexte.embedFonts=true;

      addChild(zoneTexte);

      switch (numeroStyle) {

        case 1 :
          styleTitre();
          break;

        case 2 :
          styleAuteur();
          break;
      }
    }
    public function styleTitre() {

      styleEnTitre=new TextFormat();
      styleEnTitre.color="0xFFFFFF";
      styleEnTitre.size=24;
      styleEnTitre.align = TextFormatAlign.RIGHT;
      styleEnTitre.font="Helvetica Neue Bold Condensed";
      zoneTexte.setTextFormat(styleEnTitre);

    }
    public function styleAuteur() {

      styleEnTitre=new TextFormat();
      styleEnTitre.color="0xDDDDDD";
```

```
        styleEnTitre.size=14;
        styleEnTitre.align = TextFormatAlign.RIGHT;
        styleEnTitre.font="Helvetica Neue Bold Condensed";
        zoneTexte.setTextFormat(styleEnTitre);
      }

    }
}
```

En comparant avec la classe précédente, vous constaterez que nous n'avons modifié qu'une seule ligne du fichier `TexteEnFormeMain.as`.

```
titre1 = new TexteEnForme("Le soleir de la mer",1);
titre2 = new TexteEnForme("par Paul GreenBegmann",2);
```

Nous avons simplement ajouté un paramètre lors de l'instanciation de la classe et, de ce fait, cela sous-entend que nous avons modifié la classe `TexteEnForme()`. Le changement est flagrant et se caractérise par les lignes suivantes :

```
switch (numeroStyle) {

  case 1 :
    styleTitre();
    break;

  case 2 :
    styleAuteur();
    break;
  }
```

En fonction de la valeur du paramètre précisé lors de l'instanciation de la classe `TexteEnForme()`, les fonctions `styleTitre()` ou `styleAuteur()` sont appelées. Ces dernières sont similaires aux lignes d'instructions exposées précédemment :

```
public function styleTitre() {
    styleEnTitre=new TextFormat();
    styleEnTitre.color="0xFFFFFF";
    styleEnTitre.size=24;
    styleEnTitre.align = TextFormatAlign.RIGHT;
    styleEnTitre.font="Helvetica Neue Bold Condensed";
    zoneTexte.setTextFormat(styleEnTitre);
  }
```

Grâce à cette technique, vous pouvez définir des styles bien précis pour une charte graphique et faciliter ainsi sa mise en œuvre.

Conclusion

Il peut paraître fastidieux de devoir générer des nombreuses lignes de code pour aboutir à une action assez simple, mais c'est en pratiquant les classes externes que vous réaliserez à quel point cette technique de développement d'interface s'avère efficace.

Sans l'utilisation des classes, la réalisation de certains projets deviendrait très rapidement inintéressante à cause de tâches répétitives. Si vous n'êtes pas habitué à travailler avec des classes externes, effectuez de nombreuses tentatives pour vous entraîner : un tel investissement de temps sera très rapidement bénéfique.

7

Les variables

Par définition, une variable est un contenant dont le contenu est susceptible de varier. Les variables sont présentes dans la plupart des animations. Dans la vie courante, nous manipulons continuellement des notions ou des objets comparables à des variables. Nous substituons bien souvent un ou plusieurs mots à un ou plusieurs autres pour nous représenter précisément quelque chose ou quelqu'un.

Qui n'a jamais vu les quatre lignes ci-dessous inscrites sur un tableau d'école ?

a=2

b=3

c=a+b

Combien vaut c ?

Sans le savoir, vous avez déjà manipulé des variables ! Le calcul de c vous semble évident car vous substituez aux lettres a et b les valeurs 2 et 3. Les trois lettres a, b et c sont ici comparables à trois variables.

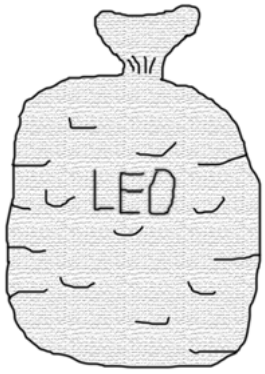
Définition métaphorique

Imaginons le sac de billes d'un petit garçon qui joue dans la cour de son école. Il s'agit d'un petit morceau de toile que lui a cousu sa maman, sur lequel elle a également brodé son prénom.

Nous pourrions dire de ce sac qu'il s'agit d'une variable, qui va représenter le nombre de billes que possède Léo. Le terme variable traduit simplement le fait que le contenu du sac est variable, justement, en fonction du résultat des parties de billes auxquelles Léo participe.

Figure 7-1

Le nom brodé sur le sac de billes d'un petit garçon est comparable au nom d'une variable.



Le sac peut contenir 20, 28, 35 ou 57 billes, peu importe le nombre, mais lorsque le petit garçon y fera référence lors d'une conversation, il évoquera le nombre de billes. En disant « Je te donne mon sac de billes contre tous tes autocollants », il substitue la représentation de ses billes par un terme métaphorique, « mon sac ».

Le petit garçon ne mettra-t-il que des billes dans son sac ? Il pourrait en effet y placer n'importe quel autre jouet, mais s'il ne veut pas se tromper sur l'évaluation du nombre de billes qu'il possède, il a tout intérêt à réserver ce sac au rangement de ses billes.

Cette métaphore du sac de billes, contenant exclusivement des billes, est comparable à la notion de variable : le sac, et de type de variable : des billes mais pas d'objets d'un autre type. Abordons à présent la notion de variable en programmation et notamment en ActionScript.

Déclaration d'une variable

Avant de pouvoir faire référence à une variable dans un script, il est indispensable que vous informiez l'ordinateur de l'existence de celle-ci. Pour cela vous devez, en tout début de script (aussi bien en mode de programmation séquentielle qu'en mode orienté objet), indiquer le nom que vous avez choisi pour représenter la variable. Cette instruction s'appelle une déclaration de variable. Le nom de la variable doit être précédé du mot-clé `var`.

```
var prenom;
var monsac;
var leo;
```

Remarque

Ces trois exemples de déclaration de variables mettent en évidence le choix personnel et libre d'un nom de variable ; nous reviendrons sur cette notion un peu plus loin dans ce chapitre.

Pour l'instant, vous avez déclaré une variable nommée `prenom`, mais il est possible de lui associer n'importe quel type de valeur, ce qui n'est pas une bonne façon de procéder.

Théoriquement et dans la pratique, il est préférable d'associer à une variable ce qu'on appelle un type, c'est-à-dire une information qui précise la nature de son contenu.

```
var prenom:String;
```

Comme vous pouvez le constater, nous avons ajouté deux points derrière le nom de la variable puis un nom, ici `String`. Il s'agit précisément du nom du type, qui impose alors un contenu strict à la variable, que ce soit lors d'une initialisation (action de donner une valeur initiale à une variable) ou lors de toute modification de la variable (changement du contenu de la variable au cours du script). Pour illustrer la notion de type de variable, prenons les deux exemples suivants.

Exemple 1 :

```
var jour:String;
```

Exemple 2 :

```
var jour:Number;
```

Dans le premier exemple, nous savons que nous avons l'obligation d'initialiser la variable avec une chaîne de caractères (comme « lundi » ou « jeudi »), alors que dans le deuxième nous devons lui affecter un nombre (1 pour lundi, 3 pour mardi…).

Il existe de nombreux types de variables. Cependant, nous commencerons par vous présenter les quatre les plus utilisés qui sont : `String` (chaîne de caractères), `Number` (nombre entier), `Array` (tableau) et `Boolean` (valeur booléenne de type vrai/faux). À chaque nouveau besoin de mémorisation d'un type d'information vous découvrirez, ou avez découvert au travers des exemples de ce livre, un type spécifique.

Le choix d'un nom de variable

Le nom d'une variable doit toujours être représentatif de la valeur stockée. Il est préférable d'avoir un nom long et explicite plutôt qu'une abréviation qui n'exprime rien de précis, comme `a` ou `b`... Avant de vous présenter des exemples de noms de variables, voici les principales raisons pour lesquelles les noms des variables doivent être clairs.

- Dans le cas d'un travail en équipe, la mise en commun de scripts est grandement facilitée lorsque les noms des variables traduisent à eux seuls les informations qu'elles représentent.
- Lorsque vous devez effectuer des mises à jour de programmes plusieurs jours après leur création, il est plus facile de se souvenir de la fonctionnalité de chaque variable si elle possède un nom explicite.
- Si vous transmettez un projet sur lequel vous avez travaillé seul à une autre personne chargée d'assurer les mises à jour ou de continuer votre développement, des noms de variables explicites permettent d'éviter la rédaction d'un document expliquant le rôle de chaque variable et des autres fonctions.

Voici à présent des exemples de noms de variables. À la lecture de ces quelques lignes, vous pourriez même deviner le sujet de la production.

```
var titreLegendeVignette:String;
var categorieLegendeVignette: String;
var etudiantsLegendeVignette:Array;
var pitchFilm:String;
var modeLectureVideo:Boolean;
var tableauFichiersFilmsEnContinu:Array;
var tableauNomsFilmsEnContinu:Array;
var numeroFilmALire:Number;
var nbrVignettesMosaic:Number;
var nomImagePremierPlan:String;
var imagePleinEcran:Boolean;
var toucheShiftEnfoncee:Boolean;
var messageAccueilPhotoGraphisme:String;
var legendeVignetteSurvolee:CreationTexte;
var numeroLigneCliquee:Number;
```

Il s'agit de la réalisation d'un *book*, présentant des travaux graphiques, dont nous avons extrait quelques noms de variables. Comme vous pouvez le constater, leurs noms sont longs, mais nous n'aurons pas de mal à comprendre ce qu'elles représentent, contrairement aux variables ci-dessous :

```
var njs:Array;
var sg:Number;
var rar:Number;
```

Il aurait été plus simple de les écrire clairement :

```
var nomsDesJoueursSelectionnes:Array;
var scoreGlobal:Number;
var reductionAvantRemise:Number;
```

Interdiction

Dans le nom d'une variable, vous n'avez pas le droit d'utiliser des caractères accentués ou spéciaux, ni même d'espace. Ne commencez pas non plus le nom d'une variable par un chiffre ou une majuscule car, par convention, une capitale en début de mot indique une classe. Pour finir, tentez de lire le nom de la variable ci-dessous :

```
var numerossectionsretenues:Array;
```

Vous avez sûrement dû faire un effort de lecture pour déchiffrer les mots contenus dans le nom de la variable. Pour éviter cela, une technique consiste à écrire un nom avec une capitale à chaque mot.

```
var numerosSectionsRetenues:Array;
```

Rappel

Comme nous vous l'avons dit, n'utilisez pas de majuscule en début de nom de variable.

Initialiser une variable

Dans la section Déclaration d'une variable de ce chapitre, nous avons évoqué le terme initialisation sans l'avoir défini. Voici de quoi il s'agit.

Une initialisation consiste à donner une valeur initiale à une variable, éventuellement au moment de sa déclaration. Pour cela, vous écrivez le nom de la variable suivi de l'opérateur = puis de la valeur à lui attribuer.

Exemple 1 :

```
var jour:String = "Lundi";
```

Exemple 2 :

```
var jour:Number = 28;
```

> **Remarque**
> L'ajout d'espace avant et/ou après le signe = est facultatif.

Dans les deux exemples ci-dessus, nous initialisons les variables au moment de leur déclaration. Mais dans certains cas, l'initialisation aura lieu bien plus loin dans le script. Ce qui donne :

```
var jour:String;
```

… puis quelques lignes d'instructions plus bas dans le script :

```
jour = "Lundi";
```

Attention, il ne faudra surtout pas réutiliser le mot-clé `var` et encore moins essayer de redéfinir le type, ce qui occasionnerait une erreur. Ainsi, les exemples suivants :

```
var jour:String;
```

… puis quelques lignes d'instructions plus bas dans le script :

```
var jour = "Lundi";
```

…ou encore :

```
jour:String = "Lundi";
```

produisent une erreur.

Pourquoi typer une variable ?

La réponse à cette question est extrêmement simple : si vous ne typez pas les variables d'un script, son débogage sera plus fastidieux et vous vous priveriez ainsi de l'aide que peut vous apporter le compilateur au moment de la publication d'une animation.

En effet, lorsque vous publiez une animation par le raccourci clavier Ctrl-Entrée (Windows) ou Commande-Entrée (Mac), Flash fait appel à son compilateur pour transformer le fichier FLA en SWF. Au moment de cette publication, le compilateur vérifie la validité du code contenu dans le fichier FLA et les différents fichiers AS qui y sont rattachés. Le fait de typer des variables aide le compilateur à déceler les erreurs de type.

Définition

Un compilateur est un programme chargé d'interpréter des lignes d'instructions dans le but de transformer un fichier en un autre lisible par une application tierce. Flash possède un compilateur qui est sollicité au moment de la publication d'une animation.

Voici un premier exemple :

```
var gagnants:Array;
```

…quelques lignes plus loin dans le script…

```
gagnants = 3;
```

Nous avons déclaré une variable de type tableau (`Array`) car nous souhaitons y stocker des prénoms d'individus. Après l'écriture de nombreuses lignes de code, vous finissez parfois par vous interroger sur l'intérêt, le rôle ou le sens de certaines variables que vous avez déclarées au début du programme (d'où l'importance du nom donné au différentes instances et variables). Dans cet exemple, nous tentons d'affecter la valeur 3 à la variable `gagnants`, car nous pensons qu'il pourrait s'agir de mémoriser le nombre de gagnants. Au moment de la publication de l'animation, voilà l'erreur qu'indique Flash !

```
1067: Contrainte implicite d'une valeur du type int vers un type sans rapport Array.
```

Au moment de sa création, nous avons déclaré notre variable `gagnants` de type tableau car nous savions précisément ce qu'elle allait contenir : une liste de prénoms. Au moment de l'initialiser, nous pensons à tort qu'il s'agit d'une variable dans laquelle ont peut stocker un nombre. Le compilateur découvre qu'il y a une erreur, il nous la signale et interrompt l'exécution du programme.

Si nous n'avions pas typé la variable, aucune erreur ne se serait produite. Nous pourrions même changer son contenu accidentellement.

```
var gagnants =  ["David","Marine","Luna"];
gagnants = 3;
```

Ces deux lignes d'instructions sont correctes, elles ne produisent aucune erreur de compilation !

En conclusion, faites l'effort de typer toutes vos variables afin de bénéficier de l'aide que le compilateur de Flash est capable de vous apporter.

Le type *

Dans de rares cas, vous aurez besoin de typer une variable avec deux ou plusieurs types. Cela se produit notamment pour des variables locales dans une boucle `for (in)`. Vous pourrez alors utiliser le caractère `*` pour signaler au compilateur un type multiple.

Vous ne devez surtout pas abuser de cette possibilité qui réduit l'aide que peut vous apporter le compilateur au moment de la publication.

```
var elementsListeArticles:*;
```

Modifier une variable

Comme nous vous l'avons déjà mentionné dans ce chapitre vous pouvez, à n'importe quel moment, modifier la valeur d'une variable. Cependant, faites très attention à ne pas commettre l'une des deux erreurs suivantes :

- Ne vous trompez pas dans la nature de l'information à stocker. Si vous avez déclaré un type `String`, vous devez saisir la valeur entre guillemets ou faire appel à la méthode `toString()` pour effectuer une conversion en chaîne de caractères. Si vous avez déclaré un nombre ou un booléen, vous ne devez surtout pas saisir la nouvelle valeur entre guillemets.
- Ne réutilisez surtout pas le mot-clé `var` qui ne sert qu'au moment de la déclaration.

Voici quelques exemples d'affectation :

- Variable de type `String` : `nomVainqueur="Luna";`
- Variable de type `Number` : `nbrPointsEcran=12;`
- Variable de type `Boolean` : `toucheShiftEnfoncee= true;`
- Variable de type `Boolean` : `toucheShiftEnfoncee= ! toucheShiftEnfoncee;` Cette instruction permet d'inverser la valeur d'une variable booléenne : si la valeur de la variable `toucheShiftEnfoncee` est `true`, elle passera à `false`.
- Variable de type `Number` : `nbrPointEcran= nbrPointEcran + 20;`

Valeur null

Pour effacer le contenu d'une variable, vous pouvez aussi définir sa valeur à `null`.

Portée d'une variable

Avant de poursuivre la lecture des lignes qui vont suivre, référez-vous au chapitre 10 pour apprendre une notion très importante en programmation : la fonction.

Vous pourriez penser qu'à partir du moment où elle est déclarée, une variable peut être modifiée n'importe où dans le programme. Il n'en est rien, tout dépend de l'endroit où se

trouve sa déclaration. Ce qui nous amène à aborder la notion très importante de portée d'une variable.

Prenez garde à la fonction !

Considérez attentivement le script ci-dessous :

Fichier de référence : `Chapitre7/Portee1.fla`

```
var nomAuteur:String = "TARDIVEAU";
btValidation.addEventListener(MouseEvent.MOUSE_DOWN,fenetreSortie);

function fenetreSortie (evt:MouseEvent) {
  trace(nomAuteur);
}
```

Remarque

`btValidation` est le nom d'une occurrence de type Clip sur la scène.

La variable `nomAuteur` a été déclarée dans le corps principal du script en-dehors de toute fonction. Elle est visible dans tout le script, à tous les niveaux, sur toutes les lignes d'instructions. Nous pouvons donc nous y référer dans la fonction `fenetreSortie()`.

Dans ce deuxième exemple, nous allons déclarer une variable dans la fonction `fenetreSortie()` et tenter d'y faire référence en dehors de celle-ci.

```
btValidation.addEventListener(MouseEvent.MOUSE_DOWN,fenetreSortie);

function fenetreSortie(evt:MouseEvent) {
  var nomAuteur:String = "TARDIVEAU";
  trace(nomAuteur);
}

trace(nomAuteur);
```

Au moment de la publication de l'animation, le compilateur de Flash nous renvoie un message d'erreur.

```
1120: Accès à la propriété non définie nomAuteur.
```

Comme la variable `nomAuteur` est déclarée entre les accolades `{ }` délimitant la fonction `fenetreSortie()`, elle n'est visible que dans cette partie du script. Elle n'est donc pas reconnue en dehors de la fonction et toute tentative d'y accéder génère une erreur.

Ce qu'on appelle la portée d'une variable est la partie du code dans laquelle cette variable est visible, donc accessible. La variable `nomAuteur` du premier exemple a une portée globale, alors que celle du deuxième exemple a une portée réduite à la fonction `fenetreSortie()`.

En conclusion, si vous souhaitez qu'une variable puisse être accessible depuis n'importe quel endroit du script, vous devez la déclarer en début de programme, à l'extérieur d'une fonction.

Les variables en programmation orientée objet

Si vous n'avez pas encore pris connaissance du chapitre 1, lisez les notions qui y sont abordées avant de poursuivre votre lecture.

La portée d'une variable s'applique de la même façon en programmation orientée objet.

Fichiers de référence : `Chapitre7/Portee2.fla` et `Main.as`

```
package {

  import flash.display.Sprite;

  public class Main extends Sprite {

    public var nomAuteur:String;

    function Main() {
      nomAuteur = "TARDIVEAU";
      trace("Depuis la fonction constructrice 'Main' :  "+nomAuteur);
      afficherNomAuteur();
    }
    private function afficherNomAuteur() {
      trace("Depuis la fonction 'afficherNomAuteur' :  "+nomAuteur);
    }
  }
}
```

Classe du document

Rappelons que le fichier `Portee2.fla` possède le nom `Main` dans la palette Propriétés au niveau du champ de saisie Classe du document.

La variable `nomAuteur` a été déclarée en dehors du constructeur de la classe et de la méthode `afficherNomAuteur()`. De ce fait, elle peut être utilisée à n'importe quel endroit dans le code. En revanche, si nous l'avions déclarée et initialisée à l'intérieur d'une fonction, elle n'aurait pas été accessible à un autre endroit du script.

```
package {

  import flash.display.Sprite;

  public class Main extends Sprite {

    function Main() {
      trace("Depuis la fonction constructrice 'Main' :  "+nomAuteur);
      reglageNomAuteur()
      afficherNomAuteur();
    }
```

```
    private function reglageNomAuteur() {
      var nomAuteur:String = "TARDIVEAU";
      trace("Depuis la fonction 'reglageNomAuteur' :  "+nomAuteur);
    }

    private function afficherNomAuteur() {
      trace("Depuis la fonction 'afficherNomAuteur' :  "+nomAuteur);
    }
  }
}
1120: Accès à la propriété non définie nomAuteur.
1120: Accès à la propriété non définie nomAuteur.
```

Le code ci-dessus génère une erreur car les fonctions `Main()` (constructeur) et `afficherNomAuteur()` tentent d'accéder à une variable dont la portée est limitée. En effet, la variable `nomAuteur` a été déclarée et initialisée dans la fonction `reglageNomAuteur()`, et sa portée se limite donc au bloc de code contenu entre les accolades `private function reglageNomAuteur() { }`.

> **Rappel**
> Un bloc de code commence par une accolade ouvrante et se termine par une accolade fermante.

public, private, static

Lorsque vous déclarez des variables dans une classe, vous devez non seulement les préfixer avec le mot-clé `var`, mais vous pouvez, et même dans certains cas devez, ajouter un spécificateur de contrôle d'accès (`public`, `private` ou `static`).

C'est en particulier le cas lorsqu'on travaille avec des classes et des scripts répartis dans plusieurs fichiers. Il faut alors gérer la portée d'une variable d'un fichier par rapport à un autre.

Les variables static

Considérons l'exemple suivant, où nous définissons une classe de document nommée `MainCreaTexte()` dans laquelle nous déclarons une variable intitulée `nomAuteur`. Nous aimerions que cette variable soit accessible à partir d'une autre classe intitulée `CreaTexte()`.

Afin qu'une variable soit accessible depuis une autre classe, il faut habituellement instancier la classe dans laquelle elle se trouve pour y accéder à partir de l'instance. Dans notre cas, cela n'est pas possible et pourtant nous souhaitons tout de même atteindre la variable `nomAuteur`. Nous allons tout simplement transformer notre variable en variable `static` afin d'y avoir accès de n'importe où dès lors qu'on précise en préfixe le nom de la classe.

Fichiers de référence : `Portee3.fla`, `MainCreaTexte.as` et `CreaTexte.as`.

Voici le script de la classe du document `Portee3.fla`. Vous pouvez constater que nous avons ajouté le spécificateur de contrôle d'accès `static` devant le mot-clé `var`.

```
package {

  import flash.display.Sprite;

  public class MainCreaTexte extends Sprite {

    public static  var nomAuteur:String;

    function MainCreaTexte() {
      new CreaTexte();
    }
  }
}
```

Cela autorise l'accès à la variable `nomAuteur` à partir de la classe `CreaTexte()` dans la mesure où elle est préfixée par le nom de la classe : `MainCreaTexte.nomAuteur`.

```
package {

  import flash.display.Sprite;

  public class CreaTexte extends Sprite {

    function CreaTexte() {
      MainCreaTexte.nomAuteur = "TARDIVEAU";
      trace(MainCreaTexte.nomAuteur);
    }

  }

}
```

Un telle variable, qui appartient de ce fait à la classe et non à une instance (ou occurrence) de classe, permet de conserver une information accessible depuis n'importe quel endroit à n'importe quel moment. Cela s'avère fort utile pour accéder à des variables qui représentent des informations telles que des scores de jeu, un nom d'utilisateur, etc.

Les variables public et private

Comme nous venons de l'expliquer, une variable `static`, est accessible depuis n'importe quelle autre classe, mais elle appartient, en revanche, à la classe dans laquelle elle a été déclarée. Par opposition à cette notion, découvrons à présent comment associer une variable à une instance de classe.

Fichiers de référence : `Portee4.fla`, `MainJeu.as` et `Bateau.as`.

Dans cet exemple, nous définissons une classe de document intitulée `MainJeu.as` dans laquelle nous déclarons une variable.

```
public  var bateau1:Bateau;
```

Nous instancions ensuite la classe `Bateau()`.

```
bateau1 = new Bateau();
```

Puis nous définissons des variables d'instance.

```
bateau1.nom="L'ami râle";
bateau1.longueur=130;
```

Si nous pouvons initialiser ainsi ces variables, en précisant le nom d'instance, c'est parce qu'elles ont été déclarées comme `public` dans la classe `Bateau`. Sans cela, il serait impossible de faire référence aux variables `nom` et `longueur`.

Voici les deux classes `MainJeu` et `Bateau` pour vous rendre compte de leurs structures respectives.

```
package {

  import flash.display.Sprite;

  public class MainJeu extends Sprite {

    public  var bateau1:Bateau;

    function MainJeu() {
      bateau1 = new Bateau();

      bateau1.nom="L'ami râle";
      bateau1.longueur=130;

      trace(bateau1.nom);
      trace(bateau1.longueur);

    }
  }
}
```

et

```
package {

  import flash.display.Sprite;

  public class Bateau extends Sprite {

    public var nom:String;
    public var longueur:Number;

  }
}
```

Si nous remplacions le spécificateur `public` par `private`, il deviendrait alors impossible de faire référence aux variables `nom` et `longueur`, car elles ne seraient accessibles qu'à partir de la classe.

Les constantes

Une information est parfois stockée dans une variable sans qu'il soit nécessaire de pouvoir la modifier dans le script. C'est ce qu'on appellera une constante. Elle se déclare avec le mot-clé `const`, comme dans l'exemple suivant :

```
const var TVA55:Number = 0.055;
const var remise10:Number = 0.10;
```

8

Les tableaux

Dans le chapitre précédent, nous avons découvert que les variables nous permettent de mémoriser une information afin de pouvoir y faire référence en utilisant une étiquette au lieu de la valeur elle-même.

Dans certains cas, vous aurez besoin de mémoriser plusieurs valeurs dans une même variable, c'est le rôle des tableaux, aussi appelés listes ou *array.*

Nous allons voir que cette technique est extrêmement simple d'emploi : un tableau est généralement déclaré avec le mot-clé `var` et le type `Array`. On utilise également des spécificateurs de contrôle d'accès, si nécessaire. Enfin, l'accès aux éléments d'un tableau se fait par des index.

Créer un tableau

Vous disposez de deux techniques pour créer un tableau. Vous pouvez utiliser la classe `Array` de la façon suivante :

```
var semaine:Array = new Array("Lundi","Mardi","Mercredi");
```

ou employer des crochets comme dans l'exemple ci-dessous :

```
var semaine:Array = ["Lundi","Mardi","Mercredi"];
```

Chaque élément du tableau, qualifié d'entrée, est séparé des autres par une virgule. Dans cet exemple, nous avons stocké des chaînes de caractères, c'est pourquoi nous les avons placées entre guillemets, mais nous aurions pu mémoriser des éléments d'un autre type :

- Des nombres, qui doivent alors être saisis sans guillemets en utilisant le point comme séparateur, s'il s'agit d'un nombre décimal comme `8.3`.

- Des variables.
- Des booléens.
- Des noms d'instances.
- Des tableaux.

En fonction de vos besoins, vous serez en outre amenés à écrire un jour ou l'autre un programme dont les tableaux pourront contenir d'autres types de données, comme ceux que nous allons examiner ci-dessous.

> **Remarque**
>
> L'aide officielle de l'ActionScript proposée par Adobe utilise le terme élément pour désigner l'entrée d'un tableau, ce qui est l'appellation la plus courante.

Tableau de propriétés ou tableau associatif

Il serait pratique de pouvoir écrire dans un programme les lignes d'instructions ci-dessous pour afficher, par exemple, les noms des capitales de la France et de l'Angleterre :

```
trace(capitales.France);
trace(capitales.Angleterre);
```

C'est tout à fait possible, mais pour que les noms de `Paris` et `Londres` puissent être affichés, vous allez devoir créer un tableau associatif, appelé aussi tableau de propriétés. D'un point de vue purement programmatique, nous n'allons pas créer un tableau, mais un objet qui va contenir des propriétés, au même titre qu'une occurrence possède des propriétés (comme `x`, `y`, `alpha`, `rotation`, etc.).

Commencez par déclarer une variable en spécifiant le type `Object`.

```
var europe:Object
```

Assignez ensuite à cette variable une paire d'accolades pour indiquer une liste de valeurs.

```
var europe:Object = {}
```

Définissez entre ces accolades une série d'entrées qui se présentent sous la forme :

```
nomDeLaPropriete:valeur
```

Toutes les entrées doivent être séparées par des virgules, comme nous l'avons vu avec les tableaux au début de ce chapitre.

Attention : le nom qui figure devant les deux points ne doit pas être placé entre guillemets car il s'agit d'une propriété. De même, si la valeur est un nombre, vous ne devez pas la saisir entre guillemets contrairement à une chaîne de caractères. Voici deux exemples pour illustrer nos propos.

```
var capitales:Object = {France:"Paris",Angleterre:"Londres",Espagne:"Madrid"};
var populations:Object = {France:62354654,Angleterre:64234845,Espagne:47658475};
```

Un tableau à deux dimensions

Paradoxalement, nous avons utilisé jusqu'à présent le terme tableau pour faire référence à une liste de données, bien que cela ne soit pas le sens commun, puisqu'un tableau est généralement constitué au minimum de 2 colonnes et de 2 lignes (en programmation, un tableau peut tout de même ne posséder qu'une seule colonne ou une seule ligne, ce qui revient en fait à une simple liste d'éléments).

Nous avons également indiqué qu'un tableau peut contenir des éléments de type tableau. Nous allons expliciter cette notion.

En effet, vous pouvez créer ce qu'on appelle communément un tableau à deux dimensions. Pour cela, vous devez créer plusieurs tableaux, avec des entrées de type quelconque, puis, créer un tableau supplémentaire qui regroupe tous les tableaux précédents.

Imaginez un grand magasin établi sur quatre niveaux.

```
var etage0:Array=["Jouets pour enfants","Habits pour enfants","Lingerie féminine"];
var etage1:Array=["Parfumerie","Agence de voyage","Accueil"];
var etage2:Array=["Alimentation exotique","Café","Vaisselle"];
var etage3:Array=["Linge de maison","Vêtements Homme"];

var lePrendTout:Array=[etage0,etage1,etage2,etage3];
```

Nous stockons dans des tableaux simples des valeurs de type chaîne de caractères, puis générons un tableau à deux dimensions qui contient l'ensemble des données. Chaque élément du tableau `lePrendTout` est donc un tableau. Consultez la section Lire une entrée de tableau pour comprendre comment accéder à une entrée et ainsi lire sa valeur ; vous comprendrez alors pourquoi nous parlons de deux dimensions.

L'avantage de créer de tels tableaux est de pouvoir manipuler de nombreuses données à partir d'une seule étiquette, c'est-à-dire le nom d'un tableau de synthèse.

Un tableau à deux dimensions contenant des tableaux associatifs

Vous pouvez combiner des tableaux associatifs ou de propriétés dans un tableau normal. Cela vous donnera davantage de souplesse dans la gestion de données plus complexes. Examinez l'exemple ci-dessous pour mesurer la portée de notre propos.

```
var europe:Object = {France:"Paris",Angleterre:"Londres",Espagne:"Madrid"};
var afrique:Object = {Niger:"Niamey",Congo:"Brazzaville",Gabon:"Libreville"};
var monde:Array = new Array(europe,afrique);
```

Nous avons créé deux tableaux associatifs qui possèdent des propriétés correspondant à des noms de pays et nous avons défini leurs valeurs en faisant référence aux noms de leur capitale.

Pour obtenir l'ensemble que nous avons appelé `monde`, nous déclarons et initialisons un tableau à deux dimensions. Nous aurions également pu imbriquer ces tableaux associatifs dans un autre.

```
var europe:Object = {France:"Paris",Angleterre:"Londres",Espagne:"Madrid"};
var afrique:Object = {Niger:"Niamey",Congo:"Brazzaville",Gabon:"Libreville"};
var monde:Object = {europe:europe,afrique:afrique};

trace(monde.europe.France);
```

Créer un tableau vide

Au cours de la lecture d'une animation, vous aurez très souvent besoin de mémoriser des informations que vous ne connaissez pas à l'avance ; par exemple le panier d'un site marchand, dans lequel vous pouvez ajouter et supprimer des articles. Nous allons donc apprendre à ajouter et supprimer des entrées dans un tableau. Pour cela, il faut préalablement créer un tableau vide. La technique est simple et vous la connaissez déjà !

```
var scoresJoueurs:Array = [];
```

ou

```
var scoresJoueurs:Array = new Array();
```

Il suffit en effet de créer un tableau dans lequel vous ne spécifiez aucune entrée, contrairement à ce que nous avons fait jusqu'à présent.

Si vous souhaitez créer un tableau qui contient des entrées vides, utilisez les lignes d'instructions ci-dessous :

```
var villes:Array;
villes = new Array();
villes[9]="";
```

De cette façon, vous créez dix entrées vides portant les index 0 à 9. La notion d'index est explicitée dans la section suivante.

Lire une entrée de tableau

Pour constituer nos tableaux, nous avons utilisé la classe `Array`, mais également une paire de crochets `[]`. Ce sont ces deux caractères que nous allons utiliser pour lire une entrée de tableau. Avant de donner un exemple, nous devons préciser que chaque élément d'un tableau possède un index, c'est-à-dire un numéro, que nous allons utiliser pour accéder à l'élément. Par convention, la première entrée d'un tableau possède toujours l'index 0. Ainsi, pour afficher le mot `Lundi`, premier élément du tableau ci-dessous, nous devons mentionner le chiffre 0.

```
var semaine:Array = new Array("Lundi","Mardi","Mercredi");
trace(semaine[0]);
```

Que vous utilisiez la classe `Array` ou les crochets pour créer un tableau, la lecture sera toujours conduite de la même façon.

Pour lire la dernière entrée d'un tableau, utilisez la ligne d'instruction suivante :

```
trace(semaine[semaine.length-1]);
```

L'instruction `semaine.length` donne la taille du tableau, donc son nombre d'éléments, valeur à laquelle il suffit d'enlever 1 pour obtenir l'index du dernier élément. Pour parcourir toutes les entrées d'un tableau, référez-vous au chapitre 10 qui traite des boucles `for()`.

Lire l'entrée d'un tableau à deux dimensions

Nous allons utiliser le même principe pour lire les éléments d'un tableau à deux dimensions, mais en ajoutant un deuxième index.

```
var etage0:Array=["Jouets pour enfants","Habits pour enfants","Lingerie féminine"];
var etage1:Array=["Parfumerie","Agence de voyage","Accueil"];
var etage2:Array=["Alimentation exotique","Café","Vaisselle"];
var etage3:Array=["Linge de maison","Vêtements Homme"];

var lePrendTout:Array=[etage0,etage1,etage2,etage3];

trace(lePrendTout[1][2]);
```

Dans cet exemple, la première paire de crochets fait référence à l'une des entrées du tableau `lePrendTout`, et plus précisément à son deuxième élément, le tableau `etage1`. À l'intérieur de ce dernier, la deuxième paire de crochets fait référence à l'entrée portant l'index 2, c'est-à-dire `Accueil`.

Lire l'entrée d'un tableau associatif

La technique de lecture d'une entrée d'un tableau associatif est extrêmement simple, car elle est comparable à la lecture des propriétés d'une occurrence. Vous devez faire référence au tableau, qui est avant tout un objet, puis indiquer ensuite le nom d'une des propriétés séparée par un point.

```
var europe:Object = {France:"Paris",Angleterre:"Londres",Espagne:"Madrid"};
var afrique:Object = {Niger:"Niamey",Congo:"Brazzaville",Gabon:"Libreville"};

trace(europe.France);
```

Lire l'entrée d'un tableau à deux dimensions contenant un tableau associatif

Pour lire l'entrée d'un tableau à deux dimensions contenant un tableau associatif, vous devez combiner les deux techniques de lecture précédentes de la façon suivante :

```
var europe:Object = {France:"Paris",Angleterre:"Londres",Espagne:"Madrid"};
var afrique:Object = {Niger:"Niamey",Congo:"Brazzaville",Gabon:"Libreville"};
var monde:Array = new Array(europe,afrique);

trace(monde[0].France);
```

Ce code fait tout d'abord référence à l'une des entrées du tableau à l'aide de son index, puis précise le nom de la propriété à utiliser.

Exemple 1

Fichier de référence : `Chapitre8/Tableau2.fla`

Dans cet exemple, sans l'utilisation d'un tableau, il faudrait effectuer de nombreuses comparaisons pour afficher le jour et le mois sur la scène.

```
var numeroJour:Number=0;

var semaine:Array = new Array("Lundi","Mardi","Mercredi","Jeudi","Vendredi",
➥"Samedi","Dimanche");

btSuivant.addEventListener(MouseEvent.MOUSE_DOWN,jourSuivant);
btPrecedent.addEventListener(MouseEvent.MOUSE_DOWN,jourPrecedent);

function jourSuivant (evt:MouseEvent) {
  numeroJour++;
  if(numeroJour>6) numeroJour=6;
  nomJourSemaine.text = semaine[numeroJour];
}

function jourPrecedent (evt:MouseEvent) {
  numeroJour--;
  if(numeroJour<0) numeroJour=0;
  nomJourSemaine.text = semaine[numeroJour];
}

nomJourSemaine.text = "Lundi";
```

Exemple 2

Fichier de référence : `Chapitre8/Tableau3.fla`

Dans ce deuxième exemple, l'utilisation d'un tableau s'avère être la meilleure solution pour obtenir les noms des jours et des mois. Précisons que la classe `Date()` va d'ailleurs nous renvoyer des valeurs comprises entre 0 et 6 pour les jours de la semaine et de 0 à 11 pour les mois. Nous allons les utiliser pour faire référence aux index des deux tableaux.

```
var jour:String;
var dateJour:String;
var mois:String;
var annee:Number;

var liste_semaine:Array = new Array("Lundi","Mardi","Mercredi","Jeudi","Vendredi",
➥"Samedi","Dimanche");
var liste_mois:Array = new Array("Jan","Fev","Mar","Avr","Mai","Juin","Juil","Aou",
➥"Sep","Oct","Nov","Dec");

var instant:Date = new Date();

jour = liste_semaine[instant.getDay()];
```

```
dateJour=instant.getDate().toString();
mois=liste_mois[instant.getMonth()];
annee=instant.getFullYear();

affichageDate.text = jour+" "+dateJour+" "+mois+" "+annee.toString();
```

Modifier une entrée de tableau

Nous avons appris à créer et lire un tableau ; nous devons à présent apprendre à modifier une entrée. Une fois encore la technique est extrêmement simple, car il suffit de faire référence à l'élément à modifier et de lui affecter une nouvelle valeur.

Supposons que vous ayez déclaré et initialisé le tableau suivant, en commettant une erreur sur le mot Nancy.

```
var villes:Array = ["Paris","Toulon","Brest","Nacy"];
```

Nous pouvons modifier cet élément, qui porte l'index 3, de la façon suivante :

```
villes[3] = "Nancy";
```

Remarque

Si vous essayez de lire une entrée qui n'existe pas, le lecteur Flash vous renvoie la valeur `undefined`. Si vous tentez de changer la valeur d'un élément qui n'existe pas, des entrées vides seront ajoutées dans le tableau jusqu'à la position de l'élément que vous souhaitiez modifier ; puis un dernier élément sera créé et initialisé à la valeur indiquée.

Pour vider la valeur d'une entrée, assignez-lui tout simplement la valeur `null`.

Nous allons à présent vous exposer deux exemples relativement complexes, mais qui vous donneront un aperçu de la notion de modification d'une entrée de tableau.

Exemple 1

Fichier de référence : `Chapitre8/Tableau4.fla`

Dans ce premier exemple, nous avons disposé quatre zones de saisie sur la scène. Lorsque nous modifions l'une d'entre elles, la valeur saisie est enregistrée dans le tableau `etatCases`.

```
var etatCases:Array = new Array("Prénom","Prénom","Prénom","Prénom");

nomJoueur0.addEventListener(Event.CHANGE,miseAJourTableau);
nomJoueur1.addEventListener(Event.CHANGE,miseAJourTableau);
nomJoueur2.addEventListener(Event.CHANGE,miseAJourTableau);
nomJoueur3.addEventListener(Event.CHANGE,miseAJourTableau);

function miseAJourTableau(evt:Event) {
  etatCases[evt.currentTarget.name.substr(9,1)]=evt.currentTarget.text;
  listeComplete.text = etatCases.toString();
}
```

Comme vous pouvez le constater, la fonction de rappel est la même pour toutes les zones de saisie. La ligne d'instruction ci-dessous permet de déterminer l'instance qui est en cours de changement, ce qui nous permet d'obtenir un numéro d'index pour modifier une entrée précise du tableau `etatCase`.

```
etatCases[evt.currentTarget.name.substr(9,1)]=evt.currentTarget.text;
```

L'objet `evt`, accompagné de sa propriété `currentTarget`, renvoie l'instance en cours de changement. La propriété `name` précise son nom et la méthode `substr()` permet de récupérer le caractère portant l'index 9 (le dixième caractère du mot). Il s'agit du numéro placé à la fin de chaque nom d'instance, comme 2 pour le dixième caractère de l'instance `nomJoueur2`. Il ne nous reste plus qu'à l'utiliser comme index du tableau `etatCases`.

Remarque

Nous utilisons la méthode `toString()` pour convertir le tableau en chaîne de caractères.

Exemple 2

Fichier de référence : `Chapitre8/Tableau5.fla`

Ce deuxième exemple est plus simple, car nous nous contentons de reprendre le même principe que l'exemple 1 en l'adaptant à un nouveau besoin. Un clic sur une occurrence permet de basculer la valeur booléenne d'une entrée du tableau `etatCases`. Nous l'utilisons alors pour régler l'opacité de l'instance cliquée.

```
var etatCases:Array = new Array(true,true,true,true);

case0.addEventListener(MouseEvent.MOUSE_DOWN,changerValeurEntree);
case1.addEventListener(MouseEvent.MOUSE_DOWN,changerValeurEntree);
case2.addEventListener(MouseEvent.MOUSE_DOWN,changerValeurEntree);
case3.addEventListener(MouseEvent.MOUSE_DOWN,changerValeurEntree);

function changerValeurEntree(evt:MouseEvent) {
  var numeroCase:Number = Number(evt.currentTarget.name.substr(4,1));
  etatCases[numeroCase] = ! etatCases[numeroCase];
  evt.currentTarget.alpha = Number(etatCases[numeroCase] )+0.5;
}
```

Lorsque les valeurs booléennes `true` et `false` sont converties en nombres par l'intermédiaire de la classe `Number()`, nous obtenons respectivement les chiffres 1 et 0.

Ajouter une entrée

Pour ajouter une entrée à un tableau, vous disposez de plusieurs techniques. Tout dépend de l'endroit dans le tableau où l'ajout a lieu : à la fin, au début, au milieu ou à une position encore inexistante, en spécifiant un index supérieur au nombre d'éléments du tableau.

Commençons par la dernière technique que vous rencontrerez souvent dans les livres et/ou autres publications numériques.

Par index

Considérons la déclaration suivante :

```
var jours:Array = new Array();
```

Nous allons simplement définir la valeur des trois premières entrées du tableau en faisant référence aux index 0, 1 et 2, éléments qui n'existent pas encore.

```
jours[0]="Lundi";
jours[1]="Mardi";
jours[2]="Mercredi";
```

L'affichage des éléments du tableau est sans surprise :

```
trace(jours);
Lundi,Mardi,Mercredi
```

Cette technique est simple, mais elle ne s'applique pas à tous les cas.

À la fin d'un tableau

Imaginons, à présent, que nous disposions déjà d'un tableau et que nous souhaitions tout simplement ajouter une entrée à la fin. Pour cela, vous allez utiliser la méthode `push()` de la classe `Array()`.

```
var jours:Array = new Array("Lundi","Mardi","Mercredi");
jours.push("Jeudi");
```

La valeur de l'entrée à insérer est un paramètre de la méthode `push()`et doit donc être placée à l'intérieur des parenthèses. Si plusieurs éléments doivent être ajoutées, séparez-les simplement par des virgules.

```
jours.push("Vendredi","Samedi","Dimanche");
```

C'est la méthode que vous utiliserez le plus fréquemment.

Au début d'un tableau

Dans certains cas, vous devrez ajouter un élément au début du tableau et non à la fin. Vous utiliserez alors la méthode `unShift()` de la classe `Array()`.

```
var jours:Array = new Array("Mardi","Mercredi");
jours.unshift("Lundi");
```

De façon analogue à la méthode `push()`, vos pouvez insérer plusieurs entrées si vous spécifiez, en paramètres de la méthode `unShift()`, plusieurs valeurs séparées par des virgules.

Au milieu d'un tableau

Ce type d'ajout est plus rare, mais il est indispensable de le connaître. Cela vous évitera des manipulations complexes avec plusieurs tableaux, le jour où vous en aurez besoin. Vous allez utiliser la méthode `splice()` avec 3 paramètres.

```
var jours:Array = new Array("Lundi","Mardi","Jeudi");
jours.splice(2,0,"Mercredi");
```

Le premier paramètre précise l'index à partir duquel l'insertion de la nouvelle entrée doit avoir lieu. Le deuxième paramètre indique un nombre éventuel d'entrées à supprimer avant d'effectuer l'insertion ; il est généralement réglé à `0`, comme dans notre exemple. Enfin, le troisième paramètre correspond à la valeur de la nouvelle entrée à insérer.

Vous utiliserez également cette méthode pour supprimer une ou plusieurs entrées dans un tableau, mais sans définir le troisième paramètre et en donnant au deuxième le nombre d'éléments à supprimer, nombre forcément différent de 0.

Voici deux exemples très intéressants qui démontrent l'utilisation de la méthode `push()` à travers deux cas d'école.

Exemple 1

Fichier de référence : `Chapitre8/Tableau6.fla`

À chaque clic sur la scène, la coordonnée horizontale de la position de la souris est mémorisée dans un tableau. Pour les besoins de la démonstration, nous affichons le contenu du tableau sur la scène, ce que nous ne ferions probablement pas dans un cas concret.

```
var listeCoordonnees:Array = new Array();

stage.addEventListener(MouseEvent.MOUSE_DOWN,ajouterPoint);

function ajouterPoint(evt:MouseEvent) {
  listeCoordonnees.push(mouseX);
  affichageCoordonnees.text = listeCoordonnees.toString();
}
```

Exemple 2

Fichier de référence : `Chapitre8/Tableau7.fla`

Un inventaire, autre cas d'école… Vous devez déplacer des occurrences dans une zone précise de l'interface pour les mémoriser dans le but de les réutiliser par la suite. Ce script n'est pas optimisé car, nous n'avons volontairement pas utilisé de boucle `for()`, cependant il permet d'illustrer le mécanisme de gestion d'un glisser-déposer qui conduit à modifier les éléments d'un tableau.

```
var inventaire:Array = new Array();

article0.nom="Poires";
```

```
article1.nom="Pommes";
article2.nom="Raisin";
article3.nom="Cerises";

article0.numeroListe=-1;
article1.numeroListe=-1;
article2.numeroListe=-1;
article3.numeroListe=-1;

article0.addEventListener(MouseEvent.MOUSE_DOWN,deplacer);
article1.addEventListener(MouseEvent.MOUSE_DOWN,deplacer);
article2.addEventListener(MouseEvent.MOUSE_DOWN,deplacer);
article3.addEventListener(MouseEvent.MOUSE_DOWN,deplacer);

article0.addEventListener(MouseEvent.MOUSE_UP,relacher);
article1.addEventListener(MouseEvent.MOUSE_UP,relacher);
article2.addEventListener(MouseEvent.MOUSE_UP,relacher);
article3.addEventListener(MouseEvent.MOUSE_UP,relacher);

function deplacer(evt:MouseEvent) {
  evt.currentTarget.startDrag(false);
}

function relacher(evt:MouseEvent) {
  stopDrag();
  if (evt.currentTarget.x>350 && evt.currentTarget.numeroListe==-1) {
    inventaire.push(evt.currentTarget.nom);
    evt.currentTarget.numeroListe = inventaire.length-1;
  }
  if (evt.currentTarget.numeroListe>=0 && evt.currentTarget.x<350) {
    inventaire.splice(evt.currentTarget.numeroListe,1);
    evt.currentTarget.numeroListe=-1;
  }
  listeArticles.text = inventaire.toString();
}
```

Supprimer une entrée

À la fin d'un tableau

De la même façon que vous utilisez la méthode `push()`pour ajouter une entrée à la fin d'un tableau, vous pouvez la retirer avec la méthode `pop()`.

Son emploi est très simple, il suffit de l'appeler à partir du nom du tableau :

```
var jours:Array = new Array("Lundi","Mardi","Mercredi");
jours.pop();
```

Après l'exécution de ces deux lignes d'instructions, le tableau `jours` ne contiendra plus que deux éléments : `Lundi` et `Mardi`.

Au début d'un tableau

Pour ajouter une entrée au début d'un tableau, nous avons utilisé la méthode `unShift()` ; pour la supprimer, nous allons utiliser la méthode `shift()`.

```
var jours:Array = new Array("Lundi","Mardi","Mercredi");
jours.shift();
```

Après l'exécution de ces deux lignes d'instructions, le tableau `jours` ne contiendra plus que deux entrées : `Mardi` et `Mercredi`.

Au milieu d'un tableau

Comme nous l'évoquions à propos de l'ajout d'une entrée au milieu d'un tableau, nous allons utiliser la méthode `splice()` avec seulement deux paramètres.

```
var jours:Array = new Array("Lundi","Mardi","Mercredi","Jeudi","Thursday","Vendredi");
jours.splice(4,1)
```

Le premier paramètre précise l'index à partir duquel la suppression doit avoir lieu. Le deuxième paramètre indique le nombre d'éléments à supprimer. Généralement, les éléments sont supprimés un par un, ce dernier paramètre est donc la plupart du temps fixé à 1.

Après l'exécution des deux lignes d'instructions ci-dessus, le tableau `jours` contiendra les mêmes éléments qu'au départ à l'exception de celui d'index 4, c'est-à-dire le cinquième du tableau, `Thursday`.

Trier les entrées d'un tableau

Lorsque vous créez ou ajoutez des entrées dans un tableau, vous ne le faites pas obligatoirement dans l'ordre alphabétique ou croissant. C'est pourquoi il est possible de trier les entrées d'un tableau à n'importe quel moment. Vous utiliserez pour cela les méthodes `sort()` (pour les tableaux avec index) et `sortOn()` (pour les tableaux associatifs) de la classe `Array()`.

Considérons l'exemple suivant :

```
var jours:Array = new Array("Eva","Béatrice","Julien","David","Marine","Luna");
```

Comme vous pouvez le constater, les noms ne sont pas triés. La ligne d'instruction ci-dessous permet d'effectuer un tri par ordre croissant :

```
jours.sort()
```

L'instruction suivante trie le tableau par ordre décroissant, comme l'indique la constante `DESCENDING` :

```
jours.sort(Array.DESCENDING);
```

La classe `Array()` propose également d'autres options de tri :

- `Array.CASEINSENSITIVE`
- `Array.UNIQUESORT`
- `Array.RETURNINDEXEDARRAY`
- `Array.NUMERIC`

Enfin, si vous souhaitez trier les éléments d'un tableau dans l'ordre inverse, utilisez la méthode `reverse()`.

Filtrer un tableau

Avec la méthode `filter()` de la classe `Array()`, vous allez découvrir qu'il est possible de générer un nouveau tableau sans être obligé de le parcourir à l'aide d'une boucle `for()` accompagnée d'une structure conditionnelle.

Prenons l'exemple d'une liste de prénoms avec un préfixe propre à chaque pays (`fr`, `it`, `es` et `gb`). Nous aimerions générer un nouveau tableau contenant seulement les noms français (ceux ayant pour préfixe `fr`), ce qui revient à filtrer le tableau.

```
var participants:Array = new Array("frFrançois","itLuna","frMarine","esJuan",
➥"gbDoglas","itGina","frMarc");
```

Nous commençons par créer un nouveau tableau qui contiendra les entrées filtrées.

```
var nomsFrancais:Array;
```

Nous le remplissons avec les éléments obtenus par la méthode `filter()`, qui contient le nom de la fonction de rappel qui effectue le traitement approprié.

```
nomsFrancais=participants.filter(slectionFrancais);
```

Pour qu'un élément soit filtré et donc sélectionné, la fonction de rappel doit renvoyer la valeur `true` pour l'élément concerné.

```
function slectionFrancais(nomEntree:*, index:int, arr:Array):Boolean {
  return nomEntree.substr(0,2)=="fr";
}
```

Dans cet exemple, nous cherchons à savoir si les deux premières lettres d'une entrée sont `fr`. Voici le script global.

```
var nomsFrancais:Array;

var participants:Array = new Array("frFrançois","itLuna","frMarine","esJuan",
➥"gbDoglas","itGina","frMarc");
nomsFrancais=participants.filter(slectionFrancais);
trace(nomsFrancais);

function slectionFrancais(nomEntree:*, index:int, arr:Array) {
  return nomEntree.substr(0,2)=="fr";
}
```

Dans cet autre exemple, un nouveau tableau intitulé `meilleursScores` est généré à la suite d'un filtre qui recherche des valeurs supérieures ou égales à 100.

```
var meilleursScores:Array;

var scores:Array = new Array(125,78,65,98,114,99,139);
meilleursScores=scores.filter(scoreSupA100);
trace(meilleursScores);

function scoreSupA100(lescore:*, index:int, arr:Array) {
  return lescore >= 100;
}
```

9

Les structures conditionnelles

L'interactivité d'une animation Flash est très souvent gérée par l'évaluation de certaines conditions. C'est pourquoi la plupart des scripts que vous rédigerez contiendront des structures conditionnelles.

Structure conditionnelle if()

Préambule

Avant de réaliser ensemble nos premiers tests, commençons par un petit rappel : il existe différents types de variables, tels que `String`, `Number`, `Array`, etc., mais il en est un qui va particulièrement nous être utile pour effectuer un test : le type booléen (`Boolean`). Ce qui caractérise une variable de ce type est qu'elle ne peut prendre que les deux valeurs `true` (pour vrai) ou `false` (pour faux).

Pour effectuer un test dans un script, il existe de nombreuses syntaxes. Nous les aborderons d'ailleurs toutes au cours de ce chapitre, mais commençons par la plus simple :

```
if(scoreJoueur>200) bonus = 10;
```

Il est également possible d'écrire cette structure sur plusieurs lignes, les deux écritures étant équivalentes :

```
if(scoreJoueur>200) {
  bonus = 10;
}
```

Cette dernière écriture présente l'avantage de pouvoir contenir plusieurs lignes d'instructions entre les accolades, ce que ne permet pas la première syntaxe. Par ailleurs, il est également

très fréquent de rencontrer la syntaxe suivante, où la première accolade est renvoyée à la ligne.

```
if(scoreJoueur>200)
{
  bonus = 10;
}
```

Quelle que soit la syntaxe choisie, le mot-clé `if` doit toujours être écrit en minuscule et le test placé entre parenthèses. Ce dernier peut prendre plusieurs formes, suivant ce qu'on cherche à évaluer.

Une structure conditionnelle, telle que nous venons de l'écrire, se traduit par la phrase suivante : si le résultat du test décrit entre parenthèses est vrai, alors l'instruction, ou le groupe d'instructions, qui suit est exécuté. Lorsque vous n'avez qu'une seule ligne d'instruction à exécuter si le résultat du test est vrai, vous pouvez la saisir directement derrière la parenthèse qui ferme le test (comme dans le premier exemple) ; sinon vous devez décrire l'ensemble des instructions entres accolades (comme dans le deuxième ou le troisième exemple).

Les différentes formes de test

Avant de poursuivre la lecture de ce chapitre, il est important que vous compreniez le mécanisme d'un test.

Un test est une valeur booléenne qui est soit vraie (`true`), soit fausse (`false`). C'est le résultat d'une comparaison, comme dans l'exemple précédent avec l'instruction `scoreJoueur>200`, qui sera vraie si la valeur de la variable `scoreJoueur` est supérieure à `200`. Un test peut également être simplement la valeur d'une variable booléenne, comme une propriété.

Lors du codage, il faut s'assurer qu'un test renvoie toujours une valeur booléenne. Voici différents opérateurs que vous pouvez utiliser dans un test.

Les opérateurs

Dans la plupart des cas, vous aurez besoin de comparer des valeurs. Vous utiliserez alors des opérateurs pour tester une supériorité (`>`), une infériorité (`<`), une différence (`!=`) ou une égalité (`==`) entre deux valeurs.

Remarque

Il existe également les opérateurs `<=` et `>=`.

```
if(carteJoueurActif.x>250) carteJoueurActif.alpha=0.5;
if(score<2000) bonus=0;
if(motDePasse.text==250) carteJoueurActif.alpha=0.5;
if(reponse.text!="Non") score = score +1;
```

Remarque

Le test d'une égalité s'écrit toujours avec deux signes égale : ==. Si vous n'en saisissez qu'un, il ne s'agit plus d'un test, mais d'une affectation. Le résultat du test sera alors très différent de ce à quoi vous vous attendiez.

Si vous connaissiez les contextes d'exécution des quatre lignes ci-dessus, vous pourriez aisément déterminer si chaque test est vrai ou faux.

Voici un test, portant sur une propriété, qui pourrait être beaucoup plus concis, essayez de comprendre pourquoi…

```
if(boutonAlerte.visible==true) activationSaisie=true;
```

N'oubliez pas que la ligne d'instruction qui suit les parenthèses sera exécutée lorsque le test renverra la valeur `true`. Comme la propriété `boutonAlerte.visible` est de type booléen, la comparaison avec la valeur `true` est inutile. Il vous suffit de saisir :

```
if(boutonAlerte.visible) activationSaisie=true;
```

Une occurrence ne peut en effet qu'être visible (`visible==true`) ou pas (`visible==false`).

Voici un premier exemple qui permet d'appliquer un test dans un contexte précis. Il s'agit d'une occurrence représentant une balle qui se déplace de 5 pixels horizontalement et verticalement. Lorsqu'elle dépasse certaines limites sur la scène (simulant le choc contre un mur), nous changeons alors la valeur de la variable qui définit le pas initial du déplacement vertical ou horizontal.

Fichier de référence : `Chapitre9/if1.fla`

```
var vitesseHorizontale = 5;
var vitesseVerticale = 5;

balle.addEventListener(Event.ENTER_FRAME,deplacerBalle);

function deplacerBalle (evt:Event) {

  balle.x+=vitesseHorizontale;
  balle.y+=vitesseVerticale;

  if(balle.x>490) vitesseHorizontale = -3;
  if(balle.x<10) vitesseHorizontale = 3;
  if(balle.y>267) vitesseVerticale = -3;
  if(balle.y<10) vitesseVerticale = 3;

}
```

Remarque

Pour tester la position d'une occurrence sur la scène et notamment des dépassements de zones, ne testez pas avec une égalité mais plutôt avec les signes inférieurs ou supérieurs. Si l'on souhaite par exemple faire l'action suivante : si la balle arrive à droite de la scène qui fait 490 pixels de largeur, on la replace à gauche ; un programmeur inexpérimenté aura tendance à écrire `if (balle.x==490) balle.x=0`, alors qu'il faut impérativement écrire `if(balle.x>490) balle.x=0`. En effet, un déplacement d'occurrence ne se faisant pas toujours pixel par pixel, il y a un grand risque que votre occurrence ne passe pas précisément par le 490e pixel du bord gauche de la scène si le pas de déplacement est de plus d'un pixel.

Effectuer plusieurs tests avec && et ||

Vous aurez parfois besoin d'effectuer plusieurs tests avant d'exécuter une ou plusieurs lignes d'instructions. Vous les combinerez alors entre les parenthèses d'une structure conditionnelle en les séparant par les paires de signes `&&` et `||`.

La double esperluette (`&&`) s'assure que les deux tests renvoient la valeur `true` alors que le double pipe (prononcez le mot « pipe » à l'anglaise) vérifie qu'au moins un sur les deux renvoie la valeur `true`.

> **Remarque**
> Sur Mac, pour obtenir le caractère |, utilisez le raccourci Shift-Alt-L.

Étudions tout de suite deux exemples qui illustrent ces propos :

```
if(autorisationRelache && carteEnCours.x >450) inventaire++;
```

Pour que la variable `inventaire` soit incrémentée d'une unité, il faut que la valeur de la variable `autorisationRelache` ait pour valeur `1` (équivalent à `true`) et que l'occurrence intitulée `carteEnCours` soit située à plus de 450 pixels du bord gauche de la scène.

```
if(autorisationRelache || carteEnCours.x >450) inventaire++;
```

Dans ce second exemple, il suffit que l'une des deux conditions que nous venons d'évoquer renvoie la valeur `true`.

> **Définition**
> Pour qualifier ces tests multiples, on parle de conditions inclusives et exclusives.

Voici un autre exemple qui reprend et améliore le script d'une balle qui rebondit sur les bords de la scène. Nous gérons maintenant les changements de valeur des variables `vitesseHorizontale` et `vitesseVerticale` en fonction d'une double condition.

```
var vitesseHorizontale = 5;
var vitesseVerticale = 5;

balle.addEventListener(Event.ENTER_FRAME,deplacerBalle);

function deplacerBalle (evt:Event) {

  balle.x+=vitesseHorizontale;
  balle.y+=vitesseVerticale;

  if(balle.x>490 || balle.x<10) vitesseHorizontale*=-1;
  if(balle.y>267 || balle.y<10) vitesseVerticale*=-1;

}
```

Si l'occurrence en mouvement intitulée `balle` dépasse l'une des 4 limites mentionnées (490, 10, 267 et 10), un changement de vitesse a lieu sur l'une des deux directions de déplacement.

Pour conclure cette section, précisons que vous n'êtes pas obligé de vous limiter à deux tests entre les parenthèses de la structure conditionnelle ; vous pouvez en ajouter autant que vous le souhaitez.

```
if(scoreJoueur1>250000 && scoreJoueur2>25000 && scoreJoueur3>25000) bonusEquipe=2000;
```

if… else…

Vous aurez besoin, dans certains cas, de prévoir l'exécution d'une ou plusieurs lignes d'instructions si le test renvoie la valeur `false`. Nous devons donc adapter notre structure conditionnelle comme dans l'exemple suivant :

```
if(test) {
  ligne d'instruction à exécuter si test est vrai;
} else {
  ligne d'instruction à exécuter si test est faux;
}
```

Comme vous pouvez le constater, nous avons simplement ajouté le mot-clé `else` accompagné d'une ligne d'instruction placée entre accolades. Ainsi, lorsque le test renvoie la valeur `false`, c'est-à-dire lorsqu'il est faux, ces instructions seront exécutées. Dans l'exemple qui suit, lorsqu'une occurrence sera déplacée sur la scène par une technique de glisser-déposer, elle grossira de 200 % ou reviendra à son échelle d'origine (100 %) en fonction de la position où elle sera déposée.

Fichier de référence : `Chapitre9/if2.fla`

```
balle.addEventListener(MouseEvent.MOUSE_DOWN,deplacerBalle);
balle.addEventListener(MouseEvent.MOUSE_UP,relacherBalle);

function deplacerBalle(evt:Event) {
  evt.currentTarget.startDrag();
}

function relacherBalle(evt:Event) {
  stopDrag();
  if (evt.currentTarget.x>250) {
    evt.currentTarget.scaleX=2;
    evt.currentTarget.scaleY=2;
  } else {
    evt.currentTarget.scaleX=1;
    evt.currentTarget.scaleY=1;
  }
}
```

C'est grâce à la structure conditionnelle qui contient un `else` que nous pouvons réaliser une telle action.

Vous pouvez imbriquer plusieurs `if` : il suffit tout simplement d'ajouter un nouveau `if` avec des parenthèses et des accolades, si besoin est. Dans l'exemple qui suit, nous cherchons

à afficher sur la scène la position de l'occurrence intitulée balle. Elle se trouve soit dans la partie haute de la scène, soit dans la partie basse. Dans ce dernier cas, elle peut se situer à gauche ou à droite ; nous utilisons donc deux structures conditionnelles imbriquées pour tester sa position.

```
if (balle.y<300) {
  messageBalle.text="Haut";
} else {
  if (balle.x<400) {
    messageBalle.text="Bas Gauche";
  } else {
    messageBalle.text="Bas Droite";
  }
}
```

L'opérateur ternaire

Il existe une autre syntaxe pour écrire une structure conditionnelle de type if... else... Elle s'avère plus simple à utiliser et à rédiger dès lors que le principe de la structure conditionnelle est assimilé. Cependant sa lecture est assez surprenante au premier abord.

Opérateurs unaires, binaires et ternaires

Un opérateur unaire ne possède qu'un seul opérande, comme dans l'instruction `-a`, où l'opérateur `-` permet simplement de prendre l'opposé de `a`. Un opérateur binaire utilise deux opérandes, comme les opérateurs = et +, qui utilisent nécessairement deux constantes ou variables (`a=6` ou `b+c`). L'opérateur utilisé ici est un opérateur ternaire qui utilise trois opérandes.

Une structure conditionnelle écrite avec l'opérateur ternaire peut être perçue de la façon suivante :

```
Valeur à récupérer = test ? valeur à renvoyer si test est vrai : valeur à renvoyer
➥si test est faux.
```

Nous pourrions aussi la simplifier ainsi :

```
Valeur = test ? valeur si vrai : valeur si faux;
```

ou encore, si les valeurs à renvoyer sont `true` et `false`, par :

```
Valeur = test ? true : false;
```

Observez bien les signes typographiques de séparation `?` et `:`. Ils séparent simplement les deux valeurs à renvoyer. Si le test est vrai, la variable `Valeur` reçoit la valeur placée après le signe `?` ; si le test est faux, elle reçoit la valeur placée après le signe `:`.

Considérons la structure conditionnelle suivante :

```
  if (balle.x<400) {
    messageBalle.text="Bas Gauche";
  } else {
    messageBalle.text="Bas Droite";
  }
```

Elle peut être écrite à l'aide de l'opérateur ternaire ainsi :

```
messageBalle.text = balle.x<400 ? "Bas Gauche" : "Bas Droite";
```

Si cette syntaxe vous gène, vous pouvez également placer le test entre parenthèses.

```
messageBalle.text = (balle.x<400) ? "Bas Gauche" : "Bas Droite";
```

En outre, vous pouvez effectuer un test avec plusieurs conditions inclusives ou exclusives. Pour conclure sur ce développement, ajoutons que la valeur à récupérer n'est pas toujours de type `String`, comme c'est le cas dans l'exemple précédent ; vous pouvez faire appel à l'opérateur ternaire dans de nombreux cas.

Affectation d'une valeur à une variable :

```
resultat = puce.x>250 ? 25 : 35;
reduction = nombrePersonnes>=3? 25 : 15;
```

Affectation d'une valeur à un texte dynamique :

```
messageAccord.text = reponseCandidat.text=="Oui" ? "Bonne réponse" : "Mauvaise réponse";
```

Réglage d'une propriété :

```
carte.rotation = carte.alpha<0.5? 45 : -45;
```

Comparer deux chaînes de caractères

La comparaison de chaînes de caractères est parfois délicate. Considérons l'exemple ci-dessous.

```
if (reponseCandidat.text=="Oui") {
  messageAccord.text="Accord accepté";
} else {
  messageAccord.text="Accord refusé";
}
```

A priori, ce script ne contient aucune erreur. Vous avez tout à fait raison, la syntaxe est bonne, et pourtant l'utilisateur risque de voir s'afficher Accord refusé sur la scène s'il tape `oui` ou `OUI` dans le texte de saisie `reponseCandidat`. Observez bien la casse du texte sur la première ligne d'instruction : nous avons écrit `if (reponseCandidat.text=="Oui")`, ce qui signifie que l'utilisateur doit très précisément saisir sa réponse avec un `O` majuscule et les lettres `u` et `i` en minuscules. Dans le cas contraire, le test renvoie la valeur `false`. Pour éviter ce problème, ce qui revient à dire que nous ne voulons pas que le programme tienne compte de la casse de la réponse, vous devez effectuer la conversion suivante :

```
if (reponseCandidat.text.toUpperCase()=="OUI") {
  messageAccord.text="Accord accepté";
} else {
  messageAccord.text="Accord refusé";
}
```

Nous convertissons en majuscules la chaîne de caractères saisie par l'utilisateur afin de faire une comparaison avec le mot `OUI` (écrit en majuscules). Ainsi, l'utilisateur peut saisir la réponse oui sous la forme qu'il souhaite, `oui`, `Oui`, `OUI`, `OuI`, etc.

Pour conclure sur ce développement, précisons que généralement, les méthodes `toUpperCase()` et `toLowerCase()` ne servent pas à réaliser ce genre de tests, mais plutôt à changer la mise en forme d'un texte.

Effectuer un test avec switch()

Pour écrire une suite de structures conditionnelles, il existe l'instruction `switch()`. Le principe consiste à effectuer un branchement sur un bloc d'instructions suivant une condition.

Analysons ensemble la série de tests ci-dessous avec les méthodes `if()` et `switch()` :

```
if (reponse.text=="David") {
  scoreEcran+=3;
  ecranSuivant=2;
}
if (reponse.text =="Olivier") {
  scoreEcran+=3;
  ecranSuivant=2;
}
if (reponse.text =="Fabienne") {
  ecranSuivant=3;
}
if (reponse.text=="Jérôme") {
  scoreEcran+=4;
  ecranSuivant=3;
}
```

Et :

```
  switch (reponse.text) {
  case "David" :
    scoreEcran+=3;
    ecranSuivant=2;
    break;
  case "Olivier" :
    scoreEcran+=3;
    ecranSuivant=2;
    break;
  case "Fabienne" :
    ecranSuivant=3;
    break;
  case "Jérôme" :
    scoreEcran+=4;
    ecranSuivant=3;
}
```

Le code utilisant l'instruction `switch()` ne semble pas être plus long ou plus court, mais il est plus facile à comprendre (sans pour autant être plus facile à écrire !).

Le premier avantage d'une telle syntaxe est d'optimiser le traitement d'une condition car tous les tests ne vont pas obligatoirement être effectués. En effet, lorsque vous faites appel à l'instruction `switch()`, l'instruction `break` permet d'interrompre la série de tests si l'une des conditions est remplie. En revanche, dans le cas d'une série d'instructions de type `if()`, tous les tests sont effectués sans interruption, du premier, au dernier.

Imaginons que l'utilisateur saisisse `Olivier` dans le texte de saisie intitulé `reponse.text`. Dans ce cas, seules les deux premières comparaisons vont être effectuées et le `break` situé juste après `case "Olivier" :` va interrompre le processus de test. Vous noterez que la dernière évaluation ne contient pas de `break`, car cela n'est pas nécessaire.

Avec une instruction `if()`, nous pouvons prévoir le traitement à effectuer dans cas où le test est faux. Si vous utilisez un `switch()`, vous pouvez, de façon analogue, prévoir un cas par défaut avec l'instruction `default`, comme dans l'exemple suivant :

```
switch (reponse.text) {
  case "David" :
    scoreEcran+=3;
    ecranSuivant=2;
    break;
  case "Olivier" :
    scoreEcran+=3;
    ecranSuivant=2;
    break;
  case "Fabienne" :
    ecranSuivant=3;
    break;
  case "Jérôme" :
    scoreEcran+=4;
    ecranSuivant=3;
    break;
  default :
    ecranSuivant=7;
}
```

Pour écrire un script avec un processus de vérification faisant appel à l'instruction `switch()`, vous devez :

1. Écrire la structure qui englobe le processus.

```
switch() {

}
```

Saisie rapide

Vous pouvez utiliser le raccourci clavier Esc+S+W. Rappelons que vous devez appuyer puis relâcher la touche Esc puis faire de même avec la touche S et enfin avec la touche W.

2. Saisir le test à évaluer entre les parenthèses.

```
switch(reponse.text) {

}
```

Vous ne devez pas spécifier d'opérateurs de type `==`, `>`, `<` ou encore `!=`, car la structure globale d'une instruction `switch()` sous-entend que vous testez différentes égalités. Ce sont les mots `case` qui vont faire office de signe `=`.

3. Saisir les différents cas à évaluer.

```
case "David" :
  scoreEcran+=3;
  ecranSuivant=2;
```

Le mot `case` doit toujours être saisi avec une minuscule et la ligne d'instruction doit se terminer par deux points.

4. Pour terminer, n'oubliez pas l'instruction `break` pour interrompre le processus de vérification.

Ce type de test ne permet pas uniquement de tester la saisie d'un utilisateur. Examinons un dernier exemple qui évalue la position d'une occurrence en mouvement.

Fichier de référence : `Chapitre9/switch2.fla`

```
var vitesseHorizontale = 5;
var vitesseVerticale = 5;

balle.addEventListener(Event.ENTER_FRAME,deplacerBalle);

function deplacerBalle (evt:Event) {

  balle.x+=vitesseHorizontale;
  balle.y+=vitesseVerticale;

  switch (balle.x>=250) {

    case true :
    balle.alpha=0.3;
    break;

    case false :
    balle.alpha=1;
    break;
  }

  if(balle.x>490) vitesseHorizontale = -3;
  if(balle.x<10) vitesseHorizontale = 3;
  if(balle.y>267 || balle.y<10) vitesseVerticale*=-1;
}
```

La balle, qui se déplace et rebondit sur les quatre côtés de la scène, devient plus ou moins transparente en fonction de sa position sur l'axe horizontal.

La technique de test faisant appel à l'instruction `switch()` ne peut et ne doit pas remplacer le `if()` : elle est simplement plus adaptée dans certains cas.

10

Les itérations : boucles for()

Lorsque vous programmerez une animation, vous vous apercevrez que certaines lignes d'instructions sont uniques, alors que d'autres doivent être reproduites plusieurs fois. Prenons l'exemple d'une galerie de photos : il serait fastidieux de devoir placer des images sur la scène à l'aide de glisser-déposer de symboles ou de copier-coller de lignes d'instructions. En programmation, il existe un mécanisme qui permet d'obtenir une itération, c'est-à-dire une répétition d'actions : il s'agit des boucles `for()`. Elles permettent d'éviter de nombreux copier-coller de lignes de code. Notons que la vitesse d'exécution du code ainsi répété ne peut pas être configurée.

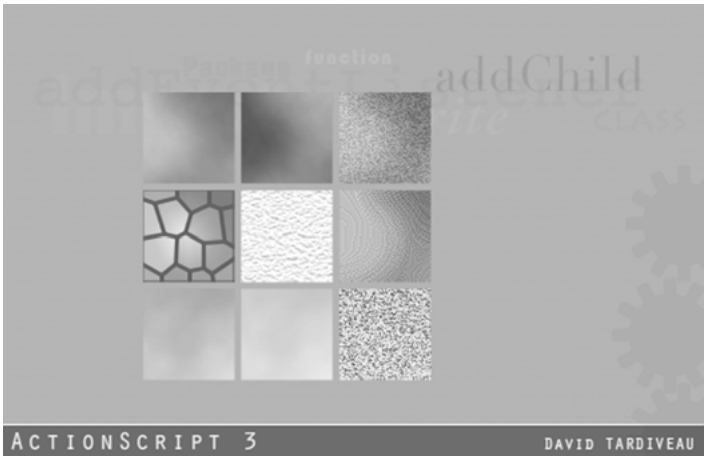


Figure 10-1

Les neuf images ont été placées sur la scène à l'aide d'une boucle for().

La boucle for()

Pour obtenir le résultat de la copie d'écran de la figure 10-1, il faudrait répéter plusieurs fois le groupe de lignes d'instructions ci-dessous en faisant varier les paramètres `img1.jpg`, `80` et `120`.

```
chargeur = new Loader();
adresseImage.url = "images/img1.jpg";
chargeur.load(adresseImage);
addChild(chargeur);
chargeur.x=80;
chargeur.y=120;
```

Non seulement le code deviendrait très long, augmentant ainsi considérablement la taille de votre script, mais copier neuf fois le même code serait très fastidieux, comme l'illustre l'extrait de script ci-dessous, dans lequel seuls quatre des neuf blocs de lignes nécessaires sont présentés :

```
chargeur = new Loader();
adresseImage.url = "images/img1.jpg";
chargeur.load(adresseImage);
addChild(chargeur);
chargeur.x=120;
chargeur.y=120;

chargeur = new Loader();
adresseImage.url = "images/img2.jpg";
chargeur.load(adresseImage);
addChild(chargeur);
chargeur.x=80;
chargeur.y=120;
…
chargeur = new Loader();
adresseImage.url = "images/img8.jpg";
chargeur.load(adresseImage);
addChild(chargeur);
chargeur.x=120;
chargeur.y=160;

chargeur = new Loader();
adresseImage.url = "images/img9.jpg";
chargeur.load(adresseImage);
addChild(chargeur);
chargeur.x=160;
chargeur.y=160;
```

Vous retrouverez plus loin dans ce chapitre le script correspondant à ce fonctionnement avec une boucle `for()` (`for6.fla`).

Avant de vous présenter de nombreux exemples, essayons de comprendre le fonctionnement et la structure d'une boucle `for()`.

1. Vous devez commencer par identifier les lignes d'instructions que vous souhaitez répéter et localiser les paramètres qui vont devoir varier. Il est généralement conseillé d'écrire une première fois le code que vous allez intégrer dans une boucle `for()`. En voici un exemple :

```
caseTableau = new CaseOmbree();
caseTableau.x = 90;
caseTableau.y = 150;
addChild(caseTableau);
```

Remarque

Les valeurs 90 et 150 sont celles que nous allons devoir changer à chaque itération.

2. Vous devez ensuite résoudre les problèmes liés à la répétition. Dans cet exemple, nous souhaitons placer un même symbole à différents endroits sur la scène ; il faut donc calculer les écarts entre chaque occurrence. Pour cela, nous effectuons l'estimation suivante : nous positionnons, à l'aide de la souris, deux occurrences sur la scène, puis nous déterminons l'écart entre leurs deux bords gauches ou leurs deux bords supérieurs.
3. Nous sommes fin prêts à créer notre première boucle ; commençons par la structure globale.

```
for() {

}
```

Saisie rapide

Vous pouvez utiliser le raccourci clavier Esc+F+R. Rappelons que vous devez appuyer puis relâcher la touche Esc, faire de même avec la touche F, puis appuyer sur la touche R.

Nous allons saisir entre les parenthèses, les informations nécessaires à la répétition des quatre lignes d'instructions, qui seront placées entre les accolades.

4. La répétition est basée sur l'utilisation d'un compteur, la variable `doigt` dans le code ci-dessous (comme si nous comptions sur les doigts de la main), dont nous allons préciser 3 paramètres, séparés par des points-virgules.

```
for(var doigt:Number=1 ; doigt<=5 ; doigt++ {

}
```

   – Le premier concerne la déclaration et l'initialisation de la variable locale qui sert à compter le nombre de répétitions. Nous aurions pu commencer à compter à partir de 0, mais nous avons préféré débuter à 1. Nous verrons plus loin dans ce chapitre qu'il est très souvent nécessaire d'initialiser cette variable à 0.
   – Le deuxième paramètre définit la condition de la répétition. Tant que cette condition sera vraie, les instructions placées entre les accolades seront répétées. Dans cet exemple,

nous avons utilisé l'opérateur `<=`, mais nous aurions également pu employer `<` ou `!=` (différent de). Les instructions de placement du symbole sur la scène seront donc répétées tant que la variable `doigt` aura une valeur inférieure ou égale à `5`. Dès que cette condition sera fausse, le lecteur Flash exécutera la première instruction située après les accolades.

– Bien entendu, il faut modifier la valeur de la variable qui sert de compteur à chaque répétition. C'est l'objet du troisième paramètre qui définit le pas d'évolution du compteur. Nous aurions pu écrire `doigt+=2` si nous avions souhaité compter de 2 en 2 ou bien `doigt+=0.1` si nous voulions un pas très petit. Très souvent, on compte de 1 en 1, et on écrit alors l'instruction `doigt = doigt+1` ou `doigt++`. Dans cet exemple, à chaque exécution du bloc d'instructions délimité par les accolades, la variable `doigt` est augmentée de 1 : elle prendra donc toutes les valeurs entières de `1` à `5`, ce qui donne bien cinq répétitions, ou cinq itérations.

5. Nous devons compléter le code par l'écriture des lignes d'instructions à répéter en substituant la ou les valeurs qui doivent changer par la variable locale ou par un calcul basé sur cette variable.

> **Remarque**
>
> Afin que notre script ne produise pas d'erreur, nous devons déclarer la variable `caseTableau` à laquelle nous faisons référence pour placer le symbole `CaseOmbree`.

Dans l'exemple que nous venons d'utiliser pour notre démonstration, nous avons simplement souhaité placer 4 fois le même symbole sur la scène. Verticalement, ils se trouvent tous à la même distance du haut de la scène (150 pixels), alors que, horizontalement, nous les décalons de 30 pixels. Dans la mesure où le symbole mesure 60 pixels de largeur, nous obtenons un décalage de 90 pixels. Ainsi, lorsque la variable `doigt` vaut `1`, `doigt*90` a pour valeur `90` ; lorsque `doigt` vaut `2`, `doigt*90` donne `180`, et ainsi de suite... Nous obtenons donc bien un pas de 90 pixels. Retrouvez cet exemple dans le fichier `for1.fla`.

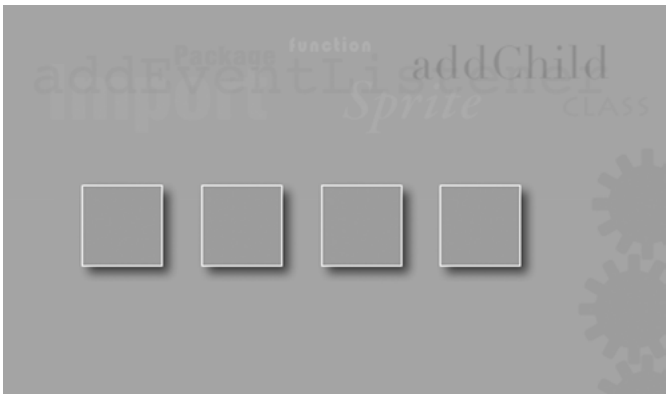


Figure 10-2

Résultat de l'exécution du script for1.fla

Voici le script complet :

```
var caseTableau:CaseOmbree;
for (var doigt=1; doigt <=4; doigt ++) {
  caseTableau = new CaseOmbree();
  caseTableau.x= doigt *90;
  caseTableau.y=150;
  addChild(caseTableau);
}
```

Premier exemple

Fichier de référence : `for1.fla`

`CaseOmbree` est le nom de la classe d'un symbole dans la bibliothèque. Pour vous aider à réaliser ce premier exemple, voici la procédure initiale manquante, celle qui consiste à créer le symbole avec liaison.

1. Tracez une forme sur la scène.
2. Sélectionnez-la.
3. Pressez la touche de fonction F8.
4. Précisez un point d'alignement, un nom de symbole et le type Clip.
5. Cliquez sur le bouton Avancé situé en bas à droite de la fenêtre afin d'obtenir une palette plus grande.

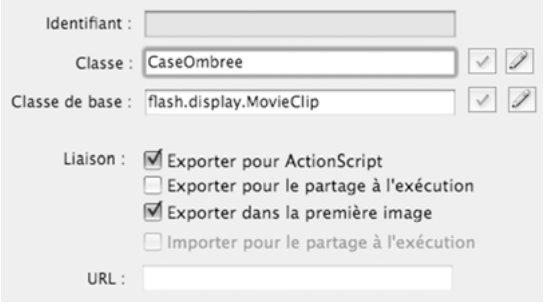


Figure 10-3

Vous devez définir un nom de classe pour pouvoir placer un symbole dynamiquement sur la scène.

6. Cochez la case Exporter pour ActionScript et donnez un nom de classe qui ne contient ni espace, ni caractères spéciaux ou accentués. Par convention, utilisez une majuscule en début de mot.
7. Validez.

Vous pouvez à présent placer le script que nous venons d'examiner dans une classe ou dans la fenêtre Actions.

Pourquoi initialiser une variable à 0 ?

Dans la mesure où nous avons généralement besoin de stocker des informations sous forme de listes pour pouvoir les exploiter à travers une itération, nous utilisons les schémas XML ou les tableaux. Or, le premier nœud d'un schéma XML et le premier élément d'un tableau portent l'index 0. Dans une boucle `for()`, il faudra donc utiliser une variable qui débute à 0 pour se référer à ces deux éléments : la variable compteur aura alors 0 pour valeur initiale.

Exemples

Dans la mesure où la technique des boucles `for()` est difficile à comprendre et à manipuler pour la première fois, nous allons aborder une série d'exemples vous démontrant ainsi la diversité des cas.

Exemple 1

Fichier de référence : `for2.fla`

Afin que vous vous concentriez sur la structure de la boucle `for()`, ce premier exemple ne fait rien en particulier sur la scène ; il utilise uniquement la fonction `trace()`.

```
for(vari=0;i<=10;i++) {
  trace(i);
}
```

Essayez simplement de substituer le contenu de notre fonction `trace()` par les lignes d'instructions ci-dessous. Faites varier également les valeurs 0 et 10 pour vérifier à chaque fois le résultat avec le raccourci Ctrl-Entrée (Windows) ou Commande-Entrée (Mac).

Concaténation : `trace("Joueur "+i);`

Calcul : `trace(i*10);`

Un chiffre aléatoire : `trace(Math.random()*100);`

Chiffre aléatoire arrondi : `trace(Math.round(Math.random()*100));`

Lire l'entrée d'un tableau : `trace(artiste[i]);`

Exemple 2

Fichier de référence : `for3.fla`

Ce deuxième exemple est une application que vous rencontrerez souvent et qui consiste à exploiter un tableau et à créer un texte dynamiquement sur la scène.

```
var prenoms:Array = new Array("Marine","Luna","David","Olivier","Marjorie");

var caseTableau:TextField;

for (var i=0; i<=4; i++) {
  caseTableau = new TextField();
```

```
    caseTableau.text = prenoms[i];
    caseTableau.x=i*90;
    caseTableau.y=150;
    addChild(caseTableau);
}
```

Nous commençons par déclarer une instance de la classe `TextField()` et une autre de la classe `Array()`, que nous initialisons.

Nous effectuons ensuite une boucle `for()` pour parcourir le tableau `prenoms`.

Remarque

Pour décaler l'ensemble des textes du bord gauche de la scène, il faudrait adapter la ligne d'instruction faisant référence à la propriété `x` de la façon suivante : `caseTableau.x=50+(i*90)`.

Exemple 3

Fichier de référence : `for4.fla`

L'exemple ci-dessous illustre simplement que le pas de la répétition n'est pas systématiquement un entier supérieur ou égal à 1. Consultez en particulier le chapitre 5 consacré aux mouvements sur la scène : l'un des exemples traite des cosinus et sinus, aspect non détaillé ici.

```
var puce:Puce;
var pas:Number = 0;

for (var i=0; i<=6.2; i+=0.1) {

  puce = new Puce();
  addChild(puce);
  puce.x=250+Math.cos(pas)*100;
  puce.y=150+Math.sin(pas)*100;
  pas+=0.1;

}
```

Exemple 4

Fichier de référence : `for5.fla`

Le script que nous allons aborder ci-dessous correspond à l'animation dont la copie d'écran est présentée sur la figure 10-1. La problématique du chargement d'image est récurrente en développement ; c'est pourquoi nous avons souhaité vous en présenter un exemple.

Fichier de référence : `for6.fla`

```
var accrocheX=100;
var accrocheY=40;

var chargeur;
var adresseImage = new URLRequest();
```

```
for (var i=1; i<=9; i++) {
  chargeur = new Loader();
  adresseImage.url = "images/img"+i+".jpg";
  chargeur.load(adresseImage);
  addChild(chargeur);
  chargeur.x=accrocheX;
  chargeur.y=accrocheY;

  accrocheX+=70;

  if (accrocheX>300) {
    accrocheX = 100;
    accrocheY+= 70;
  }
}
```

Exemple 5

Ce dernier exemple est intéressant car il présente de nombreuses notions mises en œuvre dans une boucle `for()`. Sans cette dernière, chaque pièce de notre puzzle devrait subir un traitement à part. En conclusion, plus les lignes d'instructions à répéter sont nombreuses, plus la boucle `for()` joue un rôle indispensable.

Fichier de référence : `fortween.fla`

```
import fl.transitions.easing.*;
import fl.transitions.*;

var accrocheX=100;
var accrocheY=40;

var mouvement:Tween;

var chargeur:Loader;
var cadreChargeur:Sprite;
var adresseImage = new URLRequest();

for (var i=1; i<=9; i++) {
  cadreChargeur = new Sprite();
  chargeur = new Loader();
  adresseImage.url = "piecespuzzle/case"+i+".jpg";
  chargeur.load(adresseImage);
  cadreChargeur.x=accrocheX+35;
  cadreChargeur.y=accrocheY+35;
  addChild(cadreChargeur);
  cadreChargeur.addChild(chargeur);
  chargeur.x=-35;
  chargeur.y=-35;

  cadreChargeur.rotation=360-(Math.round(Math.random()*4))*90;
  cadreChargeur.blendMode = BlendMode.MULTIPLY;

  accrocheX+=70;

  if (accrocheX>300) {
    accrocheX = 100;
    accrocheY+= 70;
  }
```

```
    cadreChargeur.addEventListener(MouseEvent.MOUSE_DOWN,tournerOccurrence);
    function tournerOccurrence(evt:MouseEvent) {
      var sens = (evt.shiftKey ? 90 : -90);
      var angleDepart:Number = evt.currentTarget.rotation;
      if (mouvement == null) {
        mouvement = new Tween(evt.currentTarget,"rotation",Regular.easeIn,angleDepart,
        ➥angleDepart+sens,0.5,true);
      } else if ( ! mouvement.isPlaying) {
        mouvement = new Tween(evt.currentTarget,"rotation",Regular.easeIn,angleDepart,
        angleDepart+sens,0.5,true);
      }
    }
}
```

La boucle for each()

Avant d'aborder ce type de boucle, vous devez bien connaître le mécanisme et la logique d'une boucle `for()`.

La boucle `for each()` est une variante de la boucle `for()`, mais le principe reste le même : répéter une action. Ce type de boucle est particulièrement adapté à la manipulation des tableaux, des arborescences XML et des objets pour récupérer des valeurs de propriétés. La différence avec une boucle `for()` porte sur les arguments placés entre parenthèses.

```
for each (var nomduneVariable:Type in Instance d'un objet) {
}
```

Plutôt que de vous expliquer théoriquement la nature et le rôle des arguments, examinons l'exemple suivant. Nous allons créer un tableau et écrire une itération sur ses éléments, mais agir comme si nous ne connaissions pas le nombre d'éléments qu'il contient. Pour cela, nous allons simplement utiliser une boucle `for each()`.

```
var semaine:Array = new Array("Lundi","Mardi","Mercredi","Jeudi","Vendredi",
➥"Samedi","Dimanche");

for each (var jour:String in semaine) {
  trace(jour);
}
```

Avec ce simple exemple, nous comprenons mieux le rôle des deux paramètres de l'instruction `for each()`. La variable `jour` sert à stocker, l'une après l'autre, toutes les valeurs d'un objet, d'un tableau ou d'une instance XML.

Le paramètre `semaine` correspond au nom du tableau, de l'instance XML ou de l'objet à parcourir. C'est le mot-clé `in` qui permet d'établir la liaison entre les deux. On parle alors de parcours d'une collection.

Un problème se pose tout de même ! Vous avez remarqué que nous devions typer la variable `jour`. Mais quel type faudrait-il utiliser si les valeurs de la collection à parcourir étaient de types différents ? Vous n'auriez pas d'autre choix que d'utiliser le type `*` et écrire : `var jour:* in semaine`.

Exemple 1

Dans cet exemple, nous n'allons pas parcourir un tableau, mais un objet (que nous pourrions tout de même appeler tableau associatif ou tableau de propriétés).

> **Remarque**
>
> `contenant` et `contenu` sont des textes dynamiques qui ont été placés sur la scène.

Fichier de référence : `foreachin1.fla`

```
var heure:Object = {heure:"De manger",minutes:25,secondes:13};

contenant.text="";
contenu.text="";

for each (var laValeur:* in heure) {
  contenant.appendText(laValeur+"\n");
}
```

Avant d'analyser ce script, rappelons qu'un objet possède des propriétés auxquelles on associe des valeurs sous la forme `nomPropriete:valeur` et non `nomPropriete=valeur`. Le nom de la propriété ne doit pas être saisi entre guillemets contrairement à sa valeur (s'il s'agit d'un texte). Une valeur de type nombre ne doit pas être écrite entre guillemets.

Dans l'exemple ci-dessus, grâce à la boucle `for each()`, nous ne cherchons pas à savoir combien de propriétés contient l'objet `heure`, ce que nous ferions dans le cas d'une boucle `for()`. Nous utilisons la syntaxe `var laValeur:* in heure`. La variable `laValeur` va prendre successivement les valeurs `De manger`, `25` et `13`. Si vous êtes habitué aux boucles `for()` classiques, cette technique surprend un peu, mais elle s'avère très vite incontournable pour un tel traitement.

Exemple 2

Fichiers de référence : `foreachin2.fla`, `main.as` et `Nuage.as`

> **Remarque**
>
> Consultez éventuellement le chapitre 16 dédié à la création de classes personnalisées pour comprendre le mécanisme d'une classe rattachée à un symbole.

Dans cet exemple, nous allons démontrer qu'il est possible de lire les propriétés définies dynamiquement pour une occurrence.

Notre premier fichier `foreachin2.fla` ne contient aucun script : seule la classe du document a été précisée en bas à droite de la palette Propriétés.

Nous avons également créé un symbole dont la liaison nous a permis de définir sa classe, comme nous l'avons présenté dans la figure 10-3. Nous l'avons intitulée `Nuage`.

Voici le script de cette classe.

```
package {

  import flash.display.Sprite;

  dynamic public class Nuage extends Sprite {

  function Nuage(placeX,placeY,echelle,transparence) {

    x=placeX;
    y=placeY;
    scaleX=echelle;
    scaleY=echelle;
    alpha=transparence;


  }
}
}
```

Un code extrêmement simple ! Nous avons simplement créé 5 propriétés dont les valeurs vont être définies dans la classe `main` (la classe du document) au moment de l'instanciation de la classe `Nuage()`.

```
package {

  import flash.display.Sprite;

  public class main extends Sprite {

      public var nuage1:Nuage;

    function main() {

      nuage1 = new Nuage(150,120,0.3,0.5);
      nuage1.temps="Mauvais";
      nuage1.vitesseVent=3;
      nuage1.chargeElectrique=230000;
      addChild(nuage1);

      for each (var laValeur:String in nuage1) {
        trace(laValeur);
      }

      for (var nomPropriete:String in nuage1) {
        trace(nomPropriete);
      }
    }
  }
}
```

Le résultat de l'exécution de l'animation est le suivant :

```
230000
Mauvais
3
chargeElectrique
temps
vitesseVent
```

Nous déclarons une variable nommée `nuage1` pour pouvoir instancier la classe `Nuage()`.

Nous définissons ensuite les propriétés que nous allons pouvoir lire à l'aide de la boucle `for each()`. Nous vous rappelons que ces propriétés peuvent être ajoutées car la classe `Nuage()` a été déclarée comme `dynamic`.

Lorsque nous cherchons ensuite à saisir le code de la boucle `for each()`, nous faisons référence à l'instance `nuage1`. Nous obtenons alors les valeurs des propriétés que nous avons définies.

La boucle for (in)

Avant d'aborder ce type de boucle, vous devez bien connaître le mécanisme et la logique d'une boucle `for()`.

Contrairement à la boucle `for each()`, une boucle `for (in)` ne permet pas de lire les valeurs des propriétés d'une instance mais plutôt ses propriétés. La structure est exactement la même, seuls le résultat ainsi que la syntaxe diffèrent.

Fichier de référence : `foreachin1.fla`

```
var heure:Object = {heure:"De manger",minutes:25,secondes:13};

contenant.text="";
contenu.text="";

for each (var laValeur:* in heure) {
  contenant.appendText(laValeur+"\n");
}

for (var nomPropriete:String in heure) {
  contenu.appendText(nomPropriete+"\n");
}
```

Vous noterez que le mot-clé `each` a bien été retiré et que le résultat est bien différent de celui produit par une boucle `for each()`.

Conclusion

La construction dynamique d'une interface passe généralement par des appels à des fichiers XML, des manipulations d'objets, des parcours de tableaux, des chargements multiples de médias et des placements dynamiques de symboles et de textes sur la scène : vous ne pourrez donc jamais vous dispenser d'utiliser des boucles !

Il est vrai que la manipulation des boucles constitue, pour les néophytes en programmation, un premier obstacle : cette logique d'écriture de code est relativement abstraite. Nous vous conseillons donc, lors de vos premières réalisations, de volontairement copier-coller les lignes de code assurant la même fonction afin de prendre conscience de la nécessité des boucles `for()`, `for each()` et `for (in)`, mais également de déterminer facilement les portions de code à répéter.

11

Les fonctions

La notion de fonctions est une notion élémentaire en algorithmique. Elle est indispensable en programmation, notamment en ActionScript 3 pour le système de gestion des événements.

Il en existe trois types :

- Les fonctions simples.
- Les fonctions avec paramètres.
- Les fonctions de rappel (qui fonctionnent avec les gestionnaires d'événements).

Une fonction est comparable à une macrocommande, un raccourci, une recette, c'est-à-dire une technique qui consiste à représenter ou appeler un ensemble d'actions par un ou quelques mots.

Prenons l'exemple suivant : une personne (un professeur) pourrait convenir avec d'autres (des étudiants) des conditions d'expressions suivantes :

- Myriam, lorsque je prononcerai ton prénom, tu te lèveras de ta chaise, tu avanceras jusqu'à l'interrupteur, tu appuieras sur le bouton de la lumière de la salle et tu reviendras t'asseoir à ta place.
- Adrien, lorsque je prononcerai ton prénom, tu te lèveras de ta chaise, tu te tourneras face à la classe et tu diras : il est l'heure de la pause.

Le professeur vient de donner des instructions à deux étudiants qui savent maintenant ce qu'ils doivent faire lorsqu'ils entendent leur prénom. Imaginons, à présent, qu'au bout de 5 minutes de parole, l'enseignant dise : « et voilà comment régler ce problème. Adrien. » À la suite de ce mot que vient de prononcer le professeur, une série d'actions va se dérouler : Adrien va se lever, il va se tourner vers la classe et dire « Il est l'heure de la pause ».

Grâce à cette technique, non seulement le professeur n'a pas besoin de répéter ce qu'il avait appris à Adrien, mais de plus, au prochain cours (avec le même public), il pourra dire à nouveau « Adrien » car ce dernier sait désormais ce qu'il doit faire. Il pourra, de la même façon, prononcer « Myriam » en entrant dans la salle !

Si vous avez compris qu'en prononçant un mot, on peut demander à une personne d'exécuter des instructions, voyons comment adapter ce système en programmation. La technique est extrêmement simple ! Vous allez commencer par initialiser votre fonction avant de l'appeler.

La fonction simple

Définition

Commencez par initialiser une fonction en écrivant la structure globale suivante :

```
function () {

}
```

Nous attirons tout de suite votre attention sur l'orthographe du mot `function` : il contient un `u` et non un `o`, cela est une source d'erreurs assez fréquente…

Vous devez définir un nom de fonction au même titre que nous avons défini dans les exemples précédents les noms Adrien et Myriam.

```
function placerTexte() {

}
```

Terminez la procédure en saisissant des lignes d'instructions à exécuter de façon analogue aux instructions qu'a données le professeur dans l'exemple précédent.

```
function placerTexte() {
  var messagePersonnel:TextField = new TextField();
  messagePersonnel.text = "Bonjour le monde !";
  addChild(messagePersonnel);
}
```

Vous avez à présent terminé l'initialisation de votre fonction ; nous allons pouvoir maintenant l'appeler dans un programme.

Appel d'une fonction

Comme vous l'avez peut-être déjà deviné, nous allons utiliser le nom `placerTexte` pour faire référence aux trois lignes d'instructions de la fonction, mais en y ajoutant des parenthèses. Vous étudierez dans la section suivante le rôle de ces parenthèses.

```
placerTexte();
```

L'exécution de cette instruction provoque l'affichage sur la scène de l'expression « Bonjour le monde ! ». Elle n'est pour l'instant ni positionnée, ni formatée, mais l'objectif était simplement d'illustrer l'exécution de lignes d'instructions à partir d'un simple nom.

La fonction avec paramètres

L'initialisation et l'appel d'une fonction sont deux techniques très simples, mais elles présentent pour l'instant principalement deux limites majeures avec cette syntaxe.

- Le texte est toujours le même.
- Sa position est toujours la même.

Pour pallier ce problème, vous allez devoir créer une fonction avec paramètres qui se caractérise par l'ajout, entre les parenthèses, des noms que nous devons qualifier de variables. Ils vont porter les valeurs que nous aurons précisées au moment de l'appel de la fonction. Ces variables seront également utilisées dans les lignes d'instructions qui se trouvent à l'intérieur de la fonction, et seront remplacées par leurs valeurs respectives au moment de l'appel.

La fonction avec paramètres diffère de la fonction simple (sans paramètre) à deux niveaux :

- Au moment de l'initialisation de la fonction, nous devons prévoir des variables locales.
- Au moment de l'appel de la fonction, nous devons définir (renseigner) ces variables.

En reprenant notre exemple initial, imaginons que nous souhaitions écrire un texte personnalisé et le placer à un endroit précis de la scène. Voilà comment nous devons adapter le code :

```
function placerTexte(texteDuMessage:String,positionX:Number, positionY:Number ) {
  var messagePersonnel:TextField = new TextField();
  messagePersonnel.text = texteDuMessage;
  addChild(messagePersonnel);
  messagePersonnel.x=positionX;
  messagePersonnel.y=positionY;
}
...
placerTexte("1850",160,150);
placerTexte("1900",200,150);
placerTexte("1950",240,150);
placerTexte("2000",280,150);
```

L'ordre dans lequel sont définis les paramètres doit être respecté lors de l'appel de la fonction. En revanche, dans les lignes d'instructions contenues entre les accolades, ces paramètres peuvent être présents, éventuellement à plusieurs reprises, à n'importe quel endroit. Au moment de l'appel de la fonction, nous devons donc fournir la valeur de chaque paramètre ; sans cela, une erreur se produit. Avec cette technique, nous avons évité de copier-coller plusieurs lignes de code !

La fonction de rappel

Les fonctions de rappel ont été abordées dans le chapitre 3 : La gestion des événements. Une fonction de rappel n'est ni plus ni moins qu'une fonction telle que celles que nous venons de présenter depuis le début de ce chapitre. Cependant, elle présente la particularité de contenir en paramètre un nom d'objet, accompagné de son type, pour recevoir des informations envoyées par l'écouteur qui l'invoque (qui l'appelle).

Prenons cet exemple :

```
function masquerEtoile(evt:MouseEvent) {
  etoileBlanche.visible=false;
}
```

Cette fonction ressemble à celles que nous venons d'étudier précédemment avec un paramètre, mais elle ne va pas être appelée comme nous l'avons vu. Elle sera invoquée par un écouteur.

```
etoile.addEventListener(MouseEvent.MOUSE_DOWN,masquerEtoile);
```

Vous remarquez que l'appel ne contient pas de parenthèses : il est donc impossible de passer simplement des paramètres à cette fonction. Voici tout de même une technique.

Utiliser des paramètres avec une fonction de rappel

La technique que nous allons aborder n'est pas conforme aux règles de programmation en AS3, mais elle constitue une solution au problème posé et respecte les règles de la programmation orientée objet.

Comme nous n'avons pas le droit d'écrire :

```
etoile.addEventListener(MouseEvent.MOUSE_DOWN,masquerEtoile,nominstance);
```

nous devons trouver une solution pour pouvoir personnaliser l'exécution de la fonction selon l'occurrence cliquée.

Vous travaillerez généralement avec des tableaux ou des fichiers XML, dont vous allez avoir besoin afin de récupérer un numéro d'index propre à une occurrence. Voici un exemple qui montre comment obtenir cet index sans utiliser une boucle `for()`.

1. Commencez par attribuer un numéro unique à chaque occurrence à partir desquelles une action différente devra être réalisée.

```
etoile1.sonNumero=1;
etoile2.sonNumero=2;
```

2. Dans la fonction de rappel, vous allez utiliser la propriété `currentTarget` pour pouvoir faire référence à la propriété `sonNumero`.

```
function masquerEtoile(evt:MouseEvent) {
  trace(evt.currentTarget.sonNumero);
}
```

3. Ajoutez des écouteurs aux occurrences sur lesquelles vous souhaitez que l'utilisateur interagisse (clic, survol, etc.).

```
etoile1.addEventListener(MouseEvent.MOUSE_DOWN,masquerEtoile);
etoile2.addEventListener(MouseEvent.MOUSE_DOWN,masquerEtoile);
```

Voilà ce qui se passe précisément lorsque l'utilisateur clique sur une occurrence.

Cette dernière possède des propriétés, qui vont être envoyées à l'objet que vous avez indiqué en paramètre de la fonction de rappel, au moment de la constatation de l'événement. Pour y accéder à partir des lignes d'instructions qu'elle contient, vous devez donc faire référence à ce même mot (le nom de l'objet) et aux propriétés disponibles. Celle qui s'intitule `currentTarget` permet de faire référence à l'occurrence cliquée.

Deux autres solutions sont également envisageables :

- Utiliser l'instruction `switch()` pour exécuter des lignes d'instructions différentes.
- Traiter le nom de l'occurrence pour extraire un suffixe sous forme de numéro, ce qui permet d'éviter de définir une propriété pour chaque occurrence.

12

Le chargement de médias sous forme de fichiers externes

En matière de multimédia, ce chapitre est sûrement le plus important car il traite des techniques et problèmes de chargement de fichiers externes de types texte, son, image et vidéo. Le sujet de cet ouvrage étant le langage ActionScript 3, il serait impensable que nous vous fassions importer les médias à partir du menu Fichier de l'IDE de Flash. Nous allons donc gérer le chargement et le contrôle des médias à partir de lignes d'instructions, ce qui présente les avantages suivants :

- L'aspect dynamique est pris en considération.
- Le poids des fichiers SWF s'en trouve allégé.
- La mise à jour des SWF est facilitée.
- Certaines manipulations ne peuvent être réalisées qu'en ActionScript.

Sans compter le plaisir éprouvé à gérer des lignes d'instructions !

Avant de vous lancer dans la réalisation de gros projets contenant des médias textes, images et vidéos, nous vous conseillons de lire le chapitre 2, si vous ne l'avez pas déjà fait. Cela vous permettra de mieux contrôler le chargement, le positionnement et le déchargement de vos médias.

Charger une image sur la scène

Pour charger une image sur la scène, nous allons devoir faire appel à deux classes distinctes :

- La classe `Loader()`.
- La classe `URLRequest()`.

Cette dernière va servir à générer une instance dont le rôle est d'encapsuler l'adresse de stockage de l'image, alors que la première va créer une instance servant de conteneur à l'image chargée sur la scène.

Fichier de référence : `Chapitre12/ChargerImage.fla`

1. Commencer par créer deux instances des classes que nous venons d'évoquer.

```
var chargeur:Loader = new Loader();
var adresseImage:URLRequest = new URLRequest("image1.jpg");
```

   Si vous connaissez déjà l'adresse de l'image, comme dans cet exemple, précisez-la dès le départ. Dans le cas contraire, il faudra la préciser à l'aide de la propriété `url`.

2. Chargez l'image dans le conteneur et ajoutez l'objet d'affichage à la liste d'affichage.

```
chargeur.load(adresseImage);
addChild(chargeur);
```

3. Terminez la manipulation en plaçant votre image (son coin supérieur gauche) à l'endroit désiré sur la scène.

```
chargeur.x=100;
chargeur.y=50;
```

Vous devriez obtenir le script suivant :

```
var chargeur:Loader = new Loader();
var adresseImage:URLRequest = new URLRequest("image1.jpg");
chargeur.load(adresseImage);
addChild(chargeur);

chargeur.x=100;
chargeur.y=50;
```

Certains développeurs préfèrent instancier directement la classe `URLRequest()` en paramètre de la méthode `load()` de la classe `Loader()` ; cela n'est pas une obligation. Cette technique présente l'inconvénient de rendre le code moins accessible ou tout du moins plus difficile à lire.

```
var chargeur:Loader = new Loader();
chargeur.load(new URLRequest("image1.jpg"));
addChild(chargeur);

chargeur.x=100;
chargeur.y=50;
```

Cette technique peut paraître complexe à celles et ceux qui avaient l'habitude de charger des images en AS2 avec un simple `loadMovie()` ou `loadMovieNum()`. En revanche, vous allez découvrir que la gestion du chargement est plus simple. De plus, l'ajout d'un événement de type `MouseEvent` est plus facile à gérer. Souvenez-vous : il fallait attendre la fin du chargement de l'image avant de pouvoir lui affecter un gestionnaire d'événement, ce qui n'est plus le cas aujourd'hui. Enfin, cette technique ne devrait pas sembler trop complexe à celles et ceux qui utilisaient un `MovieClipLoader()`, à l'exception de l'instanciation de la classe `URLRequest()`.

Rendre une image cliquable

Vous aurez parfois besoin de rendre cliquable une image chargée sur la scène. Pour cela, il suffira de créer un écouteur (ajouter un gestionnaire d'événement).

```
chargeur.addEventListener(MouseEvent.MOUSE_DOWN,clicImage);

function clicImage(evt:MouseEvent) {
  chargeur.blendMode = BlendMode.MULTIPLY;
}
```

En reprenant notre script initial, vous devriez obtenir le code suivant :

Fichier de référence : `Chapitre12/ChargerImage2.fla`

```
var chargeur:Loader = new Loader();
var adresseImage:URLRequest = new URLRequest("image1.jpg");
chargeur.load(adresseImage);
addChild(chargeur);

chargeur.x=100;
chargeur.y=50;

chargeur.addEventListener(MouseEvent.MOUSE_DOWN,clicImage);
chargeur.addEventListener(MouseEvent.MOUSE_UP,relacherClicImage);

function clicImage(evt:MouseEvent) {
  chargeur.blendMode = BlendMode.MULTIPLY;
}

function relacherClicImage(evt:MouseEvent) {
  chargeur.blendMode = BlendMode.NORMAL;
}
```

Point d'alignement d'une image

Le point d'alignement d'une image chargée sur la scène se trouve toujours en haut à gauche. Vous le constateriez aisément si vous cherchiez à faire tourner une occurrence, la rotation s'effectuant alors autour de ce point. Cela est également vrai lors de la mise à l'échelle d'une image.

Comme nous l'évoquions au début de ce chapitre, il sera bien souvent nécessaire de charger les médias dans des conteneurs d'objets d'affichage. C'est ce qui permet de résoudre ce problème, comme nous allons le voir dans cet exemple.

Fichier de référence : `Chapitre12/ChargerImage3.fla`

1. Commençons par créer une instance de la classe `Sprite()` que nous ajoutons à la liste d'affichage et que nous plaçons sur la scène.

```
var cadre:Sprite = new Sprite();
addChild(cadre);
cadre.x=250;
cadre.y=150;
```

2. Nous effectuons ensuite le chargement de l'image comme précédemment, mais nous allons ajouter l'instance de la classe `Load()` dans l'instance de la classe `Sprite()` tout juste créée.

```
var chargeur:Loader = new Loader();
var adresseImage:URLRequest = new URLRequest("image1.jpg");
chargeur.load(adresseImage);
cadre.addChild(chargeur);
```

Pour l'instant, l'image est chargée et son coin supérieur gauche est aligné sur le centre de l'instance de type `Sprite()`. Nous devons donc décaler l'image vers la gauche et vers le haut. Comme nous connaissons les dimensions de l'image, nous les utilisons directement dans le code, mais il serait préférable de lire les valeurs renvoyées par les propriétés `width` et `height`. Il faut pour cela que vous utilisiez un gestionnaire d'événement que nous allons découvrir dans le prochain développement, Gérer la fin du chargement d'une image.

```
chargeur.x=-105;
chargeur.y=-105;
```

Ajoutez ces quelques lignes d'instructions et constatez que la rotation s'effectue correctement autour du centre de l'image.

```
cadre.addEventListener(MouseEvent.MOUSE_DOWN,clicImage);

function clicImage(evt:MouseEvent) {
  cadre.rotation+=10;
}
```

Animation de préchargement

Vous devrez probablement informer l'utilisateur de votre programme qu'une image est en cours de chargement. Pour cela, vous allez devoir utiliser un gestionnaire faisant appel à l'événement `PROGRESS` de la classe d'événement `ProgressEvent`.

Fichier de référence : `Chapitre12/ChargerImage4.fla`

Commençons notre script en chargeant l'image comme nous l'avons effectué dans les pages précédentes :

```
var chargeur:Loader = new Loader();
var adresseImage:URLRequest = new URLRequest("image1.jpg");
chargeur.load(adresseImage);
addChild(chargeur);

chargeur.x=140;
chargeur.y=30;
```

Nous ajoutons ensuite un gestionnaire d'événement, mais observez bien la ligne d'instruction ci-dessous : nous faisons d'abord référence à la propriété `contentLoaderInfo`. Nous n'associons pas directement la fonction `addEventListener()` à notre instance chargeur.

```
chargeur.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,chargementEnCours);

function chargementEnCours(evt:ProgressEvent) {
  jauge.scaleX=evt.currentTarget.bytesLoaded/evt.currentTarget.bytesTotal;
}
```

L'écouteur mentionne la fonction de rappel `chargementEnCours` qui contient l'animation. C'est donc cette ligne d'instruction que vous devez changer si vous souhaitez réaliser une autre animation.

Gérer la fin du chargement d'une image

Fichier de référence : `Chapitre12/ChargerImage4.fla`

Si ce n'est déjà fait, prenez connaissance du développement précédent, Animation de préchargement, avant de poursuivre votre lecture.

La gestion de la fin du chargement d'une image fonctionne de la même façon que son préchargement : vous devez utiliser un écouteur, mais pas avec la même classe.

```
chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementTermine);

function chargementTermine(evt:Event) {
  jauge.visible =false;
}
```

Dans la section Point d'alignement d'une image, nous avons évoqué la possibilité de déterminer les dimensions d'une image. Voilà comment procéder :

```
chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementTermine);
function chargementTermine(evt:Event) {
  jauge.visible =false;
  trace(evt.currentTarget.width);
  trace(evt.currentTarget.height);
}
```

Supprimer une image chargée

La suppression d'une image s'effectue en deux étapes. Vous devez commencer par retirer votre objet d'affichage de la liste d'affichage. Précisons que l'objet d'affichage correspond à l'instance de la classe `Load()` que nous avons créé pour charger l'image ou à celle de la classe `Sprite()` qui contient le chargeur de type `Load()`. Ensuite vous devez assigner la valeur `null` à l'instance que vous venez de retirer de la liste d'affichage.

Créer une classe de chargement d'image

Si vous devez régulièrement charger des images dans une animation, il est conseillé de créer une classe dédiée que vous n'aurez plus qu'à instancier pour charger une simple image.

Fichiers de référence : `Chapitre12/ChargerImage5.fla`, `main.as` et `ChargerImage.as`

Dans cet exemple, nous avons créé une classe intitulée `ChargerImage()` dans laquelle nous préciserons, lors de son instanciation, le nom de l'image et celui du dossier dans lequel elle se trouve.

Dans le fichier `ChargerImage5.fla`, nous avons défini le nom de la classe du document `main.as`, en bas à droite de la palette Propriétés. Dans la classe `ChargerImage.as`, voici les lignes d'instructions qui ont été saisies.

```
package {

  import flash.display.Sprite;
  import flash.display.Loader;
  import flash.net.URLRequest;

  dynamic public class ChargerImage extends Sprite {

  public var adresseImage:URLRequest;
  public var chargeur:Loader;

  public function ChargerImage(nomDossier:String,nomImage:String) {

    adresseImage=new URLRequest(nomDossier+"/"+nomImage);
    chargeur=new Loader();
    chargeur.load(adresseImage);
    addChild(chargeur);
  }
}
}
```

Vous remarquerez qu'il s'agit des mêmes lignes d'instructions que celles qui se trouvent dans le premier exemple. Cela signifie donc que cette classe n'est pas très compliquée à écrire, dès lors que vous maîtrisez la programmation orientée objet. Allons voir à présent à quoi ressemble le fichier `main.as`.

```
package {

  import flash.display.Sprite;

  public class main extends Sprite {

    public var imageDeFond:ChargerImage;
    public var imageLogo:ChargerImage;

    function main() {

      imageDeFond = new ChargerImage("images","imageFond.png");
      imageLogo = new ChargerImage("images","logoYazo.png");

      addChild(imageDeFond);
      addChild(imageLogo);

      imageLogo.x=20;
      imageLogo.y=20;
    }
  }
}
```

Ce script n'est pas plus complexe que la classe de chargement d'une image. Nous instancions simplement la classe `ChargerImage()` en précisant le nom du dossier qui contient l'image puis le nom du fichier image, dans cet ordre. Nous terminons le script en ajoutant les deux images dans la liste d'affichage et nous plaçons le logo. Nous ne plaçons pas l'image de fond car elle possède les mêmes dimensions que la scène et vient se charger automatiquement dans le coin supérieur gauche.

Charger et contrôler un son

Commençons par rappeler que seuls les fichiers au format MP3 sont gérés en ActionScript, alors que vous pouvez importer n'importe quel type dans la bibliothèque via la commande Importer du menu Fichier. Comme, dans cet ouvrage, nous traitons de l'ActionScript, il va de soi que nous déconseillons fortement l'importation d'un fichier son dans une animation.

Pour gérer le son dans une animation, vous devez utiliser plusieurs classes. Le langage AS3 en compte six, mais vous aurez besoin de faire appel uniquement à deux ou trois d'entre elles pour manipuler le son au quotidien.

Lancer un son

Pour une simple lecture de son, nous devons commencer par instancier les classes `URLRequest()` et `Sound()`.

Fichier de référence : `Chapitre12/son1.fla`

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var enceinte = new Sound(cheminSon);
```

Vous indiquez ainsi à Flash l'adresse du fichier, mais vous lui demandez également de se préparer à la lecture d'un son. Pour déclencher celui-ci, il ne vous reste plus qu'à exécuter la commande `play()`.

```
enceinte.play();
```

Si vos besoins en matière de son ne sont pas plus importants, vous pouvez vous arrêter là, mais nous vous conseillons tout de même de poursuivre la lecture des paragraphes ci-dessous car vous allez sûrement découvrir des fonctionnalités que vous pourriez utiliser.

Les trois premières lignes de code que nous venons de découvrir pourraient être écrites différemment si nous chargions le son avec la méthode `load()`.

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var enceinte = new Sound();
enceinte.load(cheminSon);
enceinte.play();
```

Si vous travaillez sur une petite application, cette méthode est comparable à la première, le script contient juste une ligne supplémentaire. En revanche, dans le cas de plus gros scripts, vous déclarerez généralement vos deux instances dès les premières lignes

d'instructions et vous les instancierez plus loin dans le programme. Vous pourriez d'ailleurs faire la même chose avec l'instance de la classe `URLRequest()`.

```
var cheminSon:URLRequest = new URLRequest();
var enceinte = new Sound();

cheminSon.url = "Instru4.mp3";
enceinte.load(cheminSon);

enceinte.play();
```

Lire un son à partir d'un instant précis

Dans l'exemple ci-dessus, nous avons effectué une simple lecture avec la méthode `play()`.

Nous aurions pu ajouter un paramètre qui mentionne la position de la tête de lecture au moment du démarrage du son. Essayez le script précédent avec la valeur suivante :

```
enceinte.play(21000);
```

La valeur passée en paramètre à la méthode `play()` est exprimée en millisecondes, ce qui donne, dans cet exemple, une lecture à partir de la 21e seconde.

Lire un son en boucle

Les morceaux de musique que nous vous avons préparés avec GarageBand et fournis avec les exemples de ce livre ont été réalisés de telle sorte qu'ils puissent être mis en boucle. Pour lire un son un nombre déterminé de fois ou à l'infini, il vous suffit de préciser un deuxième paramètre à la méthode `play()`.

```
enceinte.play(0,4);
```

Le premier paramètre précise le temps à partir duquel la lecture doit démarrer, le deuxième précise le nombre de boucles. Indiquez une valeur très élevée pour obtenir une lecture en continu.

Remarque

Pour qu'il ne se produise pas de blanc lors de l'écoute d'un son en boucle, il faut traiter le fichier son afin qu'il ne possède aucun silence ou décalage de niveau d'onde entre la fin et le début du son.

Exemple

Pour finir nos explications sur la lecture d'un son, voici un dernier script qui démontre la possibilité d'associer un son à un bouton. Pour l'instant, il n'est pas question d'arrêter le son, nous allons juste le lancer.

Remarque

Il aurait été préférable de faire appel à un tableau et une boucle `for()` pour écrire ce script, mais dans un souci d'accessibilité à un plus grand nombre, nous ne l'avons pas fait.

Fichier de référence : `Chapitre12/son5.fla`

```
var cheminSon:URLRequest = new URLRequest();
var enceinte:Sound;

btSon1.addEventListener(MouseEvent.MOUSE_DOWN,lireSon);
btSon2.addEventListener(MouseEvent.MOUSE_DOWN,lireSon);
btSon3.addEventListener(MouseEvent.MOUSE_DOWN,lireSon);
btSon4.addEventListener(MouseEvent.MOUSE_DOWN,lireSon);

btSon1.sonURL = "Instru1.mp3";
btSon2.sonURL = "Instru2.mp3";
btSon3.sonURL = "Instru3.mp3";
btSon4.sonURL = "Instru4.mp3";

function lireSon(evt:MouseEvent) {

  cheminSon.url = evt.currentTarget.sonURL;
  enceinte =  new Sound(cheminSon);
  enceinte.play();

}
```

Arrêter un son

Commençons ce développement par cette question : serait-il envisageable de gérer le son de façon professionnelle avec une seule piste son ? N'importe quel ingénieur du son vous répondrait non !

Sous sa forme actuelle, le script que nous avons saisi ne nous permet pas un contrôle parfait du son. Nous devons l'associer à une piste, ce que nous ferons au moment de la lecture. Pour cela, nous devons instancier la classe `SoundChannel()` qui, elle, possède la méthode `stop()` ce qui n'est pas le cas de la classe `Sound()`.

Fichier de référence : `Chapitre12/son2.fla`

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var enceinte = new Sound(cheminSon);
var piste1:SoundChannel;
```

Nous avons placé sur la scène deux occurrences, intitulées `btLecture` et `btArret`, et nous leur associons deux gestionnaires d'événement.

```
btLecture.addEventListener(MouseEvent.MOUSE_DOWN,lireSon);
btArret.addEventListener(MouseEvent.MOUSE_DOWN,arreterSon);
```

Voici à présent les fonctions de rappel `lireSon()` et `arreterSon()`.

```
function lireSon(evt:MouseEvent) {
  piste1 = enceinte.play();
}

function arreterSon(evt:MouseEvent) {
  piste1.stop();
}
```

Elles sont très courtes et très simples car elles se limitent au lancement et à l'arrêt du son. Comme nous le mentionnions plus haut, nous avons utilisé l'instance de la classe `SoundChannel()` dans la ligne d'instruction de lecture du son.

En conclusion, pour arrêter un son vous devez non seulement faire appel à la méthode `stop()`, mais également assigner la lecture d'un son à une piste son, que vous créez à l'aide la classe `SoundChannel()`.

Exemple

Si vous utilisez correctement le fichier `son2.fla`, vous n'aurez pas de surprise. Mais essayez de cliquer plusieurs fois sur le bouton Lecture et de cliquer sur le bouton Arrêt sans avoir préalablement cliqué sur le bouton Lecture. Que constatez-vous ? La lecture du son est démultipliée à chaque clic sur le bouton Lecture et un message d'erreur apparaît si le premier clic se fait sur le bouton Arrêt ! Pour pallier ces problèmes, voici un nouveau script qui gère également l'apparence des boutons Lecture et Arrêt.

Fichier de référence : `Chapitre12/son3.fla`

```
btArret.alpha=0.5;

var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var enceinte = new Sound(cheminSon);
var piste1:SoundChannel;

btLecture.addEventListener(MouseEvent.MOUSE_DOWN,lireSon);

function lireSon(evt:MouseEvent) {
  piste1 = enceinte.play();
  btLecture.mouseEnabled = false;
  evt.currentTarget.alpha=0.5;
  btArret.alpha=1;
  btArret.addEventListener(MouseEvent.MOUSE_DOWN,arreterSon);
}

function arreterSon(evt:MouseEvent) {
  piste1.stop();
  btLecture.mouseEnabled = true;
  btLecture.alpha=1;
  btArret.alpha=0.5;
}
```

L'analyse de ce script est intéressante car sa construction est particulière. Voici quelques remarques à son sujet :

- Tout d'abord, vous noterez que nous avons rendu le bouton Arrêt transparent pour exprimer le fait qu'il n'est pas cliquable.
- Nous créons un écouteur qui gère la fonction `arreterSon()` seulement à partir du moment où l'utilisateur à cliqué sur le bouton Lecture.
- Nous utilisons la propriété `mouseEnabled` pour interdire le clic sur le bouton Lecture lorsque la lecture du son est en cours.

- Nous changeons l'apparence des boutons Lecture et Arrêt en faisant varier leur opacité. Nous aurions tout autant pu utiliser la propriété `blendMode` (par exemple : `btLecture.blendMode = BlendMode.SCREEN`).

> **Attention**
>
> Nous n'allons pas l'expliquer maintenant, mais cette application contient une erreur qui est abordée dans le développement Gérer la fin de la lecture d'un son.

Contrôler le niveau sonore

> **Remarque**
>
> Afin que les explications relatives au réglage du son soient très claires nous ne reprendrons pas le script précédent qui permet de contrôler le lancement et l'arrêt du son.

Le réglage du niveau sonore est considéré comme une transformation. C'est pourquoi nous devons utiliser une classe supplémentaire, `SoundTransform()`, à partir de laquelle nous allons créer une instance. Nous lirons alors la propriété `soundTransform`, qui contient d'autres propriétés telles que `volume` ou `pan`. Le réglage du niveau sonore s'effectue au niveau de la piste et non sur l'instance de la classe `Sound()`.

Commençons notre script par quelques lignes d'instructions qui nous permettent de lancer un son.

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var chainehifi = new Sound(cheminSon);

var enceinteFacade:SoundChannel;
enceinteFacade = chainehifi.play();
```

Ensuite, nous devons instancier la classe `SoundTransform()` pour pouvoir accéder ultérieurement à la propriété `volume`.

```
var reglagesAudio:SoundTransform = enceinteFacade.soundTransform;
```

Vous noterez que nous instancions la classe tout en chargeant préalablement les réglages par défaut de l'instance `enceinteFacade`. Nous pouvons maintenant faire référence à cette instance et à la propriété `volume`.

```
reglagesAudio.volume = 0.8;
```

> **Remarque**
>
> Les valeurs de réglage du niveau sonore s'étalent de 0 à 1. Utilisez la valeur 0.5 pour un réglage à 50 %. Au-delà de 1 et jusqu'à 4, le son sera de plus en plus fort, mais il perdra en qualité d'écoute. Lorsque vous dépassez 4, le son devient vraiment inaudible. Notons tout de même que cette valeur peut varier selon la qualité des enceintes de votre ordinateur.

Enfin, le réglage du niveau sonore est obtenu à l'aide de la propriété `soundTransform`.

```
enceinteFacade.soundTransform = reglagesAudio;
```

Dans le cas où vous souhaiteriez réaliser un variateur pour régler le volume ou la balance d'un son, consultez l'animation `draganddrop4.fla` du chapitre 4, qui traite de l'exécution d'une action lorsque vous déplacez un curseur sur une graduation.

Pour conclure, voici deux scripts complets qui vous permettront d'avoir un aperçu du code nécessaire pour gérer le son dans une animation.

Exemple 1

Ce premier script permet simplement de lancer un son et de régler son niveau sonore à 80 % de son intensité d'origine.

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var chainehifi = new Sound(cheminSon);

var enceinteFacade:SoundChannel;
enceinteFacade = chainehifi.play();

var reglagesAudio:SoundTransform = enceinteFacade.soundTransform;
reglagesAudio.volume = 0.8;
enceinteFacade.soundTransform = reglagesAudio;
```

Exemple 2

Trois occurrences de clip ont été placées sur la scène et nommées `btNiveau10`, `btNiveau50`, `btNiveau100`. Nous leur affectons à chacune un gestionnaire chargé de régler le niveau sonore à des valeurs différentes.

Fichier de référence : `Chapitre12/son4.fla`

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var chainehifi = new Sound(cheminSon);

var enceinteFacade:SoundChannel;
enceinteFacade = chainehifi.play();

var reglagesAudio:SoundTransform = enceinteFacade.soundTransform;


btNiveau100.addEventListener(MouseEvent.CLICK,niveauSon100);
btNiveau50.addEventListener(MouseEvent.CLICK,niveauSon50);
btNiveau10.addEventListener(MouseEvent.CLICK,niveauSon10);

function niveauSon100(evt:MouseEvent) {
  reglagesAudio.volume = 1;
  enceinteFacade.soundTransform = reglagesAudio;
}
```

```
function niveauSon50(evt:MouseEvent) {
  reglagesAudio.volume = 0.5;
  enceinteFacade.soundTransform = reglagesAudio;
}

function niveauSon10(evt:MouseEvent) {
  reglagesAudio.volume = 0.1;
  enceinteFacade.soundTransform = reglagesAudio;
}
```

Contrôler la balance d'un son

Avant de lire ce passage, consultez le développement précédent, Contrôler le niveau sonore.

Pour régler la balance d'un son, nous allons devoir faire appel à la propriété `pan`. Les valeurs qu'elle peut recevoir s'étendent de -1 à 1 car elles représentent le dosage du niveau sonore dans l'enceinte de gauche et de droite du dispositif audio de votre ordinateur. Vous devriez donc obtenir un script comme celui-ci :

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var chainehifi = new Sound(cheminSon);

var enceinteFacade:SoundChannel;
enceinteFacade = chainehifi.play();

var reglagesAudio:SoundTransform = enceinteFacade.soundTransform;
reglagesAudio.pan = -1;
enceinteFacade.soundTransform = reglagesAudio;
```

Affectez la valeur 0 pour rétablir l'équilibre du niveau sonore dans les deux enceintes de votre ordinateur.

Gérer la fin de la lecture d'un son

Il est utile de détecter le moment où la lecture d'un son se termine pour plusieurs raisons. Bien que cela dépende de la nature et de l'interactivité de votre animation, citons tout de même ces deux exemples :

- Dans le fichier `son3.fla`, nous avons été capable de contrôler l'apparence des boutons Lecture et Arrêt. Nous avons désactivé le clic sur le bouton Lecture afin qu'on ne puisse pas écouter plusieurs sons à la fois, mais lorsque la lecture du son se termine, il n'est plus possible de cliquer à nouveau sur le bouton Lecture.
- Lorsque nous devons enchaîner deux lectures de sons ou synchroniser la fin de la lecture d'un son avec une autre action, la détection de la fin de la lecture d'un son est essentielle.

Fichier de référence : `Chapitre12/son6.fla`

Dans cet exemple, nous allons utiliser un gestionnaire d'événement pour pouvoir exécuter des lignes d'instructions à la fin de la lecture d'un son. Notez que la technique est particulièrement simple.

```
var cheminSon:URLRequest = new URLRequest("Instru2.mp3");
var chainehifi = new Sound(cheminSon);
var pistePercussions:SoundChannel;
pistePercussions = chainehifi.play();

pistePercussions.addEventListener(Event.SOUND_COMPLETE,sonTermine);

function sonTermine(evt:Event) {
  cheminSon.url="Instru3.mp3";
  chainehifi = new Sound(cheminSon);
  pistePercussions = chainehifi.play();
}
```

Si nous comparons ce script avec les premiers que nous avons pu découvrir dans ce chapitre au sujet du son, nous constatons que nous avons simplement ajouté ces quelques lignes :

```
pistePercussions.addEventListener(Event.SOUND_COMPLETE,sonTermine);

function sonTermine(evt:Event) {
  instruction à réaliser lorsque le son est terminé;
}
```

La classe `Event` possède la constante `SOUND_COMPLETE` qui est invoquée tout naturellement à la fin de la lecture d'un son.

Dans le fichier `son3.fla`, évoqué en introduction à ce développement, il faudrait donc ajouter ce gestionnaire d'événement et rétablir les propriétés `mouseEnabled` et `alpha` que nous avions utilisées.

Réaliser une jauge de lecture

Fichier de référence : `Chapitre12/son7.fla`

Pour réaliser une jauge de progression de la lecture d'un son, nous allons tout simplement chercher à exécuter en boucle, grâce à l'événement `Event.ENTER_FRAME` d'un gestionnaire, une ligne d'instruction chargée de régler l'échelle horizontale d'une occurrence en fonction de la position de la tête de lecture par rapport à la durée totale du son.

Remarque

L'instance `jauge` est un rectangle plus large que haut que nous avons placé sur la scène.

```
jauge.addEventListener(Event.ENTER_FRAME,animerJaugeLecture);

function animerJaugeLecture(evt:Event) {
  jauge.scaleX = piste1.position/chainehifi.length;
}
```

Vous noterez que nous obtenons la position de la tête de lecture en lisant la propriété `position` de l'instance de la classe `SoundChannel()`, alors que la durée du son est renvoyée par la propriété `length` de l'instance de la classe `Sound()`. À l'exception de ce détail, le script est assez logique, mais il présente quelques erreurs dans son application.

L'initialisation de la jauge et la valeur NaN

Au lancement de l'animation `son7.fla`, vous pouvez constater que le rectangle qui représente la jauge de chargement est complètement rempli, et que cette jauge bascule complètement à gauche.

Lorsque l'animation est lancée, la ligne d'instruction de la fonction de rappel `animerJauge-Lecture()` n'a pas encore été exécutée car le son n'est pas encore chargé. La jauge ne peut donc être réglée à 0 dès le lancement de l'animation. Il faut par conséquent régler ce paramètre en utilisant la propriété `scaleX`. Par ailleurs, le résultat du calcul du rapport `piste1.position/chainehifi.length` donne la valeur `NaN` (ce qui signifie `Not a Number`), ce qui est dû à une division par zéro, la propriété `length` ayant une valeur nulle. Toujours est-il que la jauge est réglée à -1 en `scaleX`, ce qui se traduit par le sursaut de la jauge vers la gauche. Pour éviter ces deux problèmes, nous devons donc écrire :

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var chainehifi = new Sound(cheminSon);
var piste1:SoundChannel;

jauge.scaleX=0;

piste1 = chainehifi.play();
jauge.addEventListener(Event.ENTER_FRAME,animerJaugeLecture);

function animerJaugeLecture(evt:Event) {

  if (piste1.position/chainehifi.length>=0) {
    jauge.scaleX = piste1.position/chainehifi.length;
  }
}
```

Remarque

L'utilisation du `if()` n'est utile que dans cet exemple. Dans le suivant, la lecture du son ne va pas être automatique mais obtenue à partir d'un clic ; ce test sera donc inutile.

Afin que vous puissiez insérer facilement ce code dans un de vos programmes, nous vous avons préparé une animation plus globale sur la gestion d'une jauge, mais elle ne gère pas correctement les boutons Lecture et Arrêt, ce qui aurait produit un script trop long. Consultez l'animation `son3.fla` pour compléter le script ci-dessous.

Fichier de référence : `Chapitre12/son8.fla`

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var chainehifi = new Sound(cheminSon);
var piste1:SoundChannel;
```

```
btLecture.addEventListener(MouseEvent.MOUSE_DOWN,lireSon);
btArret.addEventListener(MouseEvent.MOUSE_DOWN,arreterSon);

function lireSon(evt:MouseEvent) {
  piste1 = chainehifi.play();
  jauge.addEventListener(Event.ENTER_FRAME,animerJaugeLecture);

}

function arreterSon(evt:MouseEvent) {
  piste1.stop();
  jauge.removeEventListener(Event.ENTER_FRAME,animerJaugeLecture);
}


function animerJaugeLecture(evt:Event) {
  jauge.scaleX = piste1.position/chainehifi.length;
}
```

Vous remarquerez que nous supprimons l'écouteur qui gère l'événement `Event.ENTER_FRAME` lorsque nous cliquons sur le bouton Arrêt : il est en effet inutile de demander à Flash de chercher à mettre à jour l'affichage de l'échelle de l'occurrence `jauge` alors qu'il n'y a pas de changement.

Par ailleurs, n'oubliez pas non plus de prévoir dans la fonction de rappel du gestionnaire d'événement `Event.SOUND_COMPLETE`, la suppression de l'écouteur qui gère l'événement `Event.ENTER_FRAME`.

Effectuer une pause

Aussi curieux que cela puisse vous paraître, la méthode `pause()` n'existe pas dans la classe `SoundChannel()` ni même dans la classe `Sound()`. Nous allons donc devoir la programmer. Rassurez-vous, la technique est extrêmement basique et logique.

Fichier de référence : `Chapitre12/son9.fla`

Pour cela, il faut utiliser une variable pour mémoriser la position de la tête de lecture au moment où la lecture du son est interrompue. Lorsque nous reprendrons la lecture du son, nous utiliserons alors la méthode `play()` de la classe `Sound()` en précisant en paramètre la valeur contenue dans la variable.

```
var positionTete:Number=0;

var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var enceinte = new Sound(cheminSon);
var piste1:SoundChannel;

btLecture.addEventListener(MouseEvent.MOUSE_DOWN,lireSon);

function lireSon(evt:MouseEvent) {
  if (btLecture.etiquette.text =="Lecture") {
    btLecture.etiquette.text ="Pause";
    piste1 = enceinte.play(positionTete);
```

```
    } else {
      btLecture.etiquette.text ="Lecture";
      positionTete=piste1.position;
      piste1.stop();
    }
  }
```

Dans cet exemple, nous changeons le contenu d'un texte qui se trouve dans le bouton sur lequel nous cliquons, mais nous aurions également pu contrôler la tête de lecture d'un clip qui possède deux images-clés.

Par ailleurs, n'oubliez pas de prévoir dans la fonction de rappel de votre gestionnaire d'événement `Event.SOUND_COMPLETE` la réinitialisation du texte figurant sur le bouton.

Gérer la fin du chargement d'un son

Il est parfois nécessaire de connaître l'instant où la fin du chargement du son se produit ; vous utiliserez alors l'événement `Event.COMPLETE`. La technique est extrêmement simple, car il s'agit d'un gestionnaire classique que vous définissez sur l'instance de la classe `Sound()`.

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var chainehifi = new Sound(cheminSon);
var piste1:SoundChannel;
piste1 = chainehifi.play();

chainehifi.addEventListener(Event.COMPLETE,chargementTermine);

function chargementTermine(evt:Event) {
  messageFin.text="Chargement terminé";
}
```

Dans cet exemple, nous avons demandé la lecture du son avant même qu'il soit complètement chargé ; c'est tout à fait normal : nous employons le téléchargement progressif qui caractérise la technologie Flash. Le risque que la lecture du son soit interrompue à cause d'un débit trop bas est assez faible : les débits proposés aujourd'hui par les différents fournisseurs d'accès à Internet sont assez élevés. Cependant, si vous souhaitiez ne prendre aucun risque, vous pourriez alors adapter ce script de la façon suivante :

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var chainehifi = new Sound(cheminSon);
var piste1:SoundChannel;

chainehifi.addEventListener(Event.COMPLETE,chargementTermine);

function chargementTermine(evt:Event) {
  piste1 = chainehifi.play();
}
```

La demande de lecture du son est effectuée dans la fonction de rappel qui est appelée uniquement à la fin du chargement du son !

Réaliser une jauge de chargement

Pour conclure nos explications sur le son dans une animation Flash, découvrons deux dernières propriétés qui vous nous permettre de réaliser une jauge de préchargement. Nous allons simplement créer un écouteur qui va gérer l'événement `Event.ENTER_FRAME`.

```
var cheminSon:URLRequest = new URLRequest("Instru4.mp3");
var chainehifi = new Sound(cheminSon);
var piste1:SoundChannel;

jauge.addEventListener(Event.ENTER_FRAME,animerJaugeLecture);
chainehifi.addEventListener(Event.COMPLETE,chargementTermine);

function animerJaugeLecture(evt:Event) {
  jauge.scaleX = chainehifi.bytesLoaded/chainehifi.bytesTotal;
}

function chargementTermine(evt:Event) {
  piste1 = chainehifi.play();
}
```

Vous noterez que nous avons également ajouté le gestionnaire d'événement `Event.COMPLETE` dans cet exemple, car nous demandons la lecture de notre son uniquement à la fin de l'animation du préchargement.

Charger et contrôler une vidéo

Comme nous l'évoquions déjà dans ce chapitre, ce livre est dédié à l'ActionScript. Nous allons donc créer l'instance du composant `FLVPlayback` à partir de lignes d'instructions et non pas au travers de l'interface. Si vous préférez effectuer un glisser-déposer du composant FLVPlayback de la bibliothèque vers la scène, ne lisez pas le développement suivant, Créer une occurrence de type `FLVPlayback`, et passez directement à la suite, Configurer une occurrence de type `FLVPlayback`.

Créer une occurrence de type FLVPlayback

Fichier de référence : `Chapitre12/video1.fla`

Si vous suivez nos développements depuis les premières pages, vous aurez pu remarquer que nous essayons de vous pousser à coder tout ce qui peut l'être, vous évitant ainsi d'utiliser l'interface. Pour la vidéo, nous allons procéder de même, sans tomber dans l'extrême. Sur Internet, à travers de nombreux cours, on observe depuis plusieurs mois une utilisation intensive des classes `NetConnection()` et `NetStream()` pour gérer la vidéo. C'est à croire que la communauté des codeurs AS3 a oublié l'existence du composant `FLVPlayBack` qui est très bien conçu et surtout très pratique. Voici comment l'utiliser.

> **Remarque**
>
> Avant de suivre ces explications, assurez-vous d'avoir enregistré votre animation.

1. Commencez par importer manuellement un composant de type `FLVPlayback` dans la bibliothèque de votre animation.
2. Placez une vidéo au format FLV dans le dossier où se trouve votre animation.
3. Saisissez les lignes d'instructions ci-dessous :

```
import fl.video.FLVPlayback;

var ecranVideo:FLVPlayback;

ecranVideo = new FLVPlayback();
ecranVideo.source = "Luna.flv";
addChild(ecranVideo);
```

Il s'agit du minimum de code à saisir pour obtenir une vidéo sur la scène. Nous commençons par importer la classe `FLVPlayback()` et créer une instance du même type avant de l'ajouter à la liste d'affichage. Nous définissons également un chemin vers une vidéo.

À ce niveau du développement, nous ne disposons pas d'un contrôleur de vidéo. Pour cela, il faut ajouter une ligne de code qui va établir un lien entre le fichier SWF de l'animation et celui de l'enveloppe (*skin*) du contrôleur.

Figure 12-1

Pour contrôler une vidéo, vous devez utiliser un contrôleur plus connu sous le nom de skin en ActionScript.

Il existe plusieurs sortes d'enveloppes de contrôleur. Il s'agit de fichiers au format SWF qui se trouvent dans le dossier `Adobe Flash CS3/ Configuration/FLVPlayback Skins/ActionScript 3.0`.

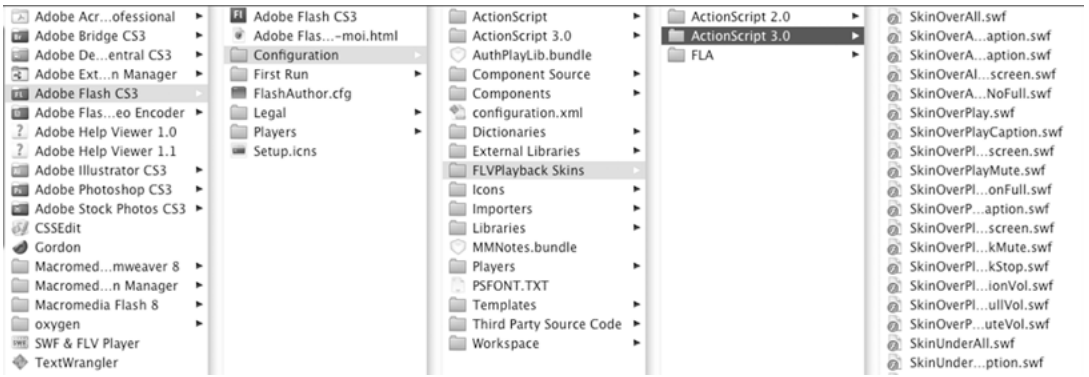


Figure 12-2

Il existe de nombreuses enveloppes, car vous pouvez choisir entre plusieurs configurations.

Sélectionnez l'un des fichiers SWF dans ce dossier `ActionScript 3.0` et copiez-le dans le dossier où se trouve votre animation Flash et la vidéo que vous utilisez. Dans notre cas, nous avons pris le fichier qui s'intitule `SkinOverAllNoFullNoCaption.swf` que nous rebaptisons `controleur.swf`.

> **Remarque**
>
> Les fichiers dont le nom commence par `SkinUnder` proposent des contrôleurs qui se placent sous la vidéo.

Ajoutez la ligne d'instruction ci-dessous pour afficher un contrôleur sur la vidéo.

```
ecranVideo.skin = "controleur.swf";
```

Terminez le script en définissant une largeur et une hauteur d'image :

```
ecranVideo.width = 384;
ecranVideo.height = 216;
```

Le script complet devrait maintenant ressembler à celui-ci :

```
import fl.video.FLVPlayback;

var ecranVideo:FLVPlayback;

ecranVideo = new FLVPlayback();
ecranVideo.source = "Luna.flv";
addChild(ecranVideo);

ecranVideo.skin = "controleur.swf";

ecranVideo.width = 384;
ecranVideo.height = 216;
```

Voyons à présent les techniques de configuration d'une occurrence de type `FLVPlayback`.

Configurer une occurrence de type FLVPlayback

Parmi les différents réglages que vous pouvez effectuer sur une occurrence de ce type, voici les plus importants.

Fichier de référence : `Chapitre12/video2.fla`

Les réglages relatifs à l'enveloppe (skin)

Si vous disposez d'une vidéo très colorée avec une dominante importante ou d'une vidéo en noir et blanc, vous souhaiterez peut-être choisir la couleur de l'enveloppe afin de garder une certaine harmonie des couleurs dans votre interface. Il suffit alors d'ajouter une ligne d'instruction à celles que nous venons de voir dans le développement précédent. Si vous avez placé une vidéo sur la scène par un glisser-déposer, cette ligne d'instruction constituera la première ligne de code de votre script.

```
ecranVideo.skin = "controleur.swf";
```

Vous pouvez également ajouter ces trois autres lignes qui sont, elles aussi, facultatives.

```
ecranVideo.skinBackgroundAlpha=0.6;
ecranVideo.skinAutoHide = true;
ecranVideo.skinFadeTime = 2000;
```

La première ligne permet de rendre l'enveloppe plus ou moins transparente ; vous devez utiliser une valeur comprise entre 0 et 1.

La deuxième ligne permet de masquer automatiquement le contrôleur lorsque le curseur de la souris ne se trouve pas sur la vidéo. Cette ligne d'instruction est particulièrement importante, voire indispensable lorsque vous optez pour une enveloppe de type SkinOver…

La troisième ligne sert à régler la vitesse de fondu entrant et sortant pour afficher ou masquer le contrôleur. Vous noterez que la valeur est exprimée en millisecondes.

L'initialisation du mode de lecture

Lorsque vous diffusez une vidéo dans une animation, il est important de définir le mode de lecture. Soit vous souhaitez que l'utilisateur clique sur le bouton lecture du contrôleur, soit, au contraire, vous préférez que la lecture démarre automatiquement. Le réglage de ce paramètre passe par la propriété `autoPlay` qui prend pour valeur `true` ou `false`.

```
ecranVideo.autoPlay = false;
```

De la même façon, c'est à vous de définir le retour automatique de la tête de lecture à la fin de la lecture d'une vidéo.

```
ecranVideo.autoRewind = false;
```

La vidéo plein écran

Depuis Flash CS3, il est possible d'afficher une vidéo en plein écran, que ce soit dans un navigateur ou à partir d'un projecteur.

Pour cela, vous devez effectuer un certain nombre de manipulations.

1. Commencez par choisir une enveloppe qui possède le bouton plein écran situé à droite du contrôleur.

Figure 12-3

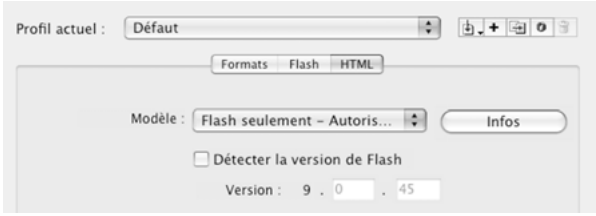
Le bouton plein écran ne se trouve pas sur toutes les enveloppes. Vous devez en choisir une qui le propose.

2. Publiez votre animation via la commande Paramètres de publication… du menu Fichier.

Si vous générez une page HTML, faites très attention de régler l'option Modèle de l'onglet HTML sur Flash seulement - Autorisation du plein écran.

Figure 12-4

Vous ne pourrez pas générer le fichier AC_RunActiveContent.js si vous ne réglez pas le modèle sur Flash seulement - Autorisation plein écran ; vous n'obtiendrez donc pas un affichage en plein écran.



Nous vous conseillons également de cocher la case Détecter la version de Flash en conservant la version 9.0.45 par défaut, pour inviter l'utilisateur à charger la bonne version du lecteur au cas où il en posséderait une plus ancienne.

Si vous réalisez un projecteur, il est important de savoir qu'il comporte une erreur qui empêche le vrai mode plein écran de la vidéo, si votre animation est déjà affichée en plein écran.

Lorsque vous lancez un projecteur ou une page HTML, il ne reste plus qu'à cliquer sur le bouton plein écran pour l'afficher dans ce mode.

> **Attention**
>
> Flash génère une fichier qui s'intitule `AC_RunActiveContent.js` lorsque vous publiez une animation avec une vidéo plein écran. N'oubliez pas de la placer toujours avec l'enveloppe.

Consultez le développement suivant pour découvrir qu'il est possible de basculer en mode plein écran sans disposer d'enveloppe.

Contrôler une vidéo

Fichier de référence : `Chapitre12/video3.fla`

Nous pourrions être tenté de dire qu'il n'est pas nécessaire d'aborder les propriétés et méthodes de la classe `FLVPlayback()` servant à contrôler une vidéo, dans la mesure où

l'enveloppe nous offre toutes les fonctionnalités nécessaires. Cependant, il existe des situations où vous ne disposerez pas d'une enveloppe, notamment dans le cas où vous souhaitez personnaliser le contrôleur.

Les propriétés et méthodes de lecture

Tableau 12-1 Méthodes et propriétés servant à contrôler la lecture d'une vidéo

Méthode/Propriété	Descriptif	Remarque
`seek()`	Méthode qui permet de déplacer la tête de lecture à une position précise.	Valeur exprimée en secondes.
`play()`	Méthode qui permet de lancer la lecture d'une vidéo.	
`pause()`	Méthode qui permet de mettre une vidéo en pause.	
`stop()`	Méthode qui permet d'arrêter la lecture d'une vidéo.	
`volume`	Propriété qui permet de régler le volume.	Valeur comprise initialement entre 0 et 1 (par exemple 0.5 pour 50 % du niveau par défaut). Vous pouvez dépasser cette valeur jusqu'à la limite du potentiel de vos enceintes.
`fullScreenButton`	Propriété qui définit le nom d'occurrence servant à basculer la projection d'une vidéo en plein écran.	Attention : vous devez placer sur la scène, un composant de type Bouton Plein écran (figure 12-5). Personnalisez ensuite les clips imbriqués qu'il contient pour changer son apparence.
`fullScreenBackgroundColor`	Propriété qui règle la couleur du fond d'écran lorsqu'une vidéo est en plein écran et qu'elle ne recouvre pas l'intégralité de la surface de l'écran.	Valeur en hexadécimal.

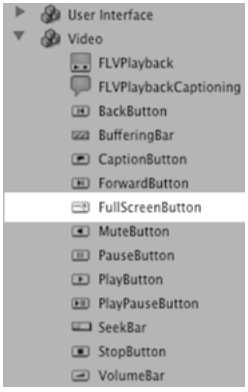


Figure 12-5
Utilisez le composant vidéo Bouton Plein écran pour pouvoir utiliser la propriété fullScreenButton.

Voici le script de l'animation `video3.flv` qui vous propose un contrôle de la vidéo sans passer par un contrôleur.

```
import fl.video.FLVPlayback;

var ecranVideo:FLVPlayback;

ecranVideo = new FLVPlayback();
ecranVideo.source = "Luna.flv";
addChild(ecranVideo);

ecranVideo.width = 384;
ecranVideo.height = 216;

ecranVideo.x=(500-384)/2; // Nous centrons la vidéo horizontalement
ecranVideo.y=(300-216)/2; // Nous centrons la vidéo verticalement
//
btLecture.addEventListener(MouseEvent.MOUSE_DOWN,lecture);
btPause.addEventListener(MouseEvent.MOUSE_DOWN,enPause);
btArret.addEventListener(MouseEvent.MOUSE_DOWN,arret);

btVolumePlus.addEventListener(MouseEvent.MOUSE_DOWN,monterVolume);
btVolumeMoins.addEventListener(MouseEvent.MOUSE_DOWN,baisserVolume);

btMoinsDix.addEventListener(MouseEvent.MOUSE_DOWN,deplacerTeteAvant);
btPlusDix.addEventListener(MouseEvent.MOUSE_DOWN,deplacerTeteApres);

function  lecture(evt:MouseEvent) {
  ecranVideo.play();
}
function  enPause(evt:MouseEvent) {
  ecranVideo.pause();
}
function  arret(evt:MouseEvent) {
  ecranVideo.stop();
}


function  monterVolume(evt:MouseEvent) {
  ecranVideo.volume += 0.2;
}
function  baisserVolume(evt:MouseEvent) {
  ecranVideo.volume -= 0.2;
}

function  deplacerTeteAvant(evt:MouseEvent) {
  ecranVideo.seek(ecranVideo.playheadTime-2);
}
function  deplacerTeteApres(evt:MouseEvent) {
  ecranVideo.seek(ecranVideo.playheadTime+2);
}

ecranVideo.fullScreenButton = btPleinEcran;
ecranVideo.fullScreenBackgroundColor = 0x333333;
```

Les propriétés et méthodes pour une jauge de lecture

Fichier de référence : `Chapitre12/video4.fla`

> **Remarque**
>
> Dans le fichier `video4.fla`, nous n'avons pas placé l'instance de la classe `FLVPlayback()` à partir de la ligne d'instructions pour que toute votre attention se porte sur le code qui gère le défilement de la jauge de lecture.

Les enveloppes proposées dans l'interface de Flash présentent une jauge de lecture, mais, elle n'affiche pas pour autant la durée de la vidéo, ni même le temps à la position de la tête de lecture. Nous allons donc réécrire une jauge de lecture accompagnée des deux valeurs que nous venons de mentionner.

Nous devons commencer par créer sur la scène, manuellement ou dynamiquement, deux textes dynamiques. Nous les nommons `dureeTotale` et `positionTete`. Nous plaçons également un rectangle assez large représentant la jauge de lecture que nous intitulons `jauge`. Nous devons à présent définir des gestionnaires d'événements pour exécuter des lignes d'instructions au chargement de la vidéo, durant la lecture et à la fin de la vidéo.

```
ecran.addEventListener(MetadataEvent.METADATA_RECEIVED,repereAtteint);
…
ecran.addEventListener(VideoEvent.PLAYHEAD_UPDATE, lectureEnCours);
…
ecran.addEventListener(VideoEvent.COMPLETE,videoTerminee);
```

Pour obtenir le temps à la position de la tête de lecture, nous allons utiliser la propriété `playheadTime` de la classe `FLVPlayback()`. Pour déterminer la durée d'une vidéo, nous devons lire la propriété `duration` contenue dans l'objet de la fonction de rappel invoquée par l'écouteur qui gère l'événement `MetatdataEvent.METADATA_RECEIVED`. Si vous ne comprenez pas bien cette dernière phrase, aidez-vous du script ci-dessous pour localiser la propriété `duration`.

Vous devriez obtenir un script global comme celui-ci :

```
import fl.video.*;
//
//Déclaration des variables
//
var dureeVideo:Number;
//
//#### Au début de la lecture
//
ecranVideo.addEventListener(MetadataEvent.METADATA_RECEIVED,repereAtteint);
function repereAtteint(evt:MetadataEvent) {
  dureeVideo = evt.info.duration;
  dureeTotale.text = convertirTemps(evt.info.duration);
}
//
// #### En cours de lecture #####
```

```
//
ecranVideo.addEventListener(VideoEvent.PLAYHEAD_UPDATE, lectureEnCours);
function lectureEnCours(evt:VideoEvent) {
  positionTete.text=convertirTemps(ecranVideo.playheadTime).toString();
  jauge.scaleX=ecranVideo.playheadTime/dureeVideo;
}
//
// #### Lorsque la vidéo est terminée #####
//
ecranVideo.addEventListener(VideoEvent.COMPLETE,videoTerminee);
function videoTerminee(evt:VideoEvent) {
  //legende.text="Fin";
}


function convertirTemps(tempsEcoule):String {
  var minutes=Math.floor(tempsEcoule  / 60);
  var secondes=Math.floor(tempsEcoule  % 60);
  minutes=minutes <= 9?"0" + minutes:minutes;
  secondes=secondes <= 9?"0" + secondes:secondes;
  var tempsFinal=minutes + ":" + secondes;
  return tempsFinal;
}
```

Cette dernière fonction ne sert qu'à convertir l'affichage des temps renvoyés par les propriétés `playheadTime` et `duration`.

Gérer les repères de temps (cuePoints)

Fichier de référence : `Chapitre12/video5.fla`

Dès qu'il vous faudra synchroniser des instants précis dans une vidéo avec des actions dans une animation, vous devrez gérer des repères de temps, appelés aussi *cuePoints*. Découvrons donc à présent la technique d'ajout et de détection des repères de temps.

Commencez par créer un texte dynamique sur la scène que vous nommez `legende`.

Ajout des repères

Nous devons maintenant ajouter des points de repères dans la vidéo avant de pouvoir les détecter à la lecture. Pour cela, vous devez utiliser la méthode `addASCuePoint()` et préciser deux paramètres. Le premier indique l'instant *t* auquel le repère doit être placé. Le deuxième contient éventuellement un texte qui peut être affiché sur la scène, ce que nous allons faire dans cet exemple.

> **Astuce**
>
> Vous pouvez stocker un numéro en guise de nom de repère afin de l'utiliser comme numéro d'index dans un tableau ou une arborescence XML.

```
ecranVideo.addASCuePoint(2.5,"Oh !");
ecranVideo.addASCuePoint(5,"Regarde combien j'en ai !");
ecranVideo.addASCuePoint(10,"C'est bien beaucoup !");
```

Il serait plus judicieux d'effectuer une boucle `for()` et de parcourir un tableau ou un fichier XML pour définir les différents repères à placer dans une vidéo.

Détecter des repères

Nous allons faire appel à un écouteur qui gère l'événement `MetadataEvent.CUE_POINT`, et nous allons chercher à lire la propriété `info.name` contenue dans l'objet renvoyé à la fonction de rappel `repereAtteint()` de notre exemple.

```
ecranVideo.addEventListener(MetadataEvent.CUE_POINT,repereAtteint);
function repereAtteint(evt:MetadataEvent) {
  legende.text = evt.info.name;
}
```

Si le nom du repère est un nombre, n'oubliez pas de l'utiliser en tant qu'index pour parcourir un tableau ou une arborescence XML. Retrouvez un exemple employant cette technique après le script ci-dessous.

Le fichier `video5.fla` contient le script global suivant :

```
import fl.video.*;

// Ajouter des repères
ecranVideo.addASCuePoint(2.5,"Oh !");
ecranVideo.addASCuePoint(5,"Regarde combien j'en ai !");
ecranVideo.addASCuePoint(10,"C'est bien beaucoup !");

//Détecter des repères
ecranVideo.addEventListener(MetadataEvent.CUE_POINT,repereAtteint);
function repereAtteint(evt:MetadataEvent) {
  legende.text = evt.info.name;
}
```

Dans cet exemple, nous ne gérons pas l'effacement des textes après un certain laps de temps entre deux repères. De ce fait, certains sous-titres ne devraient plus être affichés après plusieurs secondes. Il faudrait soit créer des repères avec des noms de repère vides ou bien faire appel à la classe `Timer()` pour effacer le texte contenu dans l'instance `legende`.

Fichier de référence : `Chapitre12/video6.fla`

```
import fl.video.*;

var repliques:Array = ["Oh !","Regarde combien j'en ai !","C'est bien beaucoup !"];

// Ajouter des repères
ecranVideo.addASCuePoint(2.5,"0");
ecranVideo.addASCuePoint(5,"1");
ecranVideo.addASCuePoint(10,"2");
```

```
//Détecter des repères
ecranVideo.addEventListener(MetadataEvent.CUE_POINT,repereAtteint);
function repereAtteint(evt:MetadataEvent) {
  legende.text = repliques[evt.info.name];
}
```

Encoder une vidéo au format FLV

Pour encoder une vidéo au format FLV, vous pouvez utiliser de nombreuses solutions logicielles, mais Flash CS3 est livré avec Flash CS3 Video Encoder.

Figure 12-6

L'application Flash Video Encoder permet de générer des fichiers au format FLV (Flash Vidéo).

Il s'agit d'un logiciel à part que nous vous conseillons d'utiliser si vous ne pouvez pas acheter un logiciel dédié tel que Sorenson Squeeze ou On2 Flix.

Son fonctionnement est assez simple car les préréglages proposés sont plutôt en phase avec les contraintes du marché.

1. Lancez le logiciel qui se trouve dans le dossier `Adobe Flash CS3 Video Encoder`, lui-même dans le dossier d'installation de Flash.
2. Faites glisser une ou plusieurs vidéos au centre de la fenêtre principale.
3. Sélectionnez une ou plusieurs lignes qui représentent les vidéos que vous vous apprêtez à encoder et cliquez sur le bouton Paramètres.

Figure 12-7

Un simple glisser-déposer suffit à ajouter un fichier à encoder dans le logiciel Flash CS3 Video Encoder.

4. Vous pouvez à présent utiliser les réglages automatiques ou procéder à des réglages manuels à travers les cinq onglets présents dans l'interface. Si vous n'avez pas de connaissances particulières dans le domaine de la vidéo, nous vous conseillons de sélectionner un profil et de ne pas toucher les paramètres vidéo et audio proposés dans les onglets éponymes.
5. Précisez un nom de fichier d'export. Dans le cas contraire, le document créé portera le même préfixe que votre vidéo qui n'est pas encore encodée.
6. Validez cet écran en cliquant sur le bouton OK. Vous retrouvez alors la fenêtre de départ, à partir de laquelle vous devez lancer la file d'attente.

L'onglet Points de repère ne vous sert à rien dans la mesure où vous savez maintenant ajouter dynamiquement des repères. En revanche, redimensionnez ou recadrez éventuellement votre vidéo à partir du dernier onglet de la barre.

Figure 12-8

Vous pouvez accéder aux différents panneaux de configuration pour l'encodage d'une vidéo en cliquant sur les différents onglets situés au milieu de la fenêtre.

Même si Flash est capable d'encoder une vidéo lorsque vous l'importez dans l'interface, nous vous déconseillons cette pratique. Vous bloqueriez alors l'IDE de Flash et vous ne pourriez donc plus agir sur l'interface.

Vous êtes à présent capable de gérer une vidéo dans une animation, non plus comme un simple média, pour lequel vous proposez une lecture, mais comme un média interactif.

Remarque

Un de mes ouvrages, La vidéo dans Flash (Éditions Eyrolles) utilise l'ActionScript 2.0. De ce fait, seule la moitié du livre est utile, notamment sur les bases de la vidéo et son encodage. De nombreuses animations vous sont proposées en AS2, mais elles constituent néanmoins des solutions utiles aux besoins de gestion de vidéo interactive. Seule la syntaxe des scripts n'est plus d'actualité.

Charger et contrôler des données (texte et PHP)

Lorsque vous devrez placer des données de type texte sur la scène de vos animations, trois solutions s'offriront à vous :

- Créer et charger un fichier XML.
- Créer et charger un fichier texte.
- Créer des bases de données et des pages dynamiques (par exemple de type PHP) dans le but de leur envoyer des requêtes.

Le chapitre 13 traite de la création et de la manipulation d'un document XML. Nous ne développerons donc aucun propos supplémentaire dans ce chapitre à ce sujet.

L'utilisation d'un fichier texte est analogue à l'appel à une base de données. Essayons donc de comprendre le mécanisme de ce type de chargement.

Quel que soit le fichier ou la source que vous chargerez dans une animation, vous devez toujours garder à l'esprit la structure et l'organisation des données auxquelles s'attend Flash.

Structure des données

En programmation, les variables servent à contenir des données. Nous allons donc utiliser la même syntaxe pour créer un fichier texte ou concevoir une page Web dynamique. Pour séparer plusieurs variables, nous utiliserons l'esperluette (&, encore appelé « et commercial »).

Pour ce premier exemple écrivez la ligne ci-dessous dans un fichier texte.

```
prenom=David&nom=TARDIVEAU&age=37&profession=Enseignant
```

Enregistrez ensuite ce document au format UTF-8 afin d'encoder correctement les données qu'il contient (nommez-le, par exemple, `donnees.txt`).

Les variables de type texte

Comme vous l'aurez peut-être remarqué, les données de type texte des variables contenues dans un tel fichier ne sont pas saisies entre guillemets.

Celles et ceux qui connaissent le PHP auront compris que nous cherchons à générer l'affichage de données sous ce format-là. Si vous avez déjà travaillé avec des bases MySQL, toutes ces explications vous sembleront basiques.

Remarque

Si vous ne connaissez pas le PHP, n'essayez pas de charger des pages PHP qui se trouveraient sur votre ordinateur, dans le même dossier que votre animation Flash. Il est nécessaire de les déposer sur un serveur afin qu'elles puissent être interprétées.

Les deux fichiers PHP que nous allons vous présenter dans quelques instants ne font pas appel à une base de données, mais ils simulent le résultat de l'interrogation d'une base. Si vous êtes néophyte en langage PHP, placez simplement ces deux documents sur le serveur qui héberge votre site pour effectuer les essais liés aux exemples abordés dans ce chapitre. Vous pouvez aussi utiliser l'adresse à laquelle nous faisons référence.

Fichier de référence : `Chapitre12/requete.php`

```
<?php
echo "prenom=David&nom=TARDIVEAU&adressemail=david@yazo.net";
?>
```

Comme vous pouvez le constater, que nous interrogions une base de données ou que nous exécutions de simples commandes `echo`, le résultat sera le même.

Fichier de référence : `Chapitre12/requete2.php`

```
<?php
switch ($prenom) {
case "david":
    echo "prenom=David&nom=TARDIVEAU&adressemail=david@yazo.net";
    break;
case "olivier":
    echo "prenom=Olivier&nom=MARTIN&adressemail=olivier@gmail.com";
    break;
case "fabienne":
    echo "prenom=Fabienne&nom=Gervasoni&adressemail=fabienne@gmail.com";
    break;
}
?>
```

Nous allons devoir envoyer des données à une page dynamique. Pour cela, nous utiliserons le système de branchement de conditions pour éviter de passer par une base de données MySQL.

Remarque

Pour vous assurer du bon fonctionnement de la page PHP que vous avez conçue, placez-la sur un serveur et interrogez-la en saisissant l'URL où elle se trouve. Pour passer les variables dans l'URL, utilisez le point d'interrogation conventionnel.

```
http://www.yazo.net/racine/Tests/requete.php
http://www.yazo.net/racine/Tests/requete2.php?prenom=Olivier
http://www.yazo.net/racine/Tests/requete2.php?prenom=David
http://www.yazo.net/racine/Tests/requete2.php?prenom=Fabienne
```

Établir une connexion avec une page dynamique ou de type texte

Vous allez constater que la procédure pour établir une connexion n'est pas plus compliquée que celle pour charger des images sur la scène d'une animation.

1. Instanciez la classe `URLLoader()` qui va contenir les données que nous allons charger.

```
var chargeurDonnees:URLLoader = new URLLoader();
```

2. Instanciez la classe `URLRequest()` en précisant entre parenthèses l'adresse du fichier à charger.

```
URLRequest("http://www.yazo.net/racine/Tests/requete.php");
```

> **Remarque**
> Si vous ne disposez pas d'un fichier texte ou d'une page PHP sur un serveur que vous pourriez charger, utilisez l'adresse que nous mentionnons.

3. Chargez les données contenues dans le fichier que vous avez indiqué dans l'instance de la classe `URLLoader()`.

```
chargeurDonnees.load(adresseDonnees);
```

Ces trois lignes vont maintenant nous permettre d'avoir accès aux variables contenues dans le fichier `requete.php`. Avant d'essayer de les lire, nous devons nous assurer qu'elles ont bien été chargées, c'est pourquoi nous devons faire appel à un gestionnaire d'événement.

Vérifier la fin d'un chargement

Si vous ne souhaitez pas effectuer cette vérification, vous prenez le risque d'essayer de lire des données qui ne se trouvent pas encore dans la mémoire vive de votre ordinateur. L'événement `Event.COMPLETE` auquel nous allons faire appel nous permet d'informer le programme de la fin du chargement des données. C'est dans la fonction de rappel, associée à l'écouteur que nous allons créer, que nous devons placer les lignes d'instructions chargées de lire les variables contenues dans le fichier `requete.php`.

```
chargeurDonnees.addEventListener(Event.COMPLETE, donneesChargees);

function donneesChargees(event:Event):void {
  trace(chargeurDonnees.data);
}
```

La fonction `addEventListener()` est à associer avec l'instance de la classe `URLLoader()` et non `URLRequest()`. Cette dernière contient bien l'adresse du fichier à charger, mais c'est dans l'instance `chargeurDonnees` que nous chargeons les variables.

Dans cet exemple, nous affichons l'ensemble des données contenues dans le fichier `requete.php`, en faisant référence à la propriété `data` de l'instance `chargeurDonnees`, mais nous ne pouvons pas encore lire individuellement chaque variable. Nous allons apprendre à le faire dans le prochain développement.

Vous devriez obtenir un script global tel que celui-ci.

Fichier de référence : `Chapitre12/urlLoader1.fla`

```
var chargeurDonnees:URLLoader = new URLLoader();
var adresseDonnees:URLRequest = new URLRequest("http://www.yazo.net/racine/
➥Tests/requete.php");

chargeurDonnees.load(adresseDonnees);

chargeurDonnees.addEventListener(Event.COMPLETE, donneesChargees);

function donneesChargees(event:Event):void {
  trace(chargeurDonnees.data);
}
prenom=David&nom=TARDIVEAU&adressemail=david@yazo.net
```

Manipuler les variables contenues dans une instance de type URLLoader()

Fichier de référence : `Chapitre12/urlLoader2.fla`

Lorsque vous chargerez des données contenues dans un fichier sous la forme `nomVariable=12&nomVariable2=13&nomVariable3=14` vous aurez alors besoin de pouvoir lire la valeur d'une variable indépendamment des autres. Nous devons donc placer le chargement contenu dans l'instance de la classe `URLLoader()` dans une instance de la classe `URLVariables()`.

```
var serieVariables:URLVariables = new URLVariables(chargeurDonnees.data);
```

Comme nous le précisions dans le développement précédent, vous ne devez pas essayer d'accéder à des données qui ne sont pas encore chargées. Vous devez donc placer cette ligne d'instruction dans la fonction de rappel d'un écouteur gérant l'événement `Event.COMPLETE`.

```
function donneesChargees(event:Event):void {
  var serieVariables:URLVariables = new URLVariables(chargeurDonnees.data);
}
```

En reprenant l'exemple précédent du chargement des données dans une instance de la classe `URLLoader()`, vous devriez obtenir le script global suivant :

```
var chargeurDonnees:URLLoader = new URLLoader();
var adresseDonnees:URLRequest = new URLRequest("http://www.yazo.net/racine/Tests/
➥requete.php");
```

```
chargeurDonnees.load(adresseDonnees);

chargeurDonnees.addEventListener(Event.COMPLETE, donneesChargees);

function donneesChargees(event:Event):void {
  var serieVariables:URLVariables = new URLVariables(chargeurDonnees.data);
  trace(serieVariables.prenom);
  trace(serieVariables.nom);
  trace(serieVariables.adressemail);
}
David
TARDIVEAU
david@yazo.net
```

Envoyer des variables à une URL

Pour pouvoir interroger une base de données, vous devrez généralement spécifier la valeur de certaines variables à passer à l'URL que vous utilisez pour charger des données. La classe `URLVariables()` que nous allons utiliser sert aussi bien à contenir des variables à charger qu'à envoyer. Dans le cas où vous devez lire l'ensemble des variables reçues lors d'un chargement, nous avons vu, dans le développement précédent, qu'il faut utiliser la propriété `data`. En revanche, pour enregistrer des variables, il suffit simplement de les mentionner en tant que propriété de l'instance de type `URLVariables`.

> **Remarque**
>
> Nous nous apprêtons à utiliser la classe `URLVariables()`. Nous vous invitons donc à prendre connaissance (si vous ne l'avez pas déjà fait) du contenu de la page précédente dédiée à la manipulation des variables contenues dans une instance de type `URLLoader()`.

1. Instanciez la classe `URLVariables()`.

```
var variablesURL:URLVariables = new URLVariables();
```

2. Ajoutez autant de variables que nécessaire.

```
variablesURL.prenom="david";
```

3. Injectez l'ensemble des variables dans l'instance de type `URLRequest()` à l'aide de la propriété `data`.

```
adresseDonnees.data = variablesURL;
```

4. Chargez cette adresse à l'aide de la méthode `load()`.

```
chargeurDonnees.load(adresseDonnees);
```

Il ne reste plus qu'à réceptionner les variables en retour de la requête. En reprenant l'exemple précédent du chargement des données dans une instance de la classe `URLLoader()`, vous devriez obtenir le script global suivant :

Fichier de référence : `Chapitre12/urlLoader3.fla`

```
var variablesURL:URLVariables = new URLVariables();
variablesURL.prenom="david";

var chargeurDonnees:URLLoader = new URLLoader();
var adresseDonnees:URLRequest = new URLRequest("http://www.yazo.net/racine/
➥Tests/requete2.php");

adresseDonnees.data = variablesURL;
chargeurDonnees.load(adresseDonnees);

chargeurDonnees.addEventListener(Event.COMPLETE, donneesChargees);

function donneesChargees(event:Event):void {
  trace(chargeurDonnees.data);
}
prenom=David&nom=TARDIVEAU&adressemail=david@yazo.net
```

Envoyer et recevoir des variables d'une URL

En nous basant sur les trois développements précédents, nous avons réalisé un dernier exemple qui présente l'envoi de variables à une URL avant d'en réceptionner d'autres.

Fichier de référence : `Chapitre12/urlLoader4.fla`

```
var variablesURL:URLVariables = new URLVariables();
variablesURL.prenom="david";
var variablesChargees:URLVariables;

var chargeurDonnees:URLLoader = new URLLoader();
var adresseDonnees:URLRequest = new URLRequest("http://www.yazo.net/racine/
➥Tests/requete2.php");

adresseDonnees.data = variablesURL;
chargeurDonnees.load(adresseDonnees);

chargeurDonnees.addEventListener(Event.COMPLETE, donneesChargees);

function donneesChargees(event:Event):void {
  variablesChargees = new URLVariables(chargeurDonnees.data);
  trace(variablesChargees.prenom);
  trace(variablesChargees.nom);
  trace(variablesChargees.adressemail);
}
David
TARDIVEAU
david@yazo.net
```

Pour conclure ces explications sur le chargement et l'envoi de données à un serveur, nous pouvons dire que ce script constitue la base de tous ceux que vous serez amené à concevoir pour échanger des données entre une animation Flash et une base de données ou une page Web dynamique.

13

Gérer le XML dans Flash

Si le formalisme XML n'existait pas ou bien s'il était impossible de l'utiliser dans Flash, les développements ne seraient pas ce qu'ils sont aujourd'hui !

Pourquoi une telle affirmation ?

Dans une animation, nous avons sans cesse besoin de faire appel à des informations et nous devons, de ce fait, les organiser et les manipuler. Structurer l'information en la hiérarchisant permet de consulter rapidement et simplement une donnée. Le formalisme XML est donc la solution la plus adaptée pour concevoir des documents bien ordonnés.

Pour vous expliquer le fonctionnement du XML, nous avons exploité plusieurs documents présentant chacun leur propre hiérarchie de l'information. Nous avons ainsi créé quatre fichiers XML contenant des lignes de code avec les 22 régions de France.

Nous n'avons pas souhaité utiliser un seul document XML complet vous présentant tous les cas de figures que nous allons évoquer, car nous estimons que l'arborescence XML obtenue aurait été trop complexe pour les premières explications. Nous avons donc conçu des exemples de difficulté progressive d'un fichier à l'autre.

Le dernier document XML que nous avons rédigé pourra servir de base de travail à vos propres exercices pratiques, dès que vous aurez assimilé les notions présentées. C'est pourquoi, il est un peu long et plus complexe. Vous pourrez ainsi imaginer plusieurs cas de figures.

Créer une source XML

Si la rédaction d'un document XML n'est pas très complexe, sa conception l'est tout de même un peu. En effet, on ne peut pas se lancer dans l'écriture d'un fichier XML sans analyser préalablement l'organisation de son contenu.

Première étape

La création d'un document XML débute par une phase de réflexion au cours de laquelle vous devez recenser les informations devant figurer dans le fichier XML. Il faut ensuite regrouper et hiérarchiser les données récoltées pour obtenir une arborescence qu'il est ensuite facile de retranscrire sous forme d'une structure XML.

Exemple : nous souhaitons utiliser les noms des 22 régions de France (`Chapitre13/franceregions.xml`, `Chapitre13/franceregions2.xml` et `Chapitre13/franceregions3.xml`), ainsi que ceux des départements associés (`Chapitre13/france.xml`). Voici l'organisation des données que nous avons choisie :

> **Remarque**
>
> Dans le fichier `france.xml` vous allez constater que nous avons décidé de regrouper les départements français par région. Mais, nous aurions tout autant pu rédiger une liste des départements pour lesquels nous aurions indiqué une région de rattachement en tant qu'attribut. Dans ce dernier cas, une répétition inutile du nom des régions aurait été inéluctable.

Fichier `franceregions.xml` :

```
<France>
  <Region nom="Alsace"/>
  <Region nom="Aquitaine"/>
  <Region nom="Auvergne"/>
  <Region nom="Basse-Normandie"/>
  <Region nom="Bourgogne"/>
  <Region nom="Bretagne"/>
  <Region nom="Centre"/>
  <Region nom="Champagne-Ardenne"/>
  <Region nom="Corse"/>
  <Region nom="Franche-Comté"/>
  <Region nom="Haute-Normandie"/>
  <Region nom="Ile-de-France"/>
  <Region nom="Languedoc-Roussillon"/>
  <Region nom="Limousin"/>
  <Region nom="Lorraine"/>
  <Region nom="Midi-Pyrénées"/>
  <Region nom="Nord-Pas-de-Calais"/>
  <Region nom="Pays de la Loire"/>
  <Region nom="Picardie"/>
  <Region nom="Poitou-Charentes"/>
  <Region nom="Provence-Alpes-Côte d'azur"/>
  <Region nom="Rhônes-Alpes"/>
</France>
```

Fichier `franceregions2.xml` :

```
<France>
  <Region>Alsace</Region>
  <Region>Aquitaine</Region>
  <Region>Auvergne</Region>
  <Region>Basse-Normandie</Region>
  <Region>Bourgogne</Region>
  <Region>Bretagne</Region>
  <Region>Centre</Region>
  <Region>Champagne-Ardenne</Region>
  <Region>Corse</Region>
  <Region>Franche-Comté</Region>
  <Region>Haute-Normandie</Region>
  <Region>Ile-de-France</Region>
  <Region>Languedoc-Roussillon</Region>
  <Region>Limousin</Region>
  <Region>Lorraine</Region>
  <Region>Midi-Pyrénées</Region>
  <Region>Nord-Pas-de-Calais</Region>
  <Region>Pays de la Loire</Region>
  <Region>Picardie</Region>
  <Region>Poitou-Charentes</Region>
  <Region>Provence-Alpes-Côte d'azur</Region>
  <Region>Rhônes-Alpes</Region>
</France>
```

Ces deux fichiers, `franceregions.xml` et `franceregions2.xml`, contiennent la même information, c'est-à-dire les noms des 22 régions de France. Nous avons volontairement utilisé les attributs de nœuds pour stocker les 22 noms dans le fichier `franceregions.xml` alors que dans le fichier `franceregions2.xml`, ces mêmes informations ont été placées en tant que valeur de nœud. Rassurez-vous, nous allons très vite vous expliquer la différence entre ces deux techniques.

Pour vous présenter justement la notion d'attribut, nous avons conçu un troisième document qui contient une information supplémentaire, le nom des habitants des différentes régions de France.

Fichier `franceregions3.xml` (source : `http://fr.wikipedia.org`) :

```
<France>
  <Region nomhabitants="Alsaciens & Alsaciennes">Alsace</Region>
  <Region nomhabitants="Aquitains & Aquitaines">Aquitaine</Region>
  <Region nomhabitants="Auvergnats & Auvergnates">Auvergne</Region>
  <Region nomhabitants="Bas-Normands & Bas-Normandes">Basse-Normandie</Region>
  <Region nomhabitants="Bourguignons & Bourguignonnes">Bourgogne</Region>
  <Region nomhabitants="Bretons & Bretonnes">Bretagne</Region>
  <Region nomhabitants="Les habitants de la région Centre !">Centre</Region>
  <Region nomhabitants="Champardennais & Champardennaises">Champagne-Ardenne</Region>
  <Region nomhabitants="Corses & Corses">Corse</Region>
  <Region nomhabitants="Francs-Comtois & Francs-Comtoises">Franche-Comté</Region>
```

```
  <Region nomhabitants="Hauts-Normands & Hauts-Normandes">Haute-Normandie</Region>
  <Region nomhabitants="Franciliens & Franciliennes">Ile-de-France</Region>
  <Region nomhabitants="Langdociens-Roussillonnais & Langdociens-Roussillonnais">
  ➥Languedoc-Roussillon</Region>
  <Region nomhabitants="Limousins & Limousines">Limousin</Region>
  <Region nomhabitants="Lorrains & Lorraines">Lorraine</Region>
  <Region nomhabitants="Les habitants de la région Midi-Pyrénées !">
  ➥Midi-Pyrénées</Region>
  <Region nomhabitants="Nord-Pas-de-Calaisiens & Nord-Pas-de-Calaisiennes">
  ➥Nord-Pas-de-Calais</Region>
  <Region nomhabitants="Les habitants de la région Pays de la Loire !">
  ➥Pays de la Loire</Region>
  <Region nomhabitants="Picards & Picardes">Picardie</Region>
  <Region nomhabitants="Picto-Charentais & Picto-Charentaises">Poitou-Charentes
  ➥</Region>
  <Region nomhabitants="Les habitants de la région Provence-Alpes-Côte d'azur !">
  ➥Provence-Alpes-Côte d'azur</Region>
  <Region nomhabitants="Rhônalpins & Rhônalpines">Rhônes-Alpes</Region>
</France>
```

Ce quatrième document est volontairement long et plus complexe afin que nous puissions manipuler une structure qui se prête aux nombreux cas de traitements abordés dans le tableau de synthèse à la fin de ce chapitre.

Fichier `france.xml` (source : `www.insee.fr`)

```
<France>

<Region nom="Alsace">
  <Departement nom="Bas-Rhin" numero="67" prefecture="Strasbourg"/>
  <Departement nom="Haut-Rhin" numero="68" prefecture="Colmar"/>
  <Population>
    <Annee numero="1990">1 624 372</Annee>
    <Annee numero="1999">1 734 145</Annee>
    <Annee numero="2005">1 806 069</Annee>
    <Annee numero="2006">1 816 841</Annee>
  </Population>
</Region>

<Region nom="Aquitaine">
  <Departement nom="Dordogne" numero="24" prefecture="Périgueux"/>
  <Departement nom="Gironde" numero="33" prefecture="Bordeaux"/>
  <Departement nom="Landes" numero="40" prefecture="Mont-de-Marsan"/>
  <Departement nom="Lot-et-Garonne" numero="47" prefecture="Agen"/>
  <Departement nom="Pyrénées-Atlantiques" numero="64" prefecture="Pau"/>
  <Population>
    <Annee numero="1990">2 795 830</Annee>
    <Annee numero="1999">2 908 359</Annee>
    <Annee numero="2005">3 080 091</Annee>
    <Annee numero="2006">3 098 819</Annee>
  </Population>
</Region>
```

```
<Region nom="Auvergne">
  <Departement nom="Allier" numero="3" prefecture="Moulins"/>
  <Departement nom="Cantal" numero="15" prefecture="Aurillac"/>
  <Departement nom="Haute-Loire" numero="43" prefecture="Le Puy-en-Vellay"/>
  <Departement nom="Puy-de-Dôme" numero="63" prefecture="Clermont-Ferrand"/>
  <Population>
    <Annee numero="1990">1 321 214</Annee>
    <Annee numero="1999">1 308 878</Annee>
    <Annee numero="2005">1 331 380</Annee>
    <Annee numero="2006">1 333 530</Annee>
  </Population>
</Region>

<Region nom="Basse-Normandie">
  <Departement nom="Calvados" numero="14" prefecture="Caen"/>
  <Departement nom="Manche" numero="50" prefecture="Saint-Lô"/>
  <Departement nom="Orne" numero="61" prefecture="Alençon"/>
  <Population>
    <Annee numero="1990">1 391 318</Annee>
    <Annee numero="1999">1 422 193</Annee>
    <Annee numero="2005">1 445 732</Annee>
    <Annee numero="2006">1 448 857</Annee>
  </Population>
</Region>

<Region nom="Bourgogne">
  <Departement nom="Côte-d'Or" numero="21" prefecture="Dijon"/>
  <Departement nom="Nièvre" numero="58" prefecture="Nevers"/>
  <Departement nom="Saône-et-Loire" numero="71" prefecture="Mâcon"/>
  <Departement nom="Yonne" numero="89" prefecture="Auxerre"/>
  <Population>
    <Annee numero="1990">1 609 653</Annee>
    <Annee numero="1999">1 610 067</Annee>
    <Annee numero="2005">1 622 542</Annee>
    <Annee numero="2006">1 623 913</Annee>
  </Population>
</Region>

<Region nom="Bretagne">
  <Departement nom="Côtes d'Armor" numero="22" prefecture="Saint-Brieuc"/>
  <Departement nom="Finistère" numero="29" prefecture="Quimper"/>
  <Departement nom="Ille-et-Vilaine" numero="35" prefecture="Rennes"/>
  <Departement nom="Morbihan" numero="56" prefecture="Vannes"/>
  <Population>
    <Annee numero="1990">2 795 638</Annee>
    <Annee numero="1999">2 906 197</Annee>
    <Annee numero="2005">3 062 117</Annee>
    <Annee numero="2006">3 080 648</Annee>
  </Population>
</Region>
```

```
<Region nom="Centre">
  <Departement nom="Cher" numero="18" prefecture="Bourges"/>
  <Departement nom="Eure-et-Loir" numero="28" prefecture="Chartres"/>
  <Departement nom="Indre" numero="36" prefecture="Châteauroux"/>
  <Departement nom="Indre-et-Loire" numero="37" prefecture="Tours"/>
  <Departement nom="Loir-et-Cher" numero="41" prefecture="Blois"/>
  <Departement nom="Loiret" numero="45" prefecture="Orléans"/>
  <Population>
    <Annee numero="1990">2 371 036</Annee>
    <Annee numero="1999">2 440 329</Annee>
    <Annee numero="2005">2 496 654</Annee>
    <Annee numero="2006">2 505 288</Annee>
  </Population>
</Region>

<Region nom="Champagne-Ardenne">
  <Departement nom="Ardennes" numero="8" prefecture="Charleville-Mézières"/>
  <Departement nom="Aube" numero="10" prefecture="Troyes"/>
  <Departement nom="Marne" numero="51" prefecture="Châlons-en-Champagne"/>
  <Departement nom="Haute-Marne" numero="52" prefecture="Chaumont"/>
  <Population>
    <Annee numero="1990">1 347 848</Annee>
    <Annee numero="1999">1 342 363</Annee>
    <Annee numero="2005">1 337 672</Annee>
    <Annee numero="2006">1 338 590</Annee>
  </Population>
</Region>

<Region nom="Corse">
  <Departement nom="Corse-du-Sud" numero="2A" prefecture="Ajaccio"/>
  <Departement nom="Haute-Corse" numero="2B" prefecture="Bastia"/>
  <Population>
    <Annee numero="1990">250 371</Annee>
    <Annee numero="1999">260 196</Annee>
    <Annee numero="2005">276 911</Annee>
    <Annee numero="2006">278 650</Annee>
  </Population>
</Region>

<Region nom="Franche-Comté">
  <Departement nom="Doubs" numero="25" prefecture="Besançon"/>
  <Departement nom="Jura" numero="39" prefecture="Lons-le-Saunier"/>
  <Departement nom="Haute-Saône" numero="70" prefecture="Vesoul"/>
  <Departement nom="Territoire-de-Belfort" numero="90" prefecture="Belfort"/>
  <Population>
    <Annee numero="1990">1 097 276</Annee>
    <Annee numero="1999">1 117 059</Annee>
    <Annee numero="2005">1 141 861</Annee>
    <Annee numero="2006">1 146 139</Annee>
  </Population>
</Region>

<Region nom="Haute-Normandie">
  <Departement nom="Eure" numero="27" prefecture="Évreux"/>
```

```
  <Departement nom="Seine-Maritime" numero="76" prefecture="Rouen"/>
  <Population>
    <Annee numero="1990">1 737 247</Annee>
    <Annee numero="1999">1 780 192</Annee>
    <Annee numero="2005">1 805 955</Annee>
    <Annee numero="2006">1 811 241</Annee>
  </Population>
</Region>

<Region nom="Ile-de-France">
  <Departement nom="Paris" numero="75" prefecture="Paris"/>
  <Departement nom="Seine-et-Marne" numero="77" prefecture="Melun"/>
  <Departement nom="Yvelines" numero="78" prefecture="Versailles"/>
  <Departement nom="Essonne" numero="91" prefecture="Évry"/>
  <Departement nom="Hauts-de-Seine" numero="92" prefecture="Nanterre"/>
  <Departement nom="Seine-Saint-Denis" numero="93" prefecture="Bobigny"/>
  <Departement nom="Val-de-Marne" numero="94" prefecture="Créteil"/>
  <Departement nom="Val-d'Oise" numero="95" prefecture="Pontoise"/>
  <Population>
    <Annee numero="1990">10 660 554</Annee>
    <Annee numero="1999">10 952 011</Annee>
    <Annee numero="2005">11 399 319</Annee>
    <Annee numero="2006">11 491 046</Annee>
  </Population>
</Region>

<Region nom="Languedoc-Roussillon">
  <Departement nom="Aude" numero="11" prefecture="Carcassonne"/>
  <Departement nom="Gard" numero="30" prefecture="Nîmes"/>
  <Departement nom="Hérault" numero="34" prefecture="Montpelier"/>
  <Departement nom="Lozère" numero="48" prefecture="Mende"/>
  <Departement nom="Pyrénées-Orientales" numero="66" prefecture="Perpignan"/>
  <Population>
    <Annee numero="1990">2 114 985</Annee>
    <Annee numero="1999">2 295 648</Annee>
    <Annee numero="2005">2 496 871</Annee>
    <Annee numero="2006">2 519 707</Annee>
  </Population>
</Region>

<Region nom="Limousin">
  <Departement nom="Corrèze" numero="19" prefecture="Tulle"/>
  <Departement nom="Creuse" numero="23" prefecture="Guéret"/>
  <Departement nom="Haute-Vienne" numero="87" prefecture="Limoges"/>
  <Population>
    <Annee numero="1990">722 850</Annee>
    <Annee numero="1999">710 939</Annee>
    <Annee numero="2005">724 243</Annee>
    <Annee numero="2006">725 301</Annee>
  </Population>
</Region>

<Region nom="Lorraine">
  <Departement nom="Meurthe-et-Moselle" numero="54" prefecture="Nancy"/>
  <Departement nom="Meuse" numero="55" prefecture="Bar-le-Duc"/>
```

```
  <Departement nom="Moselle" numero="57" prefecture="Metz"/>
  <Departement nom="Vosges" numero="88" prefecture="Epinal"/>
  <Population>
    <Annee numero="1990">2 305 726</Annee>
    <Annee numero="1999">2 310 376</Annee>
    <Annee numero="2005">2 334 245</Annee>
    <Annee numero="2006">2 338 744</Annee>
  </Population>
</Region>

<Region nom="Midi-Pyrénées">
  <Departement nom="Ariège" numero="9" prefecture="Foix"/>
  <Departement nom="Aveyron" numero="12" prefecture="Rodez"/>
  <Departement nom="Haute-Garonne" numero="31" prefecture="Toulouse"/>
  <Departement nom="Gers" numero="32" prefecture="Auch"/>
  <Departement nom="Lot" numero="46" prefecture="Cahors"/>
  <Departement nom="Hautes-Pyrénées" numero="65" prefecture="Tarbes"/>
  <Departement nom="Tarn" numero="81" prefecture="Albi"/>
  <Departement nom="Tarn-et-Garonne" numero="82" prefecture="Montauban"/>
  <Population>
    <Annee numero="1990">2 430 663</Annee>
    <Annee numero="1999">2 551 687</Annee>
    <Annee numero="2005">2 734 954</Annee>
    <Annee numero="2006">2 755 383</Annee>
  </Population>
</Region>

<Region nom="Nord-Pas-de-Calais">
  <Departement nom="Nord" numero="59" prefecture="Lille"/>
  <Departement nom="Pas-de-Calais" numero="62" prefecture="Arras"/>
  <Population>
    <Annee numero="1990">3 965 058</Annee>
    <Annee numero="1999">3 996 588</Annee>
    <Annee numero="2005">4 032 135</Annee>
    <Annee numero="2006">4 043 050</Annee>
  </Population>
</Region>

<Region nom="Pays de la Loire">
  <Departement nom="Loire-Atlantique" numero="44" prefecture="Nantes"/>
  <Departement nom="Maine-et-Loire" numero="49" prefecture="Angers"/>
  <Departement nom="Mayenne" numero="53" prefecture="Laval"/>
  <Departement nom="Sarthe" numero="72" prefecture="Le Mans"/>
  <Departement nom="Vendée" numero="85" prefecture="La Roche-sur-Yon"/>
  <Population>
    <Annee numero="1990">3 059 112</Annee>
    <Annee numero="1999">3 222 061</Annee>
    <Annee numero="2005">3 400 745</Annee>
    <Annee numero="2006">3 426 371</Annee>
  </Population>
</Region>

<Region nom="Picardie">
  <Departement nom="Aisne" numero="2" prefecture="Laon"/>
  <Departement nom="Oise" numero="60" prefecture="Beauvais"/>
```

```
  <Departement nom="Somme" numero="80" prefecture="Amiens"/>
  <Population>
    <Annee numero="1990">1 810 687</Annee>
    <Annee numero="1999">1 857 481</Annee>
    <Annee numero="2005">1 880 890</Annee>
    <Annee numero="2006">1 886 445</Annee>
  </Population>
</Region>

<Region nom="Poitou-Charentes">
  <Departement nom="Charente" numero="16" prefecture="Angoulême"/>
  <Departement nom="Charente-Maritime" numero="17" prefecture="La Rochelle"/>
  <Departement nom="Deux-Sèvres" numero="79" prefecture="Niort"/>
  <Departement nom="Vienne" numero="86" prefecture="Poitiers"/>
  <Population>
    <Annee numero="1990">1 595 109</Annee>
    <Annee numero="1999">1 640 068</Annee>
    <Annee numero="2005">1 705 347</Annee>
    <Annee numero="2006">1 712 652</Annee>
  </Population>
</Region>

<Region nom="Provence-Alpes-Côte d'azur">
  <Departement nom="Alpes de Hautes-Provence" numero="4" prefecture="Digne"/>
  <Departement nom="Hautes-Alpes" numero="5" prefecture="Gap"/>
  <Departement nom="Alpes-Maritimes" numero="6" prefecture="Nice"/>
  <Departement nom="Bouches-du-Rhône" numero="13" prefecture="Marseille"/>
  <Departement nom="Var" numero="83" prefecture="Toulon"/>
  <Departement nom="Vaucluse" numero="84" prefecture="Avignon"/>
  <Population>
    <Annee numero="1990">4 257 907</Annee>
    <Annee numero="1999">4 506 151</Annee>
    <Annee numero="2005">4 750 947</Annee>
    <Annee numero="2006">4 780 989</Annee>
  </Population>
</Region>

<Region nom="Rhônes-Alpes">
  <Departement nom="Ain" numero="1" prefecture="Bourg-en-Bresse"/>
  <Departement nom="Ardèche" numero="7" prefecture="Privas"/>
  <Departement nom="Drôme" numero="26" prefecture="Valence"/>
  <Departement nom="Isère" numero="38" prefecture="Grenoble"/>
  <Departement nom="Loire" numero="42" prefecture="Saint-Étienne"/>
  <Departement nom="Rhône" numero="69" prefecture="Lyon"/>
  <Departement nom="Savoie" numero="73" prefecture="Chambéry"/>
  <Departement nom="Haute-Savoie" numero="74" prefecture="Annecy"/>
  <Population>
    <Annee numero="1990">5 350 701</Annee>
    <Annee numero="1999">5 645 407</Annee>
    <Annee numero="2005">5 958 320</Annee>
    <Annee numero="2006">6 004 957</Annee>
  </Population>
</Region>

</France>
```

Nous sommes bien conscients que ces quatre documents XML ne signifient rien pour vous, mais nous allons devoir nous y référer très souvent. Nous les avons donc présentés ici, afin que vous puissiez venir les consulter dès que nous les évoquerons dans nos explications et nos exemples. D'autres fichiers XML sont également utilisés au travers des différents exemples de ce chapitre ; nous les présenterons au cas par cas.

Deuxième étape

Elle consiste à retranscrire le résultat de l'analyse de la première étape sous forme de balises XML.

Dans un premier temps, si nous faisions abstraction de la notion d'attribut, il suffirait d'écrire un document texte sous forme de table des matières ou de plan (tel qu'il est proposé dans Microsoft Word) auquel il ne resterait plus qu'à ajouter au début et à la fin des lignes du document les balises XML.

Dans les paragraphes qui vont suivre, nous allons analyser la structure d'un document XML et vous constaterez qu'il existe des nœuds et des attributs. Lorsque vous aurez compris la différence entre ces deux notions, vous vous demanderez si vous devez privilégier l'usage des attributs ou des nœuds. Il n'y a pas vraiment de réponse à cette question, tout dépend de votre préférence, mais gardez tout de même à l'esprit qu'il est plus pratique de saisir une information assez longue en tant que valeur de nœud plutôt qu'une valeur d'attribut.

> **Remarque**
>
> Dans les faits, la première étape est souvent réalisée de tête ou à l'aide d'un simple schéma sur une feuille de papier.

Créer un fichier

Commencez par lancer un éditeur de texte, celui qui vous convient le mieux. Il n'y a pas de meilleur logiciel que celui que vous utilisez déjà. Tant que ce dernier gère le format Unicode, n'en changez pas.

Pour celles et ceux qui n'ont jamais utilisé d'éditeur de texte ou de code, sachez que Dreamweaver est capable de générer des fichiers XML et de travailler avec, mais que vous pouvez également trouver sur Internet des freewares et sharewares et en choisir un en fonction des interfaces qui vous semblent plus familières. Il est très difficile de vous conseiller un logiciel sous Windows tant le choix est grand. Sur Mac, il en existe également quelques-uns, mais le choix se restreint très vite à une poignée de logiciels vraiment efficaces. TextWrangler a le mérite d'être gratuit, très simple à prendre en main et de proposer la coloration syntaxique.

> **Coloration syntaxique**
>
> Pour faciliter la rédaction et la lecture des documents contenant du code informatique, certains éditeurs de texte mettent en couleur les mots importants des langages de programmation. C'est ce qu'on appelle la coloration syntaxique.

1. Créez un nouveau document.
2. Enregistrez-le avec un nom représentatif des informations qu'il contient. Rappelons que vous ne devez pas utiliser de caractères accentués ou spéciaux. Il est capital que vous choisissiez le format Unicode ou UTF-8 pour l'enregistrement de votre document, afin que tous les caractères saisis dans le fichier soient reconnus par Flash.
3. Saisissez votre code XML (consultez le développement suivant, Structure élémentaire d'un fichier XML).
4. Effectuez un dernier enregistrement de votre document.
5. Faites glisser votre document dans la fenêtre d'un navigateur pour vous assurer qu'il est valide.

Cette dernière étape vous permet de vérifier si votre document contient des erreurs avant que Flash ne vous le signale dans la fenêtre Sortie. Le code d'erreur et l'information affichés par Flash ne sont pas toujours très parlants comparés au message signalé par un navigateur.

Structure élémentaire d'un fichier XML

Elle repose sur l'imbrication de nœuds intitulés, de ce fait, des nœuds enfants. La structure d'un fichier XML peut varier considérablement pour un même résultat, tout dépend des habitudes et de l'expérience de chacun.

La balise

Comme nous l'avons évoqué à plusieurs reprises, il est nécessaire de structurer l'information afin qu'elle soit plus facilement accessible et surtout plus lisible. Cette structuration pourrait se traduire visuellement sous plusieurs formes de présentations. Selon vous, quelle est la solution la plus exploitable pour un traitement de l'information : une liste ou un tableau ?

Sous forme de liste :

Alsace, Aquitaine, Auvergne, Basse-Normandie, Bourgogne, Bretagne, Centre, Champagne-Ardenne, Corse, Franche-Comté, Haute-Normandie, Ile-de-France, Languedoc-Roussillon, Limousin, Lorraine, Midi-Pyrénées, Nord-Pas-de-Calais, Pays de la Loire, Picardie, Poitou-Charentes, Provence-Alpes-Côte d'azur, Rhônes-Alpes.

Sous forme de tableau :

1	Alsace
2	Aquitaine
3	Auvergne
4	Basse-Normandie
5	Bourgogne
6	Bretagne

7	Centre
8	Champagne-Ardenne
9	Corse
10	Franche-Comté
11	Haute-Normandie
12	Ile-de-France
13	Languedoc-Roussillon
14	Limousin
15	Lorraine
16	Midi-Pyrénées
17	Nord-Pas-de-Calais
18	Pays de la Loire
19	Picardie
20	Poitou-Charentes
21	Provence-Alpes-Côte d'azur
22	Rhônes-Alpes

Il est plus simple de faire référence à la dixième ligne du tableau plutôt qu'au dixième mot de la liste (même si la numérotation des lignes n'était pas présente, il est plus facile de compter des lignes que de chercher des virgules dans une liste pour compter des mots).

Gardons ce type de présentation pour vous expliquer la structure d'un document XML.

Le tableau que nous venons de réaliser ne possède pas de titre, ce qui peut s'avérer ambigu sur la représentation et la signification des données qu'il contient. Nous devons donc en ajouter un et choisir par exemple : Les 22 régions de France. Plus généralement, si nous avions à réaliser plusieurs tableaux présentant les différentes régions de plusieurs pays nous pourrions simplifier le titre par France.

Depuis votre plus tendre enfance ou votre adolescence, vous avez appris à lire un tableau. À chaque nouvelle ligne, une nouvelle information est stockée. Le mot Normandie se trouve à la suite du terme Basse car c'est un nom composé. En revanche, Bretagne se trouve sur la ligne qui suit celle contenant Bourgogne. Nous n'aurions pas pu saisir Bourgogne-Bretagne. Toutes ces affirmations vous paraissent sûrement absurdes, mais dans ce cas, c'est que vous connaissez bien la géographie française. Si nous écrivions les noms des régions d'un pays que vous ne connaissez pas, les lignes du tableau joueraient dans ce cas un rôle capitale de séparation de l'information.

Une ligne de tableau se trouve sur un axe horizontal qui débute par un trait vertical et se termine par un autre.

21	Provence-Alpes-Côte d'azur

Pour concevoir un document XML, vous devez donc imaginer les informations que vous souhaitez stocker sous forme de tableau. Vous remplacerez alors les limites des lignes par des balises.

```
<Region>Provence-Alpes-Côte d'azur</Region>
```

Observez bien la différence entre la balise d'ouverture et celle qui termine la ligne. La dernière débute par une barre oblique (*slash*) pour indiquer qu'elle ferme la première.

Mais qu'est-ce qu'une balise ? Vous l'aurez peut-être compris, il s'agit d'un système de marquage qui délimite le début et la fin d'une information. Grâce aux balises, nous pouvons séparer hiérarchiquement plusieurs informations.

Pourquoi cette séparation est-elle hiérarchique ? Dans la mesure où il est possible d'imbriquer les balises les unes à l'intérieur des autres, vous allez générer tout naturellement une arborescence.

```
<Region>
<Departement>Corrèze</Departement>
<Departement>Creuse</Departement>
<Departement>Haute-Vienne</Departement>
</Region>
```

Remarque

Le nom d'une balise ne doit pas comporter de caractères accentués ni même de caractères spéciaux ou d'espace.

Dans notre dernier exemple, nous possédons un document XML avec un nœud racine dont le nom de balise est `Region`.

La balise autofermante

Parfois, l'information contenue entre deux balises est presque plus courte que le nom des balises elles-mêmes. Il est alors intéressant d'utiliser des balises autofermantes accompagnées d'un attribut (nous aborderons cette notion, l'attribut, dans quelques pages).

```
<Region>
<Departement nom="Corrèze"/>
<Departement nom="Creuse"/>
<Departement nom="Haute-Vienne"/>
</Region>
```

En conclusion

Vous devez toujours garder à l'esprit qu'une balise est un marqueur. De ce fait, elles fonctionnent par paire (à l'exception des balises autofermantes), une qui ouvre et l'autre qui ferme le nœud.

Les termes fermer ou ouvrir une balise ou un nœud signifient respectivement insérer une balise avec ou sans barre oblique (slash).

Le nœud

Pour expliquer clairement et simplement ce qu'est un nœud, rien ne vaut un exemple.

Nous sommes partis du fichier `France.xml` et nous l'avons modifié afin qu'il soit plus facile à manipuler.

Le premier nœud de cette arborescence possède le nom `France`. Il s'agit précisément du nœud racine. Ce dernier est qualifié ainsi car il contient tous les nœuds du document. Lorsque nous parlons du premier nœud d'un document XML, nous faisons toujours référence au premier nœud qui se trouve dans le nœud racine. Dans notre exemple, le premier nœud est donc celui de l'Alsace.

Le nœud racine possède cinq nœuds enfants dont les noms de balises sont `Region`.

Le premier nœud possède deux nœuds enfants dont les noms de balises sont `Departement`. Le deuxième nœud possède cinq nœuds enfants.

Le cinquième nœud de cette arborescence possède cinq nœuds enfants. Le cinquième nœud enfant possède à son tour quatre nœuds enfants dont les noms de balises sont `Annee`.

```
<France>

<Region nom="Alsace">
  <Departement nom="Bas-Rhin" numero="67">Strasbourg</Departement>
  <Departement nom="Haut-Rhin" numero="68">Colmar</Departement>
</Region>

<Region nom="Aquitaine">
  <Departement nom="Dordogne" numero="24">Périgueux</Departement>
  <Departement nom="Gironde" numero="33">Bordeaux</Departement>
  <Departement nom="Landes" numero="40">Mont-de-Marsan</Departement>
  <Departement nom="Lot-et-Garonne" numero="47">Agen</Departement>
  <Departement nom="Pyrénées-Atlantiques" numero="64">Pau</Departement>
</Region>

<Region nom="Auvergne">
  <Departement nom="Allier" numero="3">Moulins</Departement>
  <Departement nom="Cantal" numero="15">Aurillac</Departement>
  <Departement nom="Haute-Loire" numero="43">Le Puy-en-Vellay</Departement>
  <Departement nom="Puy-de-Dôme" numero="63">Clermont-Ferrand</Departement>
</Region>

<Region nom="Basse-Normandie">
  <Departement nom="Calvados" numero="14">Caen</Departement>
  <Departement nom="Manche" numero="50">Saint-Lô</Departement>
  <Departement nom="Orne" numero="61">Alençon</Departement>
</Region>

<Region nom="Bourgogne">
  <Departement nom="Côte-d'Or" numero="21">Dijon</Departement>
  <Departement nom="Nièvre" numero="58">Nevers</Departement>
```

```
    <Departement nom="Saône-et-Loire" numero="71">Mâcon</Departement>
    <Departement nom="Yonne" numero="89">Auxerre</Departement>
    <Population>
      <Annee numero="1990">1 609 653</Annee>
      <Annee numero="1999">1 610 067</Annee>
      <Annee numero="2005">1 622 542</Annee>
      <Annee numero="2006">1 623 913</Annee>
    </Population>
  </Region>

  </France>
```

Comme vous l'aurez compris, un nœud correspond donc à une information dans une arborescence (comme le nœud `<Region nom="Alsace"></Region>` ou encore `<Departement nom="Calvados" numero="14">Caen</Departement>`).

Le nœud est également qualifié de branche lorsqu'il possède un ou plusieurs nœuds enfants.

Nous vous le précisions déjà dans le développement précédent sur la notion de Balise, un nœud possède toujours deux balises (une ouvrante et une fermante), à l'exception des balises autofermantes.

L'attribut

Comme nous venons de le voir, un nœud n'est ni plus ni moins qu'une information qui peut en contenir une ou plusieurs autres. Comment ces dernières peuvent-elles être contenues ? C'est la question que vous vous poserez bien souvent car deux possibilités s'offrent à vous, l'une aussi avantageuse que l'autre (sauf dans le cas d'une information assez longue).

Voici une première arborescence que nous pourrions exploiter dans une animation Flash :

```
<Alsace>

  <Departement>
    <Nom>Bas-Rhin</Nom>
    <Numero>67</Numero>
    <Prefecture>Strasbourg</Prefecture>
  </Departement>

  <Departement>
    <Nom>Haut-Rhin</Nom>
    <Numero>68</Numero>
    <Prefecture>Colmar</Prefecture>
  </Departement>

</Alsace>
```

Vous pouvez constater que toutes les informations rattachées à un département sont contenues en tant que nœud. Les valeurs `Bas-Rhin`, `67` et `Strasbourg` comportent peu de

signes (chiffres et caractères). De ce fait, nous aurions tout à fait pu simplifier notre arborescence de la façon suivante :

```
<Region nom="Alsace">
  <Departement nom="Bas-Rhin" numero="67" prefecture="Strasbourg"/>
  <Departement nom="Haut-Rhin" numero="68" prefecture="Colmar"/>
</Region>
```

Cet autre constat est flagrant : il semblerait que l'usage des attributs simplifie la structure des documents XML, mais comme nous l'évoquions précédemment, tout dépend de la quantité de signes à stocker dans un nœud ou en tant que valeur d'attribut.

Un attribut correspond avant tout à la structure suivante :

```
<NomdeLaBalise NomDeLattribut="Unevaleur"/>
```

ou encore :

```
<NomdeLaBalise NomDeLattribut="Unevaleur">
  <UneAutreBalise>Une valeur</UneAutreBalise>
</NomdeLaBalise>
```

Dans le premier cas, elle s'insère dans la balise autofermante, dans le deuxième cas, elle est placée dans la balise ouvrante d'un nœud. Quelle que soit la méthode, il s'agit toujours d'un paire `nomAttribut="Valeur de l'attribut"`. Le nom de l'attribut ne doit pas contenir de caractères accentués ou spéciaux, ni même d'espace. La valeur associée à l'attribut doit toujours être saisie entre guillemets.

L'exemple qui va suivre illustre que l'usage de l'attribut possède une limite.

```
<Marine>Mais si tu ne te souviens pas de l'endroit où tu as placé ton PDA, comment
allons nous pouvoir retrouver notre chemin, car nous ne pouvons pas utiliser ton
application de géolocalisation. Ce n'est pas très malin de ta part.</Marine>

<Luna>Que tu es bête ! Je n'ai pas oublié de prendre mon GPS qui est tout de même plus
pratique que mon PDA. De plus, le son du JerryJerry est de bien meilleure qualité que
mon Achetéssé...</Luna>

<Marine>Oui, tu as raison.</Marine>
```

Les trois nœuds ci-dessus pourrait être difficilement transformés sous cette forme :

```
<Replique nom="Marine" texte="Mais si tu ne te souviens pas de l'endroit où tu as placé
ton PDA, comment allons nous pouvoir retrouver notre chemin, car nous ne pouvons pas
utiliser ton application de géolocalisation. Ce n'est pas très malin de ta part."/>

<Replique nom="Luna" texte="Que tu es bête ! Je n'ai pas oublié de prendre mon GPS qui
est tout de même plus pratique que mon PDA. De plus, le son du JerryJerry est de bien
meilleure qualité que mon Achetéssé..."/>

<Replique nom="Marine" texte="Oui, tu as raison."/>
```

La saisie et la lecture sont moins évidentes lorsque vous utilisez des balises avec des informations contenant de nombreux signes. Par ailleurs, si vous deviez utiliser des

guillemets dans la valeur d'un attribut, il faudrait les coder alors que c'est inutile entre les balises ouvrante ou fermante d'un nœud.

```
<Region nomhabitants="Les Alsaciens et les Alsaciennes">"Alsace"</Region>
```

Le mot `Alsace` correspond à la valeur du nœud. Ce dernier peut tout à fait contenir des guillemets. En revanche, le code ci-dessous génèrera une erreur.

```
<Region nomhabitants="Les "Alsaciens" et "Alsaciennes"">Alsace</Region>
```

Nous essayons de placer les termes `Alsaciens` et `Alsaciennes` entre guillemets, mais malheureusement, cela est impossible car il faut utiliser ces signes pour définir la valeur d'un attribut. Nous devons donc utiliser le code `&quot;`

```
<Region nomhabitants="Les &quot;Alsaciens&quot; et &quot;Alsaciennes&quot;">
➥Alsace</Region>
```

Le tableau 13-1 présente les codes des caractères spéciaux que vous devez utiliser dans les valeurs d'attributs :

Tableau 13-1 Les codes des caractères spéciaux

Signe	Code HTML
"	&quot;
<	&lt;
>	&gt;
&	&amp;

En conclusion

L'usage des attributs s'avère très utile et très pratique lorsqu'il faut associer quelques informations à un nœud sans être pour autant obligé de créer des nœuds enfants.

Exploiter une arborescence XML dans une animation

L'exploitation de l'arborescence d'un fichier XML débute par la procédure de chargement des données qu'il contient. Nous allons donc commencer par apprendre à placer le contenu d'un fichier XML dans la mémoire vive de l'ordinateur, afin de pouvoir parcourir ses différentes branches.

Chargement d'un document XML

Fichiers de référence : `Chapitre13/xml.fla` et `Chapitre13/franceregions.xml`

Nous allons devoir utiliser trois variables pour charger puis stocker les lignes contenues dans le fichier XML :

```
var fichierXML:XML;
var chargeurDonnees:URLLoader = new URLLoader();
var adresseFichierXML:URLRequest = new URLRequest("franceregions.xml");
```

Vous noterez que nous faisons appel à la classe `URLRequest()` afin d'obtenir une instance du même type, car la méthode `load()` de la classe `URLLoader()` s'attend à un paramètre de ce type. Par ailleurs, nous déclarons en début de script une variable intitulée `fichierXML`, mais nous l'initialiserons uniquement lorsque les données seront chargées.

Attention

Contrairement à l'AS1/AS2, l'instance de la classe XML que nous avons déclarée et que nous allons initialiser ne va pas faire appel à la méthode `load()` qui n'existe plus. La classe `XML()` n'est plus du tout la même que celle que nous utilisions en AS1/AS2. Si vous souhaitiez retravailler avec cette dernière, il faudrait instancier la classe `XMLDocument()`. Bien sûr, nous vous déconseillons de l'utiliser car elle ne présente pas les mêmes avantages et facilités d'exploitation d'une arborescence XML que ceux qu'offre la nouvelle classe de l'AS3.

Avant de demander le chargement des données, nous définissons la fonction de rappel qui va être appelée lorsque l'événement `Event.COMPLETE` sera effectif (lorsque le chargement des données XML est terminé et constaté).

```
function onComplete(event:Event):void {
  fichierXML = new XML(chargeurDonnees.data);
  zoneAffichage.text = fichierXML;
}
```

Quel rôle joue le contenu de la fonction `onComplete` ? Vous noterez que la variable que nous avions déclarée dans la première ligne du script est enfin initialisée, afin que nous puissions l'utiliser pour lire le contenu du fichier XML. Mais pour cela, nous avons dû faire appel à la propriété `data` de la classe `URLLoader`, ce qui permet d'accéder aux données chargées.

Le nom de la fonction de rappel

Dans cet exemple, le nom de la fonction est `onComplete`, mais nous aurions pu en choisir un autre. Vous devez simplement retenir que le terme ne doit pas contenir de caractères accentués ou spéciaux, ni d'espace. Il ne doit pas non plus commencer par une majuscule. Si vous préférez choisir un nom français, c'est donc tout à fait possible, comme : `function chargementTermine (event:Event):void`

Maintenant que nous avons tout déclaré et initialisé, nous faisons appel à la méthode `load()` de la classe `URLLoader()` afin que les données contenues dans le fichier `franceregions.xml` soient chargées. Nous lançons ensuite l'écouteur de fin de chargement.

```
chargeurDonnees.load(adresseFichierXML);
chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);
```

Pour information

Un certain nombre de développeurs prennent et ont pris l'habitude de déclarer les fonctions qui s'exécutent à la fin d'un chargement de données avant de faire appel à la méthode dédiée (`load()` dans le cas du XML). Mais tant que les débits Internet seront plus lents que la vitesse d'exécution d'une ligne d'instruction par un microprocesseur, cela n'a pas d'importance.

Le script complet qui vous permet de charger un document XML est à présent terminé ; vous devriez obtenir les lignes d'instructions suivantes :

```
var fichierXML:XML;
var chargeurDonnees:URLLoader = new URLLoader();
var adresseFichierXML:URLRequest = new URLRequest("franceregions.xml");

function onComplete(event:Event):void {
  fichierXML = new XML(chargeurDonnees.data);
  zoneAffichage.text = fichierXML;
}
chargeurDonnees.load(adresseFichierXML);
chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);
```

Si nous n'avons pas été assez précis dans nos explications, précisons une dernière fois que c'est uniquement à partir du moment où le chargement du contenu du fichier XML est terminé que vous pouvez y faire référence. De ce fait, si vous souhaitez effectuer la lecture d'un nœud ou d'un attribut de cette arborescence, vous devez placer le code dans la fonction `onComplete` de notre exemple, qui est utilisée par l'écouteur surveillant l'événement `Event.COMPLETE`.

Script dans un fichier AS

Fichiers de référence : `Chapitre13/xmlas.fla`, `Chapitre13/xml.as` et `Chapitre13/franceregions.xml`

Le script de chargement d'un fichier XML à partir d'un fichier externe d'extension `.as` présente une structure bien évidemment différente propre à la POO, mais la procédure reste la même.

```
package {

  import flash.display.Sprite;
  import flash.net.URLLoader;
  import flash.net.URLRequest;
  import flash.events.Event;
  import flash.text.TextField;

  public class xml extends Sprite {

    private var fichierXML:XML;
    private var chargeurDonnees:URLLoader = new URLLoader();
    private var adresseFichierXML:URLRequest = new URLRequest("franceregions.xml");

    public function xml () {
      chargeurDonnees.load(adresseFichierXML);
      chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);
    }

    private function onComplete(event:Event):void {
      fichierXML = new XML(chargeurDonnees.data);
      zoneAffichage.text = fichierXML;
    }
  }
}
```

Lire un nœud

Fichiers de référence : `Chapitre13/xml2.fla` et `Chapitre13/franceregions3.xml`

L'ActionScript 3 présente l'avantage de posséder une classe XML très performante. Il est alors très simple de lire une information contenue dans une arborescence.

Pour commencer cette découverte du XML, nous avons utilisé le fichier le plus simple (`franceregions2.xml`) que nous proposons en téléchargement. Voici la ligne d'instruction la plus élémentaire qui permet de lire un nœud.

```
zoneAffichage.text = fichierXML.Region[2];
```

Dans cet exemple, `zoneAffichage` est le nom d'occurrence d'un texte dynamique sur la scène. Nous devons donc utiliser la propriété `text` pour pouvoir stocker et afficher une information. Nous obtenons le résultat suivant :

```
Auvergne
```

À savoir

Si vous ne maîtrisez pas encore les techniques de manipulation des tableaux, consultez le chapitre 7 avant de poursuivre votre lecture.

Comme vous l'aurez constaté, la lecture d'un nœud est assez simple : il suffit de faire référence au nom d'instance de la classe XML que nous avions préalablement créée (`fichierXML`), puis d'utiliser le nom de la balise à atteindre.

Notre exemple contient des nœuds enfants portant le même nom (`Region`). De ce fait, nous sommes obligés d'utiliser des crochets pour spécifier le numéro de nœud. Rappelons que le premier de la liste porte le numéro d'index 0. Pour afficher le mot Alsace, il aurait justement fallu utiliser cet index, comme ceci :

```
zoneAffichage.text = fichierXML.Region[0];
```

Voici un exemple de script complet sur une image-clé :

```
var fichierXML:XML;
var chargeurDonnees:URLLoader = new URLLoader();
var adresseFichierXML:URLRequest = new URLRequest("franceregions3.xml");

function onComplete(event:Event):void {

  fichierXML = new XML(chargeurDonnees.data);
  btNoeud.addEventListener(MouseEvent.CLICK,lireNoeud);
}

chargeurDonnees.load(adresseFichierXML);
chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);

function lireNoeud(evt:MouseEvent):void {
  zoneAffichage.text = fichierXML.Region[2];
}
```

Commentaires du script :

`btNoeud` est le nom d'une instance de clip ou de bouton sur la scène. `zoneAffichage` est le nom d'un texte dynamique sur la scène.

Script dans un fichier AS

Fichiers de référence : `Chapitre13/xml2as.fla`, `Chapitre13/xml2.as` et `Chapitre13/franceregions3.xml`

Nous sommes partis du fichier `xml2.fla` que nous avons réenregistré en `xml2as.fla`. Nous avons précisé le nom de la classe du document : `xml2.as`. Voici son contenu.

```
package {

  import flash.display.Sprite;
  import flash.display.MovieClip;
  import flash.text.TextField;
  import flash.net.URLLoader;
  import flash.net.URLRequest;
  import flash.events.Event;
  import flash.events.MouseEvent;

  public class xml2 extends Sprite {


    private var fichierXML:XML;
    private var chargeurDonnees:URLLoader = new URLLoader();
    private var adresseFichierXML:URLRequest = new URLRequest("franceregions3.xml");

    public function xml2() {

      chargeurDonnees.load(adresseFichierXML);
      chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);

    }

    private function onComplete(evt:Event):void {
      fichierXML = new XML(chargeurDonnees.data);
      btNoeud.addEventListener(MouseEvent.CLICK,lireNoeud);
      btAttribut.addEventListener(MouseEvent.CLICK,lireAttribut);
    }

    private function lireNoeud(evt:MouseEvent):void {
      zoneAffichage.text = fichierXML.Region[2];
    }

    private function lireAttribut(evt:MouseEvent):void {
      zoneAffichage.text = fichierXML.Region[2].@nomhabitants;
    }

  }
}
```

Si vous effectuez vos premiers pas en programmation orientée objet, soyez très vigilant sur les oublis éventuels d'imports de classes et de packages.

Nous avons commencé notre approche du XML avec un premier exemple de code contenant des crochets, car il est très fréquent de travailler avec des documents possédant de nombreux nœuds avec le même nom de balise. Vous rencontrerez également des cas où plusieurs nœuds de même niveau ne portent pas le même nom de balise, comme dans l'exemple ci-dessous.

Les exemples qui vont suivre…

…ne possèdent pas de fichiers correspondants dans les exemples à télécharger.

```
<Famille>
  <David>3 février 1970</David>
  <Marine>12 novembre 1978</Marine>
  <Luna>14 octobre 2000</Luna>
  <Marjorie>15 mai 1975</Marjorie>
</Famille>
```

Nous n'avons plus besoin d'utiliser les crochets pour atteindre la valeur d'un nœud, car ce dernier est le seul à porter son nom de balise. En utilisant la ligne d'instruction ci-dessous, nous obtenons l'affichage `14 octobre 2000` sur la scène.

```
zoneAffichage.text = fichierXML.Luna;
```

Si ce nœud possède à son tour des nœuds enfants, il suffit alors d'y faire référence comme ceci :

```
zoneAffichage.text = fichierXML.Luna.DateNaissance;
```

Fichier XML :

```
<Famille>

<David>
  <DateNaissance>3 février 1970</DateNaissance>
  <LieuNaissance>Paris XIIe</LieuNaissance>
</David>

<Marine>
  <DateNaissance>12 novembre 1978</DateNaissance>
  <LieuNaissance>Paris Xe</LieuNaissance>
</Marine>

<Luna>
  <DateNaissance>14 octobre 2000</DateNaissance>
  <LieuNaissance>Rambouillet (78)</LieuNaissance>
</Luna>

<Marjorie>
  <DateNaissance>15 mai 1975</DateNaissance>
  <LieuNaissance>Nancy (54)</LieuNaissance>
</Marjorie>

</Famille>
```

Dans la pratique, les nœuds enfants d'un même niveau qui ne portent pas le même nom de balise se trouvent très souvent en fin de branche, mais rarement au premier niveau (sauf pour les fichiers XML de configuration).

Pour finir, en imaginant que plusieurs nœuds du même niveau existent dans un nœud enfant, nous devons alors procéder comme nous l'avions fait initialement. Pour obtenir `Boire` avec le fichier XML ci-dessous, nous devons écrire la ligne suivante :

```
zoneAffichage.text = fichierXML.Luna.Passion[1];
```

Ficher XML :

```
<Famille>

<David>
  <Passion>VTT</Passion>
  <Passion>Gastronomie</Passion>
  <Passion>Films d'auteurs</Passion>
</David>

<Marine>
  <Passion>Dessiner</Passion>
  <Passion>Chanter</Passion>
  <Passion>Danser</Passion>
</Marine>

<Luna>
  <Passion>Manger</Passion>
  <Passion>Boire</Passion>
  <Passion>Dormir</Passion>
</Luna>

<Marjorie>
  <Passion>Cuisiner</Passion>
  <Passion>Dessins sur supports variés</Passion>
  <Passion>Les enfants</Passion>
</Marjorie>

</Famille>
```

Lire un attribut

Fichiers de référence : `Chapitre13/xml2.fla` et `Chapitre13/franceregions3.xml`

La technique de lecture d'un attribut est très simple car elle s'apparente étrangement à celle d'un nœud, à un détail près, le caractère arobase (@).

> **Remarque**
> Un attribut ne peut pas contenir de nœud enfant, ni même d'attribut imbriqué.

Si nous utilisons la ligne d'instruction ci-dessous, le texte `Auvergnats & Auvergnates` s'affiche sur la scène.

```
zoneAffichage.text = fichierXML.Region[2].@nomhabitants;
```

Pour les développeurs néophytes en AS3, cette syntaxe est assez troublante car seul le caractère @ permet de faire la différence entre un nœud et un attribut. Il est vrai que si nous omettons de saisir l'arobase, nous faisons alors référence au nom d'une balise, c'est-à-dire au nom d'un nœud et non plus un attribut. Le risque d'erreur est donc important lors de vos premiers développements.

Voici un exemple de script complet :

```
var fichierXML:XML;
var chargeurDonnees:URLLoader = new URLLoader();
var adresseFichierXML:URLRequest = new function onComplete(event:Event):void {
  fichierXML = new XML(chargeurDonnees.data);
  btAttribut.addEventListener(MouseEvent.CLICK,lireAttribut);
}
chargeurDonnees.load(adresseFichierXML);
chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);

function lireAttribut (evt:Event):void {
  zoneAffichage.text = fichierXML.Region[2].@nomhabitants;
}
```

Commentaires du script :

`btAttribut` est le nom d'une instance de clip ou de bouton sur la scène. `zoneAffichage` est le nom d'un texte dynamique sur la scène.

Lire plusieurs attributs

Pour pouvoir lire tous les attributs d'une balise, vous pouvez utiliser la syntaxe suivante :

```
fichierXML.Region[2].attribute("*").toXMLString();
```

ou celle-ci :

```
fichierXML.Region[2].attributes().toXMLString();
```

Important

La fonction `toXMLString()` permet en temps normal d'afficher le contenu d'un nœud encadré de ses balises. Cela sous-entend donc que cette fonction ne peut fonctionner qu'avec des nœuds et non des attributs. Dans ce cas, la fonction essaye de trouver une structure XML inexistante (car tous les attributs se trouvent dans le même nœud et non dans différents nœuds). Ne la trouvant pas, chaque propriété est renvoyée à la ligne.

Afin que cet exemple fonctionne, il faudrait changer le contenu du fichier XML fourni, comme ceci :

```
<France>
<Region nomhabitants="Alsaciens" annee="1212">Alsace</Region>
<Region nomhabitants="Aquitains" annee="1345">Aquitaine</Region>
<Region nomhabitants="Auvergnats" annee="1190>Auvergne</Region>
…
</France>
```

Vous récupéreriez alors les valeurs `Auvergnats` et `1190`.

Importance de la fonction toXMLString()

En AS3, l'utilisation du XML dans une animation se fait très simplement, en comparaison avec l'AS1/AS2.

Lorsque vous tentez de lire un nœud précis, vous obtenez la valeur contenue entre les balises et non pas la valeur accompagnée des balises, comme c'était le cas en AS1/AS2. De ce fait, si vous étiez auparavant habitué à lire le contenu d'un nœud avec un `firstChild` ou un `childNode[0]`, vous adopterez vite cette technique plus pratique.

En AS3, vous pointez directement un nœud. Seule sa valeur vous est renvoyée, mais il s'avère que, dans certains cas, vous aurez besoin de récupérer et/ou connaître le nom des balises qui entourent le texte récupéré. Vous utiliserez alors la fonction `toXMLString()`.

Fichier XML :

```
<Famille>

  <David>
    <Passion>VTT</Passion>
    <Passion>Gastronomie</Passion>
    <Passion>Films d'auteurs</Passion>
  </David>

</Famille>
```

Voici le code nécessaire pour afficher `Gastronomie` dans un texte dynamique intitulé `zoneAffichage` sur la scène :

```
zoneAffichage.text=fichierXML.David.Passion[1];
```

Voici le code nécessaire pour afficher `<Passion>Gastronomie</Passion>` dans un texte dynamique intitulé `zoneAffichage` sur la scène :

```
zoneAffichage.text=fichierXML.David.Passion[1].toXMLString();
```

toXMLString() avec des attributs de balise

Par définition, si la fonction `toXMLString()` renvoie la structure XML d'un nœud, que se passe-t-il avec un ou plusieurs attributs ?

```
<Famille>

  <David nom="TARDIVEAU" surnom="Yazo">
    <Passion>VTT</Passion>
    <Passion>Gastronomie</Passion>
    <Passion>Films d'auteurs</Passion>
  </David>

</Famille>
```

En partant de l'exemple ci-dessus, si nous n'avions qu'un seul attribut, cela n'aurait aucune incidence. En revanche, avec deux ou plusieurs attributs, dans la mesure où ils ne se trouvent pas entre des balises, chaque attribut est affiché à la ligne.

```
zoneAffichage.text = fichierXML.David.attributes().toXMLString();
```

Voici le résultat :

```
TARDIVEAU
Yazo
```

Effectuer une recherche dans une arborescence XML

La recherche d'un attribut ou d'un nœud peut être effectuée de différentes façons, dans la mesure où le résultat d'une requête n'est très souvent qu'une étape préliminaire pour trouver une information encore plus précise.

Partons de l'exemple ci-dessous pour effectuer deux premières recherches :

Fichiers de référence : `Chapitre13/xml3.fla` et `Chapitre13/cinema.xml`

```
<Cinema>

<Film nom="Le Papillon">
  <Realisateur>Philippe MUYL</Realisateur>
  <ActeurPrincipal sexe="M">Michel SERRAULT</ActeurPrincipal>
  <Annee>2002</Annee>
  <Genre>Comédie dramatique</Genre>
  <Pays>France</Pays>
</Film>

<Film nom="Erin Brockovich">
  <Realisateur>Steven SODERBERGH</Realisateur>
  <ActeurPrincipal sexe="F">Julia ROBERTS</ActeurPrincipal>
  <Annee>2000</Annee>
  <Genre>Dramatique</Genre>
  <Pays>US</Pays>
</Film>

<Film nom="The Bourne Ultimatum">
  <Realisateur>Paul GREENGRASS</Realisateur>
  <ActeurPrincipal sexe="M">Matt DAMON</ActeurPrincipal>
  <Annee>2007</Annee>
  <Genre>Action</Genre>
  <Pays>US</Pays>
</Film>

<Film nom="DOGMA">
  <Realisateur>Kevin SMITH</Realisateur>
  <ActeurPrincipal sexe="M">Matt DAMON</ActeurPrincipal>
  <Annee>1999</Annee>
  <Genre>Fantastique</Genre>
  <Pays>US</Pays>
</Film>
```

```
<Film nom="Les enfants du marais">
  <Realisateur>Jean BECKER</Realisateur>
  <ActeurPrincipal sexe="M">Michel SERRAULT</ActeurPrincipal>
  <Annee>1999</Annee>
  <Genre>Comédie dramatique</Genre>
  <Pays>France</Pays>
</Film>

<Film nom="Un crime au paradis">
  <Realisateur>Jean BECKER</Realisateur>
  <ActeurPrincipal sexe="M">Jacques VILLERET</ActeurPrincipal>
  <Annee>2001</Annee>
  <Genre>Comédie dramatique</Genre>
  <Pays>France</Pays>
</Film>

<Film nom="Mona Lisa Smile">
  <Realisateur>Mike NEWELL</Realisateur>
  <ActeurPrincipal sexe="F">Julia ROBERTS</ActeurPrincipal>
  <Annee>2003</Annee>
  <Genre>Comédie dramatique</Genre>
  <Pays>US</Pays>
</Film>

</Cinema>
```

Recherche par nœud

Pour afficher `Paul GREENGRASS` sur la scène, nous avons commencé par rechercher les films crées en 2007.

```
zoneAffichage.text = fichierXML.Film.(Annee==2007);
```

Puis, nous avons spécifié le nom du nœud a récupérer.

```
zoneAffichage.text = fichierXML.Film.(Annee==2007).Realisateur;
```

Comme vous pouvez le constater, la technique de recherche par nœud est très simple. Vous commencez par saisir le début d'une ligne d'instruction classique, comme si vous vous apprêtiez à indiquer le chemin qui mène à un nœud, mais la fin de la ligne d'instruction se termine par une instruction entre parenthèses :

```
(Annee==2007);
```

Vous précisez le nom d'une balise, suivi d'un opérateur de comparaison et de la valeur à évaluer.

Attention

Vous devez faire précéder votre condition de recherche, placée entre parenthèses, par un point. La ligne ci-dessous renverrait donc une erreur.

```
zoneAffichage.text = fichierXML.Film(Annee==2007);
```

Lorsque vous réussissez à trouver un nœud, vous pouvez alors le parcourir en y ajoutant des noms de nœuds ou d'attributs comme dans les exemples ci-dessous.

Cette ligne permet d'afficher `Paul GREENGRASS`.

```
zoneAffichage.text = fichierXML.Film.(Annee==2007).Realisateur;
```

Cette ligne permet d'afficher `Action`.

```
zoneAffichage.text = fichierXML.Film.(Annee==2007).Genre;
```

Cette ligne permet d'afficher `France`.

```
zoneAffichage.text = fichierXML.Film.(Annee==2007).Pays;
```

Voici un exemple de script complet :

Rappel

Les fichiers de référence de cette animation sont `Chapitre13/xml3.fla` et `Chapitre13/cinema.xml`.

```
var fichierXML:XML;
var chargeurDonnees:URLLoader = new URLLoader();
var adresseFichierXML:URLRequest = new URLRequest("cinema.xml");

function onComplete(event:Event):void {
  fichierXML = new XML(chargeurDonnees.data);
  btNoeud.addEventListener(MouseEvent.CLICK,lireNoeud);
}

chargeurDonnees.load(adresseFichierXML);
chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);

function lireNoeud(evt:Event):void {
  trace(fichierXML.Film.(Annee==2007));
  zoneAffichage.text = fichierXML.Film.(Annee==2007).Realisateur;
}
```

Commentaires du script :

`btNoeud` est le nom d'une instance de clip ou de bouton sur la scène. `zoneAffichage` est le nom d'un texte dynamique sur la scène. La fonction `trace()` ne sert qu'à afficher l'information qu'elle contient entre parenthèses dans la fenêtre Sortie de l'IDE de Flash.

Recherche par attribut

Pour afficher `Michel SERRAULT` sur la scène, nous avons commencé par rechercher le film dont le nom est `Le Papillon` en utilisant l'attribut `nom` de la balise `Film`.

```
zoneAffichage.text=fichierXML.Film.(@nom=="Le Papillon");
```

Puis, nous avons spécifié le nom du nœud à récupérer.

```
zoneAffichage.text=fichierXML.Film.(@nom=="Le Papillon").ActeurPrincipal;
```

Comme vous pouvez le constater, la technique de recherche par attribut est aussi simple que celle par nœud. Vous commencez par saisir le début d'une ligne d'instruction classique, comme si vous vous apprêtiez à indiquer le chemin qui mène à un nœud, mais la fin de la ligne d'instruction se termine par une instruction entre parenthèses :

```
(@nom=="Le Papillon");
```

Puisque nous évaluons la valeur d'un attribut, il est impératif d'utiliser le signe arobase. Vous précisez ensuite l'opérateur de comparaison et la valeur à évaluer.

Attention

Vous devez faire précéder votre condition de recherche, placée entre parenthèses, par un point. La ligne ci-dessous renverrait donc une erreur.

```
zoneAffichage.text = fichierXML.Film(@nom=="Le Papillon");
```

Lorsque vous parvenez à trouver un nœud, vous pouvez alors le parcourir en y ajoutant des noms de nœuds ou d'attributs comme dans les exemples ci-dessous.

Cette ligne permet d'afficher `Michel SERRAULT`.

```
zoneAffichage.text = fichierXML.Film.(@nom=="Le Papillon").ActeurPrincipal;
```

Cette ligne permet d'afficher `Philippe MUYL`.

```
zoneAffichage.text = fichierXML.Film.(@nom=="Le Papillon").Realisateur;
```

Cette ligne permet d'afficher la lettre `M` pour Masculin.

```
zoneAffichage.text = fichierXML.Film.(@nom=="Le Papillon").ActeurPrincipal.@sexe;
```

Cette ligne permet d'afficher `2002`.

```
zoneAffichage.text = fichierXML.Film.(@nom=="Le Papillon").Annee;
```

Voici un exemple de script complet :

Rappel

Les fichiers de référence de cette animation sont `Chapitre13/xml3.fla` et `Chapitre13/cinema.xml`.

```
var fichierXML:XML;
var chargeurDonnees:URLLoader = new URLLoader();
var adresseFichierXML:URLRequest = new URLRequest("cinema.xml");

function onComplete(event:Event):void {
  fichierXML = new XML(chargeurDonnees.data);
  btAttribut.addEventListener(MouseEvent.CLICK,lireAttribut);
}

chargeurDonnees.load(adresseFichierXML);
chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);
```

```
function lireAttribut(evt:Event):void {
  trace(fichierXML.Film.(@nom=="Le Papillon").ActeurPrincipal.@sexe);
  zoneAffichage.text=fichierXML.Film.(@nom=="Le Papillon").ActeurPrincipal;
}
```

Commentaires du script :

`btAttribut` est le nom d'une instance de clip ou de bouton sur la scène. `zoneAffichage` est le nom d'un texte dynamique sur la scène. La fonction `trace()` ne sert qu'à afficher l'information qu'elle contient entre parenthèses dans la fenêtre Sortie de l'IDE de Flash.

Les opérateurs de recherche

Dans les exemples précédents, nous avons recherché des branches précises. Un seul film a donc été trouvé à chaque fois, mais nous aurions également pu obtenir plusieurs résultats. Ainsi, la ligne d'instruction ci-dessous permettrait d'afficher trois films.

```
fichierXML.Film.(Pays=="France");
```

L'opérateur de comparaison == ne renvoie donc pas obligatoirement une seule valeur. Mais quels sont les autres opérateurs que nous pouvons utiliser ?

Différent de…

La requête suivante permet d'obtenir tous les nœuds (de niveau 1) du document qui ne correspondent pas à des films américains.

```
fichierXML.Film.(Pays!="US");
```

L'opérateur `!=` permet d'évaluer une différence. Dans notre fichier XML, des films français et américains sont proposés, mais seules les productions venant de France sont récupérées.

Supérieur/inférieur et supérieur ou égal/inférieur ou égal

Ces opérateurs s'avèrent très pratiques pour récupérer des nœuds en fonction d'une limite. Ainsi dans l'exemple ci-dessous, nous allons pouvoir connaître tous les films sortis depuis 2002.

```
fichierXML.Film.(Annee>=2002);
```

Opérateur	Description
==	Recherche d'égalité
!=	Recherche de différence
>	Recherche de supériorité
>=	Recherche de supériorité ou d'égalité
<	Recherche d'infériorité
<=	Recherche d'infériorité ou d'égalité

Voici un exemple de script complet :

Rappel

Les fichiers de référence de cette animation sont `Chapitre13/xml4.fla` et `Chapitre13/cinema.xml`.

```
var fichierXML:XML;
var chargeurDonnees:URLLoader = new URLLoader();
var adresseFichierXML:URLRequest = new URLRequest("cinema.xml");

function onComplete(event:Event):void {
  fichierXML = new XML(chargeurDonnees.data);
  btDifference.addEventListener(MouseEvent.CLICK,rechercheDifference);
  btRecherchePosterieure.addEventListener(MouseEvent.CLICK,recherchePosterieure);
}

chargeurDonnees.load(adresseFichierXML);
chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);

function rechercheDifference(evt:Event):void {
  zoneAffichage.text = fichierXML.Film.(Pays!="US").Realisateur;
}

function recherchePosterieure(evt:Event):void {
  zoneAffichage.text=fichierXML.Film.(Annee>=2002).ActeurPrincipal;
}
```

Commentaires du script :

`btDifference` et `btRecherchePosterieure` sont les noms d'instances de clip ou de bouton sur la scène. `zoneAffichage` est le nom d'un texte dynamique sur la scène.

Croiser les recherches ou recherches multicritères

Vous l'aurez peut-être compris, tout ce qui se trouve entre les parenthèses pour effectuer une recherche relève d'une évaluation. Nous pouvons donc combiner plusieurs requêtes avec les opérateurs `||` et `&&`. Ainsi, l'exemple ci-dessous permet de trouver très précisément tous les films de Jean Becker qui sont sortis depuis 2000.

```
zoneAffichage.text = fichierXML.Film.(Realisateur=="Jean BECKER" && Annee>=2000);
```

Cet autre exemple permet de trouver très précisément tous les films américains dont l'acteur principal est une femme.

```
fichierXML.Film.(Pays=="US" && ActeurPrincipal.@sexe=="F");
```

Remarque

Utilisez l'intégralité du script précédent pour replacer les deux lignes d'instructions dans leur contexte.

Rechercher plusieurs nœuds ou attributs à des niveaux différents

Vous pourrez récupérer la valeur de plusieurs nœuds ou attributs de nœuds à des niveaux différents, en ne précisant aucune entrée :

```
fichierXML.Film.Realisateur;
fichierXML.Region.Departement.@prefecture;
```

Ainsi, en partant des fichiers `cinema.xml` et `france3.xml`, vous pourriez trouver toutes les préfectures de France ou tous les noms des réalisateurs des films proposés dans le document (testez, à ce propos, le fichier `xml6.fla`).

Les deux lignes d'instructions ci-dessous font d'ailleurs appel à la classe `XMLList()` pour stocker le résultat des requêtes.

```
var resultat:XMLList=new XMLList(fichierXML.Film.Realisateur);
  zoneAffichage.text = resultat[2];

var resultat:XMLList=new XMLList(fichierXML.Region.Departement.@prefecture);
  trace(resultat[12]);
```

Remarque

Utilisez l'intégralité du script précédent pour replacer les deux lignes d'instructions dans leur contexte.

Recherche de parent

Lorsque vous effectuez une recherche, vous pouvez l'affiner par la suite en indiquant le nom d'un nœud enfant ou d'un attribut. Vous scrutez donc les informations descendantes ou en aval. Dans notre cas, nous devons connaître le nœud parent pour pouvoir lire d'autres informations qu'il contient.

Fichiers de référence : `Chapitre13/xml6.fla` et `Chapitre13/france3.xml`

Attention

Vous ne pouvez pas rechercher un attribut dans une balise autofermante ou une balise vide.

Dans le fichier XML de référence que nous allons utiliser pour exemple, vous pouvez constater que l'arborescence est plus complexe que celles avec lesquelles nous avons travaillé jusqu'à présent.

Elle présente par ailleurs des balises `Departement` vides alors que nous aurions pu utiliser des balises autofermantes. Précisons simplement que ce type de balise ne peut faire l'objet d'une recherche.

Nous avons choisi un tel fichier, avec autant de nœuds enfants et d'attributs, pour démontrer que la recherche d'un attribut s'avère indispensable pour pouvoir, dans ce cas, retrouver une information.

Nous aimerions déterminer la région de France et le numéro du département où se trouve Toulon. Avant de nous pencher sur le code, voici la structure du nœud avec lequel nous allons travailler.

```
<Region nom="Provence-Alpes-Côte d'azur">
   <Departement nom="Alpes de Hautes-Provence" numero="4" prefecture="Digne">
   ➥</Departement>
   <Departement nom="Hautes-Alpes" numero="5" prefecture="Gap"></Departement>
   <Departement nom="Alpes-Maritimes" numero="6" prefecture="Nice"></Departement>
   <Departement nom="Bouches-du-Rhône" numero="13" prefecture="Marseille">
   ➥</Departement>
   <Departement nom="Var" numero="83" prefecture="Toulon"></Departement>
   <Departement nom="Vaucluse" numero="84" prefecture="Avignon"></Departement>
   <Population>
      <Annee numero="1990">4 257 907</Annee>
      <Annee numero="1999">4 506 151</Annee>
      <Annee numero="2005">4 750 947</Annee>
      <Annee numero="2006">4 780 989</Annee>
   </Population>
</Region>
```

Remarque

Rappelons qu'il s'agit de l'un des 22 nœuds contenus dans le fichier `france3.xml`. et non de l'arborescence complète d'un fichier XML.

La rédaction de la ligne d'instruction qui permet de trouver les valeurs `Provence-Alpes-Côte d'azur` et `83` nécessite une petite réflexion.

Nous pouvons constater que la ligne du fichier XML qui contient l'attribut `prefecture` avec la valeur `Toulon`, est vide. Nous ne pouvons donc pas récupérer d'information descendante. Seuls les attributs au même niveau que celui de la `prefecture` et les informations relatives aux nœuds ascendants (donc le nœud parent) peuvent être récupérées.

L'utilisation de la fonction `parent()` permet de lire les données contenues dans le nœud parent du chemin spécifié. Ainsi, dans l'exemple suivant, nous pouvons récupérer le nœud XML que nous vous avons présenté ci-dessus :

```
fichierXML.Region.Departement.(@prefecture == "Toulon").parent();
```

Rappelons que nous cherchons à récupérer le nom de la région dans laquelle se trouve la ville de Toulon. Cette information (`Provence-Alpes-Côte d'Azur`) figure en tant que valeur de l'attribut nom de la balise `Region` (nœud parent que nous venons de récupérer). Il ne reste plus qu'à terminer notre requête en ajoutant l'attribut `@nom`.

```
fichierXML.Region.Departement.(@prefecture == "Toulon").parent().@nom;
```

Nous risquerions d'avoir à nouveau besoin d'exploiter le nœud parent, il serait donc plus judicieux de stocker le résultat de la première requête dans une variable.

```
var regionTrouvee:XML = fichierXML.Region.Departement.(@prefecture == "Toulon").parent();
  zoneAffichage.text = regionTrouvee.@nom;
```

Information

Précisons que nous pourrions remonter de plusieurs niveaux pour obtenir les informations de nœuds parents ascendants. Dans l'exemple ci-dessous, nous récupérons l'intégralité du contenu du document XML (fichier `france3.xml`).

```
fichierXML.Region.Departement.(@prefecture == "Toulon").parent().parent();
```

Pour rechercher à présent un attribut figurant au même niveau que celui de la `prefecture`, nous devons compléter la requête avec l'attribut `numero` pour trouver le numéro de département `83`.

```
zoneAffichage.text=fichierXML.Region.Departement.(@prefecture == "Toulon").@numero;
```

Nous n'utilisons pas la fonction `parent()` car nous devons rester au même niveau que le résultat de la recherche.

Parcourir toute une arborescence

Si vous avez lu toutes les explications précédentes, vous pouvez comprendre à quel point le besoin de parcourir une arborescence est moins utile que cela pouvait l'être en AS1/AS2, pour ceux qui connaissent ces versions du langage. Les fonctions de recherche/filtre évitent en effet de faire appel à une boucle `for()` ou `for each()`.

Cependant, dans certains cas, vous ne pourrez pas faire autrement : il faudra écrire une boucle sur un document XML.

En partant de l'exemple ci-dessous, nous allons montrer que la boucle `for each()` permet d'obtenir une mise en forme particulière.

Fichiers de référence : `Chapitre13/xml8.fla` et `Chapitre13/cinema.xml`

Remarque

Référez-vous au chapitre 10 dédié à l'apprentissage des itérations pour en comprendre le fonctionnement.

Commencez par ouvrir le fichier `cinema.xml` et parcourez les lignes qu'il contient pour prendre connaissance de sa structure (il s'agit du fichier XML qui a été présenté dans la section Effectuer une recherche dans une arborescence XML).

Nous souhaiterions obtenir l'affichage du texte sous les deux formes suivantes :

Exemple 1 :

Film : Le Papillon

Film : Erin Brockovich

Film : The Bourne Ultimatum

Film : DOGMA

Film : Les enfants du marais

Film : Un crime au paradis

Film : Mona Lisa Smile

Exemple 2 :

2002 : Le Papillon

2000 : Erin Brockovich

2007 : The Bourne Ultimatum

1999 : DOGMA

1999 : Les enfants du marais

2001 : Un crime au paradis

2003 : Mona Lisa Smile

Dans le premier exemple, nous nous contentons d'ajouter la chaîne de caractères Film : devant le nom d'un film, alors que dans l'exemple 2, nous allons préalablement afficher l'année de sortie de film, qui est une information stockée sous forme d'attribut dans la balise principale de chaque nœud.

Analyse de l'exemple 1

> **Remarque**
>
> Consultez l'exemple complet du script à la fin de l'analyse de l'exemple 2 pour replacer cet extrait de code dans son contexte.

Dans le deuxième exemple (la fonction de rappel `affichageSousForme1`), nous commençons par extraire la liste des noms de films pour la stocker dans une instance de la classe `XMLList`.

```
var listeNomsFilms:XMLList = new XMLList(fichierXML.Film.@nom);
```

Nous vidons ensuite le contenu de notre texte dynamique intitulé `zoneAffichage` qui se trouve sur la scène.

```
zoneAffichage.text="";
```

Pour finir, nous allons effectuer une boucle `for each()` pour parcourir l'instance `listeNomsFilms`.

```
for each (var noms in listeNomsFilms) {
  zoneAffichage.appendText("Film : "+noms+"\n");
}
```

La fonction `appendText()` n'est pas une méthode (ou une fonction) de la classe `XML()` ni même de la classe `XMLList()`. Dans notre cas, elle fait partie des membres de la classe `TextField()`. Attardons-nous donc sur le paramètre `"Film : "+noms+"\n"`.

Cette chaîne de caractères se décompose en trois parties concaténées grâce à l'opérateur +. Puisque nous souhaitons voir s'afficher « Film : » en début de ligne, nous saisissons ce préfixe entre guillemets.

La variable locale `noms` va successivement prendre pour valeur celles qui sont stockées dans l'instance de la classe `XMLList()` grâce à la boucle `for each()`.

Enfin, nous complétons le paramètre de la méthode `appendText()` par la chaîne `\n` qui permet d'effectuer un saut de ligne à chaque itération.

Grâce à cette boucle, nous pouvons donc parcourir toutes les valeurs de l'instance `XMLList()` et obtenir un affichage sous forme de liste.

Analyse de l'exemple 2

Remarque

Consultez le script complet à la fin de l'analyse de cet exemple pour replacer cet extrait de code dans son contexte.

```
function affichagerSousForme2(evt:Event):void {
  zoneAffichage.text="";

  for each (var leNoeud in fichierXML.Film) {
    zoneAffichage.appendText(leNoeud.Annee+" :  "+leNoeud.@nom+"\n");
  }
}
```

Dans le deuxième exemple (la fonction de rappel `affichageSousForme2`), vous constaterez que nous ne faisons pas appel à la classe `XMLList()`. Nous pointons directement chaque ligne du fichier XML en cherchant à lire les propriétés contenues dans les éléments `fichierXML.Film`. Précisons qu'en faisant référence à cet extrait de code, nous pointons vers tous les nœuds.

Essayez d'effectuer un `trace(fichierXML.Film)` et un `trace(fichierXML.Film[1])` ; vous comprendrez pourquoi les paramètres de la boucle `for each()` correspondent à tous les nœuds de l'arborescence.

Voici un exemple de script complet :

```
var fichierXML:XML;
var chargeurDonnees:URLLoader = new URLLoader();
var adresseFichierXML:URLRequest = new URLRequest("cinema.xml");

function onComplete(event:Event):void {

  fichierXML = new XML(chargeurDonnees.data);
  btExemple1.addEventListener(MouseEvent.CLICK,affichageSousForme1);
  btExemple2.addEventListener(MouseEvent.CLICK,affichageSousForme2);
}

chargeurDonnees.load(adresseFichierXML);
chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);
```

```
function affichageSousForme1(evt:Event):void {
  var listeNomsFilms:XMLList = new XMLList(fichierXML.Film.@nom);
  zoneAffichage.text="";

  for each (var noms in listeNomsFilms) {
    zoneAffichage.appendText("Film : "+noms+"\n");
  }
}

function affichageSousForme2(evt:Event):void {
  zoneAffichage.text="";

  for each (var leNoeud in fichierXML.Film) {
    zoneAffichage.appendText(leNoeud.Annee+" :  "+leNoeud.@nom+"\n");
  }
}
```

Pourquoi faire appel à la classe XMLList() ?

L'objectif d'une telle méthode est de démontrer, qu'en passant par la classe `XMLList()`, on peut isoler des données en les stockant dans une instance qui bénéficie, de ce fait, des méthodes et propriétés de la classe dont elle est issue.

Modifier la valeur d'un nœud ou d'un attribut

Fichiers de référence : `Chapitre13/xml7.fla` et `Chapitre13/entreprise.xml`

Une fois encore, la technique est très simple : il suffit de faire référence à la valeur du nœud ou de l'attribut à modifier pour pouvoir modifier sa valeur. Ainsi, dans l'exemple suivant, nous allons changer le nom du service dans lequel travaille `Jacques DURAND`.

```
<Entreprise>

  <Collaborateur nom="DUPUIS" prenom="Marie">
    <Service>Financier</Service>
    <Etage>2</Etage>
  </Collaborateur>

  <Collaborateur nom="MARTIN" prenom="Lucie">
    <Service>Direction du personnel</Service>
    <Etage>3</Etage>
  </Collaborateur>

  <Collaborateur nom="DURAND" prenom="Jacques">
    <Service>Expédition du courier</Service>
    <Etage>RDC</Etage>
  </Collaborateur>

  <Collaborateur nom="DUPONT" prenom="Jean">
    <Service>Direction du personnel</Service>
    <Etage>3</Etage>
  </Collaborateur>

</Entreprise>
```

Ligne d'instruction à exécuter :

```
fichierXML.Collaborateur[2].Service="Recherche et développement";
```

Voici un exemple de script complet :

```
var fichierXML:XML;
var chargeurDonnees:URLLoader = new URLLoader();
var adresseFichierXML:URLRequest = new URLRequest("entreprise.xml");

function onComplete(event:Event):void {

  fichierXML = new XML(chargeurDonnees.data);
  btChangerValeur.addEventListener(MouseEvent.CLICK,changerValeur);
  btLireValeur.addEventListener(MouseEvent.CLICK,lireValeur);
}

chargeurDonnees.load(adresseFichierXML);
chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);

function changerValeur(evt:Event):void {
  zoneAffichage.text = "Ancien service (avant changement) :
  ➥"+fichierXML.Collaborateur[2].Service;
  fichierXML.Collaborateur[2].Service="Recherche et développement";
}

function lireValeur(evt:Event):void {
  zoneAffichage.text = "Nouveau service : "+fichierXML.Collaborateur[2].Service;
}
```

Commentaires du script :

`btChangerValeur` et `btLireValeur` sont les noms d'instances de clip ou de bouton sur la scène. `zoneAffichage` est le nom d'un texte dynamique sur la scène.

Pour changer la valeur d'un attribut, nous devons procéder de façon analogue en remplaçant le nom de nœud par le nom d'attribut, sans oublier d'utiliser le signe @.

```
fichierXML.Collaborateur[2].@nom="TARDIVEAU";
```

Il est important de comprendre que vous ne modifiez que le contenu de l'instance XML, mais en aucun cas celui du fichier que vous aviez préalablement chargé.

Ajouter un nœud

Fichiers de référence : `Chapitre13/xml10.fla` et `Chapitre13/cinema.xml`

Il existe trois méthodes d'ajout d'un nœud à une arborescence XML. En effet, il est possible de faire une insertion avant ou après un nœud spécifique ou bien même après le dernier nœud de l'instance XML. Les méthodes sont les suivantes :

- `insertChildAfter()`
- `insertChildBefore()`
- `appendChild()`

Les paramètres de ces trois méthodes peuvent être très divers. Pour ajouter un nœud, une astuce consiste à copier l'un des nœuds existants pour conserver sa structure. Il ne reste plus alors qu'à modifier les valeurs des différents nœuds enfants et des attributs du nœud copié. Dans l'exemple suivant, nous copions le premier nœud de l'arborescence à la fin de l'instance XML.

```
fichierXML.appendChild(fichierXML.Film[0]);
```

Un seul paramètre est nécessaire : il est inutile de spécifier l'emplacement où le nœud doit être ajouté dans l'arborescence ; la méthode `appendChild()` l'ajoute automatiquement après le dernier nœud. Il faudrait ensuite changer les valeurs des nœuds enfants :

```
fichierXML.Film[2].@nom = "Le père noël est une ordure";
fichierXML.Film[2].Realisateur = "Jean-Marie POIRÉ";
fichierXML.Film[2].ActeurPrincipal ="Gérard Jugnot";
fichierXML.Film[2].ActeurPrincipal.@sexe ="F";
fichierXML.Film[2].Annee="1982";
fichierXML.Film[2].Genre = "Comédie";
fichierXML.Film[2].Pays="France";
```

Pour les méthodes `insertChildAfter()` ou `insertChildBefore()`, il est nécessaire de préciser deux paramètres. Le deuxième correspond ni plus ni moins à celui que nous venons de voir dans l'exemple précédent, alors que le premier doit spécifier un emplacement. Ainsi, les lignes d'instructions ci-dessous font référence au premier nœud de l'instance XML.

```
fichierXML.insertChildBefore(fichierXML.Film[0],fichierXML.Film[0]);
fichierXML.insertChildAfter(fichierXML.Film[1],fichierXML.Film[0]);
```

Connaître le nom d'une balise

Comme nous l'avons déjà évoqué, la fonction `toXMLString()` est chargée d'afficher les balises XML autour de la valeur d'un nœud.

```
zoneAffichage.text = fichierXML.Film[1].Realisateur.toXMLString();
```

Nous obtenons l'affichage ci-dessous en utilisant le fichier `cinema.xml`.

```
<Realisateur>Steven SODERBERGH</Realisateur>
```

Il existe également une autre fonction, plus précise, qui renvoie uniquement le nom de la balise. C'est cette dernière que vous devrez utiliser pour connaître le nom d'une balise.

```
zoneAffichage.text = fichierXML.Film[1].Realisateur.localName();
```

Nous obtenons l'affichage ci-dessous :

```
Realisateur
```

Connaître le nombre de nœuds enfants d'un nœud

La technique, bien qu'elle soit logique, n'est pas des plus évidentes.

Dans un premier temps, nous devons chercher à récupérer les nœuds enfants d'un nœud précis avec la méthode `children()`.

```
zoneAffichage.text = fichierXML.children();
```

Dans un deuxième temps, nous pouvons utiliser, de façon plus logique, la méthode `length()`.

La ligne d'instruction ci-dessous, qui se trouve dans le fichier `xml7.fla`, permet de connaître le nombre de nœuds enfants à la racine du document. Il y en a quatre.

```
zoneAffichage.text = fichierXML.children().length();
```

Remarque

Pour connaître le nombre d'attributs contenus dans une balise, il suffit de les stocker dans une instance de la classe `XMLList()` puis de les compter avec la propriété `length()`.

Déterminer le numéro d'index d'un nœud

Fichiers de référence de l'exemple 1 : `Chapitre13/xml9.fla` et `Chapitre13/cinema.xml`

Afin de comprendre très simplement et rapidement comment déterminer le numéro d'index d'un nœud, examinons les exemples suivants.

Exemple 1

Nous connaissons un nom de film, mais nous souhaiterions connaître le nom de son réalisateur. Nous avons vu, dans les pages précédentes, qu'une recherche suffisait pour découvrir toutes les informations relatives à cette production, mais nous désirons connaître le numéro de nœud, appelé aussi numéro d'index, afin d'utiliser la valeur obtenue.

```
var numeroNoeud:Number=fichierXML.Film.(@nom=="Mona Lisa Smile").childIndex();
  zoneAffichage.text=fichierXML.Film[numeroNoeud].Realisateur;
```

Nous stockons dans une première variable, le numéro d'index du nœud dont la valeur de l'attribut `nom` est `Mona Lisa Smile`. Nous utilisons ensuite la valeur obtenue comme paramètre pour pointer vers un nœud précis (`Film[numeroNoeud]`).

Exemple 2

Le fichier `france.xml` contient 22 nœuds à la racine (car il y a 22 régions françaises) qui regroupent des nœuds enfants dont les noms de balises sont `Departement`. Nous souhaiterions savoir quel est le nom du département qui porte le numéro 28. Voici la ligne d'instruction correspondante :

```
fichierXML.Region.Departement.(@numero==28).@nom;
```

Et maintenant, comment pouvons-nous connaître le numéro d'index du nœud parent, qui fait partie des 22 nœuds ? Voici la réponse.

```
fichierXML.Region.Departement.(@numero==28).parent().childIndex();
```

Nous avons remplacé l'attribut `@nom` par `parent()` car nous souhaitons remonter d'un niveau, mais nous ne cherchons pas à exploiter des données contenues dans le nœud. Nous souhaitons remonter au niveau du nœud parent, car nous voulons seulement connaître le numéro d'index de ce dernier. Nous utilisons alors la méthode `childIndex()`. Ainsi, la valeur renvoyée est bien le chiffre 6, dans cet exemple.

Tableau de synthèse

Voici un tableau récapitulatif des principales lignes d'instructions à retenir. Nous avons fait appel au fichier `france.xml`.

Le script initial dans lequel se trouvent ces lignes d'instructions est le suivant :

```
var fichierXML:XML;
var chargeurDonnees:URLLoader = new URLLoader();
var adresseFichierXML:URLRequest = new URLRequest("france.xml");

function onComplete(event:Event):void {
  fichierXML = new XML(chargeurDonnees.data);
  trace(fichierXML); //Remplacez fichierXML par l'un des exemples du tableau ci-dessous
}
chargeurDonnees.load(adresseFichierXML);
chargeurDonnees.addEventListener(Event.COMPLETE, onComplete);
```

Le fichier `tableausynthese.fla` contient les fonctions `trace()` qui ont permis d'obtenir ces résultats.

Paramètre de la fonction trace() du script ci-dessus	Résultat	Description
`fichierXML`	Consultez le paragraphe placé après ce tableau.	Lire tout le contenu d'un document XML
`fichierXML.Region[20]`	Consultez le paragraphe placé après ce tableau.	Lire un nœud (premier exemple)
`fichierXML.Region[20].Population.Annee[2]`	4 750 947	Lire un nœud (deuxième exemple)
`fichierXML.Region[20].@nom`	Provence-Alpes-Côte d'azur	Lire un attribut (premier exemple)
`fichierXML.Region[20].Departement[1].@nom`	Hautes-Alpes	Lire un attribut (deuxième exemple)
`fichierXML.Region[20].Population.Annee`	<Annee numero="1990">4 257 907</Annee> <Annee numero="1999">4 506 151</Annee> <Annee numero="2005">4 750 947</Annee> <Annee numero="2006">4 780 989</Annee>	Lire tous les nœuds enfants d'un nœud

Paramètre de la fonction trace() du script ci-dessus	Résultat	Description
`fichierXML.Region.(@nom== ➥"Provence-Alpes-Côte d'azur")`	Même résultat que la deuxième ligne de ce tableau.	Rechercher un nœud en fonction d'un attribut donné.
`fichierXML.Region.Population.Annee. ➥(@numero==1990)`	Renvoie 22 lignes correspondants aux années 1990 <Annee numero="1990">1 624 372</ Annee> … <Annee numero="1990">5 350 701</ Annee>	Rechercher des nœuds en fonction d'un attribut donné.
`zoneAffichage.text = fichierXML.Film.(Annee==2007)`	Cet exemple ne peut s'appliquer au document `france.xml` car les nœuds `Departement` ne contiennent pas de valeurs.	Rechercher un nœud
`fichierXML.Region.Departement. ➥(@nom.substr(0,1)=="A" && @numero<=6)`	Nous recherchons les noms de département qui commencent par la lettre A et dont le numéro de département est inférieur ou égal à 6.	Recherche multicritère
`fichierXML.Region[0].Population. ➥Annee[2]="0"`		Modifier la valeur d'une balise
`fichierXML.Region[0].@nom="Alzasse"`		Modifier la valeur d'un attribut
`fichierXML.Region.Departement[1]. ➥localName()`	Département	Connaître le nom d'une balise
`fichierXML.Region[3].children(). ➥length()-1`	Dans notre exemple, nous retirons 1 à la valeur renvoyée par la méthode `length()` car le dernier nœud n'est pas un département.	Connaître le nombre de nœuds enfants d'un nœud
`fichierXML.Film.(@nom=="Mona ➥Lisa Smile").childIndex()`	Nous recherchons le numéro d'index en fonction du résultat d'une recherche sur la valeur d'un attribut.	Connaître le numéro d'index d'un nœud.
`fichierXML.appendChild(fichierXML. ➥Film[0]);`	Ajoute, à la fin de l'instance XML, un nœud dont le contenu est celui qui correspond au premier nœud du document.	Insérer un nœud à la fin de l'instance XML.
`fichierXML.insertChildBefore(fichierXML.Film[5],fichierXML.Film[0])`	Insère le premier nœud de l'arborescence (en tant que modèle) avant le cinquième nœud du document XML.	Insérer un nœud avant un autre.

Les résultats retournés par les deux premières lignes de ce tableau sont trop longs pour tenir dans une cellule du tableau : `fichierXML` renvoie l'intégralité du document alors que `fichierXML.Region[20]` renvoie les informations ci-dessous :

```
<Region nom="Provence-Alpes-Côte d'azur">
  <Departement nom="Alpes de Hautes-Provence" numero="4" prefecture="Digne"/>
  <Departement nom="Hautes-Alpes" numero="5" prefecture="Gap"/>
  <Departement nom="Alpes-Maritimes" numero="6" prefecture="Nice"/>
  <Departement nom="Bouches-du-Rhône" numero="13" prefecture="Marseille"/>
  <Departement nom="Var" numero="83" prefecture="Toulon"/>
  <Departement nom="Vaucluse" numero="84" prefecture="Avignon"/>
  <Population>
    <Annee numero="1990">4 257 907</Annee>
    <Annee numero="1999">4 506 151</Annee>
    <Annee numero="2005">4 750 947</Annee>
    <Annee numero="2006">4 780 989</Annee>
  </Population>
</Region>
```

14

La gestion du texte

Lorsque nous entendons parler de Flash, nous pensons tout de suite à l'animation à base d'interpolations, au son, à la vidéo, aux images, au dessin vectoriel, à l'ActionScript, mais plus rarement au texte.

Depuis très longtemps, cette notion est plus ou moins négligée. Il suffit d'ailleurs de consulter les programmes de formation des nombreux établissements qui proposent des stages sur l'ActionScript pour s'apercevoir que cette notion est souvent traitée de façon incomplète, voire pas du tout. Et pourtant, avec l'évolution croissante et importante de l'ActionScript dans Flash, les possibilités de développement sont de plus en plus grandes. La gestion du texte dans une animation joue donc un rôle important car elle se traduit par de nombreuses possibilités :

- la création dynamique d'un texte sur la scène ;
- sa mise en forme ;
- la surveillance d'une zone de texte de saisie.

Ces trois notions regroupent de nombreuses techniques de manipulation du texte qui permettent de gagner du temps en développement. Malheureusement, seuls les développeurs aguerris savent gérer correctement ce média.

Remarque

Il est à noter que toutes les propriétés abordées dans ce chapitre sont valables pour les textes créés directement sur la scène à l'aide de l'outil dédié (outil Texte) ainsi que pour les textes créés dynamiquement à l'aide de la classe `TextField()`.

Si vous n'avez aucune notion de typographie, nous vous recommandons fortement la lecture de cette page Web, très claire et très explicite : `http://www.blaco.ch/Pages_prepresse/Un_peu_de_typo.htm`.

Créer un texte dynamiquement

Il paraît généralement évident de devoir importer dynamiquement (en AS) une image sur la scène, mais la création d'un texte semble moins automatique. Pour preuve, nombreux sont les développeurs qui transforment un texte dynamique sur la scène en symbole afin de pouvoir placer ce dernier en ActionScript. Cela permet, en effet, d'éviter la programmation de l'incorporation d'une police de caractères et la mise en forme du texte. Cependant cette technique est moins souple que ce que vous allez pouvoir lire dans ce chapitre.

Pour créer un texte en ActionScript, la technique est extrêmement simple : il suffit de créer une instance de la classe `TextField()` puis de faire appel à la propriété `text` pour définir son contenu. Bien sûr, terminez le script par un ajout de cette nouvelle instance à la liste d'affichage.

Fichier de référence : `Chapitre14/texte1.fla`

```
var messageAccueil:TextField = new TextField();
messageAccueil.text = "Bonjour";
addChild(messageAccueil);
```

ou dans un fichier externe intitulé `main.as` et `Chapitre14/texte2.fla`

```
package {

  import flash.text.TextField;
  import flash.display.Sprite;

  public class main extends Sprite {

    public var messageAccueil:TextField;

    function main() {
      messageAccueil = new TextField();
      messageAccueil.text = "Bonjour";
      addChild(messageAccueil);
    }


  }

}
```

Remarque

Dans la pratique, lorsque vous aurez besoin de travailler du texte en ActionScript, il sera tout de même plus judicieux de créer une classe dédiée. Consultez le dernier développement (Instanciation de classes personnalisées) du chapitre 6 ; vous y découvrirez un exemple.

Quelques mots sur les pages de ce chapitre

Comme nous l'évoquions quelques lignes plus haut, toutes les propriétés abordées dans ce chapitre sont valables pour les textes créés directement sur la scène ainsi que pour les textes créés dynamiquement à l'aide de la classe `TextField()`. Dans le cas où vous créeriez votre propre texte sur la scène en lui donnant pour nom celui qui figure devant l'instruction `:TextField = new TextField();` de chaque exemple, vous pourriez alors vous passer de saisir les trois lignes ci-dessous :

```
var titreAccueil:TextField = new TextField();
titreAccueil.text = "Yazo";
addChild(titreAccueil);
```

À l'exécution des lignes d'instructions contenues dans tous les exemples de ce paragraphe, quelle que soit la méthode (approche POO ou programmation séquentielle), il faut reconnaître que l'apparence du texte par défaut n'est pas très agréable à lire. C'est tout à fait normal car nous ne l'avons pas formaté. En ce qui concerne sa position sur la scène, c'est à vous de faire appel aux propriétés `x` et `y` pour le placer précisément.

```
messageAccueil.x = 30;
messageAccueil.y = 20;
```

Stocker un nombre dans un texte dynamique

Pour placer un contenu dans un texte dynamique ou un texte de saisie, vous avez pu constater que la technique est extrêmement simple : il suffit de faire référence à la propriété `text`.

En revanche, un problème se pose lorsque vous souhaitez placer un nombre dans un texte dynamique, car l'instance est typée naturellement. En effet, l'exécution des lignes d'instructions ci-dessous entraîne une erreur de compilation.

```
var titreAccueil:TextField = new TextField();
titreAccueil.text = 30;
addChild(titreAccueil);
1067: Contrainte implicite d'une valeur du type int vers un type sans rapport String.
```

Cela signifie que vous essayez de stocker un nombre dans la variable `titreAccueil` qui est de type `String` (texte) et qui attend donc une variable de ce type.

Pour remédier à ce problème, vous disposez d'une solution qui consiste à placer la valeur à stocker entre guillemets.

```
var titreAccueil:TextField = new TextField();
titreAccueil.text = "30";
addChild(titreAccueil);
```

Remarque

Si vous aviez besoin de récupérer la valeur stockée dans l'instance de type texte intitulée `titreAccueil` dans le but d'effectuer un calcul, il faudrait alors faire une conversion en utilisant la classe `Number()`, comme :

```
trace(Number(titreAccueil.text)*3);
```

Méthode toString()

Si la valeur à stocker résulte d'un calcul ou de la lecture d'une propriété ou d'une variable, vous devez dans ce cas utiliser la méthode `toString()`.

```
var positionJoueur:TextField = new TextField();
positionJoueur.text = joueur.x.toString();
addChild(positionJoueur);
```

Dans cet exemple, nous cherchons à stocker un nombre de pixels (renvoyé par la propriété `x` de l'instance `joueur`) dans l'instance d'un `TextField()`. L'adjonction de la méthode `toString()` permet de convertir le résultat renvoyé par la lecture de `joueur.x`.

Méthode appendText() contre +=

Lorsque vous tenterez de faire des concaténations de texte, il est fortement conseillé par Adobe de ne pas utiliser l'opérateur += comme vous aviez éventuellement l'habitude de le faire en AS1/2. Favorisez l'usage de la méthode `appendText()`.

```
points.appendText(" - Meilleur score");
```

Vous auriez écrit en AS1/2 :

```
points+=" - Meilleur score";
```

Les propriétés de la classe TextField()

Il serait très simple et plus rapide de vous présenter un tableau de synthèse avec toutes les méthodes et les propriétés les plus utiles de la classe `TextField()`, mais cela ne servirait à rien car ce point est bien traité dans la documentation. Nous allons plutôt mettre en avant les propriétés que vous devez absolument connaître pour pouvoir gérer correctement un texte dans une animation Flash et vous proposer un exemple et son contexte.

Régler la couleur du fond

> **Remarque**
>
> Il est à noter que cette propriété est valable également pour les textes créés directement sur la scène à l'aide de l'outil dédié (outil Texte).

L'attribution d'une couleur de fond passe par deux lignes d'instructions. En effet, vous devez toujours commencer par activer la visualisation d'un fond de texte. Sans cette première étape, vous pouvez toujours essayer de changer la couleur, aucun effet se produira.

```
var titreAccueil:TextField = new TextField();
titreAccueil.text = "Yazo";
titreAccueil.background = true;
titreAccueil.backgroundColor = 0xFF6666;
addChild(titreAccueil);
```

L'attribution d'une couleur de fond s'avère indispensable lorsque vous placez un texte de saisie sur la scène, alors qu'aucune mise en évidence n'a été prévue dans la charte graphique.

Vous pouvez également vous contenter de régler la valeur de la propriété `background` à `true` sans pour autant définir une couleur de fond ; dans ce cas, vous obtiendrez du blanc.

L'application d'une couleur de fond s'effectue généralement sur un texte de saisie, mais plus rarement sur un texte dynamique.

Régler la couleur du contour

Comme nous venons de le voir pour l'attribution d'une couleur de fond, pour appliquer un contour à un texte de saisie ou un texte dynamique, il est nécessaire de faire appel à deux lignes d'instruction. La première sert à activer la visualisation d'un contour de texte. Sans cette première étape, vous pouvez toujours essayer de changer la couleur de contour, aucun effet se produira.

```
var titreAccueil:TextField = new TextField();
titreAccueil.text = "Yazo";
titreAccueil.border = true;
titreAccueil.borderColor = 0xFF6666;
addChild(titreAccueil);
```

L'attribution d'une couleur de contour s'avère indispensable lorsque vous placez un texte de saisie sur la scène, alors qu'aucune mise en évidence n'a été prévue dans la charte graphique. Elle peut remplacer ou accompagner une couleur de fond.

Si vous réglez la valeur de la propriété `border` à `true` sans pour autant définir de couleur de contour, celle-ci sera noire par défaut.

> **Astuce**
>
> Lorsque vous construisez une interface avec plus ou moins de texte, il est conseillé d'activer la propriété `border` afin de visualiser les différentes zones de type texte. Cela peut notamment vous permettre de comprendre les problèmes d'alignements rencontrés.

Régler la couleur du texte

Vous découvrirez, plus loin dans ce chapitre, la possibilité de mettre un texte en couleur avec la classe `TextFormat()`, mais vous pouvez d'ores et déjà le faire plus simplement avec la propriété `textColor` de la classe `TextField()`.

```
var titreAccueil:TextField = new TextField();
titreAccueil.text = "Yazo";
titreAccueil.textColor = 0x0000AA;
addChild(titreAccueil);
```

Si vous combinez cette mise en couleur du texte avec la méthode `setTextFormat()` de la classe `TextFormat()`, sachez que l'effet produit est celui de la dernière ligne d'instruction exécutée. La propriété `textColor` de la classe `TextField()` n'a donc pas la priorité sur la propriété `color` de la classe `TextFormat()` et réciproquement.

Régler automatiquement la largeur d'un texte

Lorsqu'un texte est plus grand que la largeur de l'instance dans laquelle il se trouve, il est possible de spécifier un retour automatique à la ligne. Dans certains cas, cela s'avère inutile, car ce n'est pas ce que nous souhaitons.

Lorsque vous créez une instance de la classe `TextField()`, vous pouvez lui définir une largeur avec la propriété `width`, mais ce n'est pas obligatoire, notamment dans le cas où vous souhaitez créer un titre, ou tout du moins un texte sur une ligne, avec un corps assez important.

Pour ne pas avoir à gérer la largeur d'une instance de type `TextField()`, vous pouvez utiliser la propriété `autoSize`.

Spécifiez dans votre code, le type de redimensionnement (`TextFieldAutoSize.NONE`, `TextFieldAutoSize.LEFT`, `TextFieldAutoSize.RIGHT`, `TextFieldAutoSize.CENTER`) ; il sera alors inutile de faire appel à la propriété `wordWrap`. Voici un exemple de texte dont le redimensionnement s'effectue automatiquement à gauche.

```
var semaine:Array = new Array("Lundi 30 février 2017","Mardi, le jour le plus long",
➥"Mercredi","Jeudi","Vendredi","Samedi","Dimanche");
var titreAccueil:TextField = new TextField();

titreAccueil.autoSize = TextFieldAutoSize.LEFT;

for each (var jour:String in semaine) {
  titreAccueil.appendText(jour+"\n");
}

addChild(titreAccueil);
```

Attention de ne pas combiner ce réglage avec la propriété `wordWrap`, car cette dernière annule la définition de propriété `autoSize`.

> **Remarque**
>
> Appliquez un contour de texte (`nomInstanceTexte.border=true`) pour visualiser le résultat du redimensionnement automatique d'une instance de `TextField()`.

Gérer un texte multiligne

La gestion d'un texte multiligne sous-entend deux problématiques à résoudre.

- La première est relative à l'insertion d'un saut de ligne entre deux données. Vous allez pour cela devoir utiliser la séquence `\n`.
- La deuxième est relative au renvoi du texte à la ligne suivante lorsqu'il est trop large.

Examinez bien la copie d'écran de la figure 14-1 : nous y mettons en évidence le retour à la ligne.

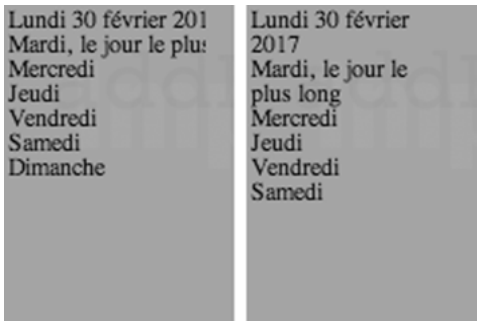


Figure 14-1

Avec et sans l'utilisation de la propriété wordWrap

Si nous ne spécifions aucun réglage dans notre programme, un texte plus grand que la largeur de l'instance du `TextField()` dans laquelle il se trouve entraîne une rupture de la lecture. Le texte est tronqué. C'est le cas dans la colonne de gauche de la copie d'écran de la figure 14-1. Pour obtenir un renvoi automatique d'une phrase à la ligne suivante, nous devons utiliser la propriété `wordWrap` et la régler à `true`.

> **Note**
> Consultez le chapitre 10 dédié aux boucles `for()` si vous ne connaissez pas les itérations.

```
var semaine:Array = new Array("Lundi 30 février 2017","Mardi, le jour le plus long",
➥"Mercredi","Jeudi","Vendredi","Samedi","Dimanche");
var titreAccueil:TextField = new TextField();

titreAccueil.wordWrap = true;

for each(var jour:String in semaine) {
  titreAccueil.appendText(jour+"\n");
}

addChild(titreAccueil);
```

Revenons sur un détail qui ne vous aura sûrement pas échappé : la présence de la séquence que nous évoquions en introduction, `\n`.

Pour pouvoir renvoyer une donnée à la ligne, nous la concaténons à l'aide de l'opérateur + suivi de la séquence `\n`.

> **Remarque**
> Pour les utilisateurs de l'AS2, vous remarquerez l'emploi de la nouvelle méthode `appendText()` qui est préconisé par Adobe, au lieu de l'opérateur +=.

Empêcher la sélection d'un texte dynamique

Lorsque vous créez un texte dynamiquement, vous pouvez, par défaut, sélectionner son contenu sur la scène. Pour éviter cela, ce qui entraînera également l'annulation du changement de curseur lors d'un survol du texte, vous devez utiliser la propriété `selectable`.

```
var titreAccueil:TextField = new TextField();
titreAccueil.text = "Yazo";
titreAccueil.selectable = false;
addChild(titreAccueil);
```

Veillez à ne pas commettre l'erreur d'appliquer cette propriété à un texte de saisie !

Régler le type de texte (saisie ou dynamique)

Lorsque vous créez un texte sur la scène, vous pouvez choisir entre les types dynamique, statique et de saisie.

Rappelons qu'un texte de type dynamique permet de placer, sur la scène, une zone dans laquelle le contenu va pouvoir être modifié avec l'ActionScript. Comme son nom l'indique, un texte de saisie donne à l'utilisateur la possibilité de saisir un texte directement dans l'interface de votre animation. Lorsque vous proposez la saisie d'un nom, d'un prénom, d'une adresse, d'un e-mail, etc., il s'agit d'un texte de saisie.

Par défaut, lorsque vous exécutez simplement les lignes d'instructions ci-dessous, un texte est de type `dynamic`.

```
var titreAccueil:TextField = new TextField();
titreAccueil.text = "Yazo";
addChild(titreAccueil);
```

Pour changer le type d'un texte, il vous suffit tout simplement d'utiliser la propriété `type` et de spécifier l'une des deux constantes suivantes :

- `TextFieldType.INPUT`
- `TextFieldType.DYNAMIC`

Vous devriez obtenir le script suivant :

```
var titreAccueil:TextField = new TextField();
titreAccueil.width = 200;
titreAccueil.text = "Veuillez saisir votre prénom";
addChild(titreAccueil);
titreAccueil.type = TextFieldType.INPUT;
```

Cacher les caractères d'un mot de passe

Pour certaines zones de texte, il est nécessaire de masquer la saisie de l'utilisateur. C'est notamment le cas pour un mot de passe. Vous devrez alors spécifier la valeur de la propriété `displayAsPassword` à `true`.

```
var titreAccueil:TextField = new TextField();

titreAccueil.border = true;
```

```
titreAccueil.width = 150;
titreAccueil.height = 20;

titreAccueil.displayAsPassword = true;
titreAccueil.type = TextFieldType.INPUT;

addChild(titreAccueil);
```

Dans cet exemple, nous avons décidé d'appliquer un contour à l'instance de type `TextField()`, ce qui indique à l'utilisateur l'endroit dans lequel il doit cliquer pour saisir un mot de passe.

Déterminer et contrôler les caractères contenus dans un texte

Le nombre de caractères contenus dans une chaîne

Pour déterminer le nombre de caractères contenus dans un texte dynamique ou un texte de saisie, utilisez la propriété `length`.

```
trace(titreAccueil.length);
```

Le nombre maximum de caractères

En revanche, si vous souhaitez limiter le nombre de caractères qu'un utilisateur peut saisir dans un texte de type input (texte de saisie), utilisez la propriété `maxChars`.

```
var titreAccueil:TextField = new TextField();

titreAccueil.border = true;
titreAccueil.maxChars = 5;
titreAccueil.type = TextFieldType.INPUT;

addChild(titreAccueil);
```

L'usage de cette propriété s'avère bien souvent indispensable pour contrôler la saisie de l'utilisateur lorsqu'il est invité à indiquer une date, un code postal, un numéro de téléphone, etc., dans l'interface de l'animation. Toutes ces informations sont effet souvent formatées avec un nombre de signes bien défini.

Restreindre la saisie à certains caractères

Dans les exemples ci-dessous, nous pouvons facilement comprendre qu'un utilisateur ne doit pas saisir des lettres dans un code postal ou dans un numéro de téléphone. À l'inverse, dans une zone où l'utilisateur est invité à saisir un prénom ou un nom, les chiffres ne peuvent être acceptés. La syntaxe de la ligne d'instruction à utiliser pour une telle contrainte est très simple. Examinez le script ci-dessous :

```
var codePostal:TextField = new TextField();

codePostal.border = true;
```

```
codePostal.type = TextFieldType.INPUT;
codePostal.maxChars = 5;
codePostal.restrict = "0-9";

addChild(codePostal);
```

Nous avons fait appel à la propriété `restrict` et nous avons indiqué la plage des caractères autorisés. Ainsi, pour n'autoriser que des caractères, nous aurions pu écrire : `"A-Z"` ou `"a-z"` ou encore `"A-z a-z"`.

> **Remarque**
>
> Si vous contraignez la saisie d'un texte en minuscules (`"a-z"`) et si l'utilisateur tape des majuscules, les caractères seront automatiquement convertis en minuscules.

Pour exclure certains caractères, vous devez utiliser le caractère `^` suivi du signe typographique ou de la plage de caractères à supprimer.

L'exemple ci-dessous autorise l'utilisateur à saisir tous les caractères minuscules de a à z à l'exception de la lettre y.

```
codePostal.restrict = "a-z^y";
```

Dans cet autre exemple, tous les caractères minuscules de a à z peuvent être saisis à l'exception des lettres r, s et t.

```
codePostal.restrict = "a-z^r-t";
```

Consultez le développement suivant pour découvrir que la casse d'une chaîne de caractères peut être modifiée.

Manipuler une chaîne de caractères

Cette partie du chapitre 14 est extrêmement importante car elle va vous apprendre à effectuer des recherches dans une chaîne de caractères sans être obligé d'utiliser les expression régulières. De plus, vous allez découvrir comment manipuler partiellement des chaînes de caractères. Ajoutons simplement que les méthodes et propriétés qui vont suivre appartiennent à la classe `String()`.

Avant que nous développions davantage d'autres propos, précisons que pour utiliser les méthodes de cette classe, vous n'êtes pas obligé d'instancier la classe `String()` avec l'opérateur `new`. Vous pouvez tout aussi bien vous contenter de typer une variable de la façon suivante :

```
var prenom_txt:String = "David";
```

Dans le cas ou vous préféreriez utiliser le mot-clé `new`, voici comment vous procéderiez :

```
var prenom_txt:String = new String("David");
```

Il est également possible d'utiliser la même ligne d'instruction ci-dessus sans l'opérateur `new`.

Changer la casse d'un texte

Vous aurez parfois besoin de changer la casse d'un texte de minuscules en majuscules et inversement. La propriété `restrict` peut convertir un affichage de caractères dans la casse de votre choix, mais uniquement lors de la saisie d'un texte de la part de l'utilisateur. Nous allons maintenant découvrir deux méthodes de la classe `String()` qui peuvent convertir la casse d'un texte a posteriori.

- `toUpperCase()` pour basculer l'affichage de caractères en majuscules ;
- `toLowerCase()` pour basculer l'affichage de caractères en minuscules.

Vous remarquerez que les deux noms employés sont accompagnés de parenthèses car il s'agit de méthodes et non de propriétés. Dans l'exemple ci-dessous, nous affichons dans un texte sur la scène une chaîne de caractères composée d'une concaténation comprenant une conversion à l'aide de la méthode `toUpperCase()`.

```
affichageMessage.text = "Votre réponse, "+reponse.text.toUpperCase()+", est bonne";
```

Lorsque vous devrez comparer le contenu d'un texte de saisie avec une variable ou une chaîne de caractères, si vous ne souhaitez pas que cette comparaison soit sensible à la casse, vous ferez alors appel à l'une de ces deux méthodes (`toUpperCase()` ou to `LowerCase()`).

```
if(reponse.text.toUpperCase()=="PARIS") pointsGeneraux++;
```

Dans cet autre exemple, l'utilisateur n'a pas à se soucier de la casse du texte qu'il a saisi : une conversion automatique est effectuée.

```
var reponse:TextField = new TextField();

reponse.border = true;
reponse.height = 20;
reponse.width = 150;
reponse.type = TextFieldType.INPUT;

addChild(reponse);

reponse.addEventListener(Event.CHANGE,basculerMajuscule);

function basculerMajuscule (evt:Event) {
  reponse.text = reponse.text.toUpperCase();
}
```

Vous remarquerez que nous avons utilisé un écouteur gérant l'événement `Event`. Consultez le développement intitulé Gérer les événements liés au texte, dans les dernières pages de ce chapitre, pour avoir une approche globale du problème.

Si vous connaissez la propriété `restrict` de la classe `TextField()`, vous ne trouverez peut-être pas cet exemple pertinent. Voici une variante du script ci-dessus qui permet de basculer partiellement l'affichage d'un texte avec une majuscule en début de mot et le reste en minuscules.

```
var prenom:TextField = new TextField();
var longeurChaine:Number;
```

```
prenom.border = true;
prenom.height = 20;
prenom.width = 150;
prenom.type = TextFieldType.INPUT;

addChild(prenom);

prenom.addEventListener(FocusEvent.FOCUS_IN,basculerMajuscule);

function basculerMajuscule (evt:Event) {
  longeurChaine = prenom.length;
  prenom.text = prenom.text.substr(0,1).toUpperCase()+prenom.text.
  ➥substr(1,longeurChaine).toLowerCase();
}
```

Que l'utilisateur saisisse son nom intégralement en majuscules ou en minuscules, une majuscule viendra toujours se placer en début de texte.

Vérifier la présence d'une chaîne de caractères dans un texte

Il sera parfois nécessaire de vérifier la présence d'un ou plusieurs caractères dans un texte de saisie, un texte dynamique ou une variable. Imaginons que vous invitiez l'utilisateur à saisir son adresse e-mail : vous devez vous assurer que les signes typographiques arobase et point ont été saisis. Vous devrez alors faire appel à la méthode `indexOf()` qui renvoie soit le numéro d'index du caractère rencontré (ou le premier caractère d'une chaîne recherchée) soit la valeur -1 si rien n'a été trouvé.

```
var adressemail:String = "david@yazo.net";
trace(adressemail.indexOf("@"));
5
var adressemail:String = "david@yazo.net";
trace(adressemail.indexOf("$"));
-1
var adressemail:String = "david@yazo.net";
trace(adressemail.indexOf("."));
10
var adressemail:String = "david@yazo.net";
trace(adressemail.indexOf("yazo"));
6
```

Nous rappelons que le premier caractère d'une chaîne porte l'index 0 ; c'est pourquoi nous obtenons les valeurs 5, 6 et 10 et non pas 6, 7 et 11.

En conclusion, lorsque vous cherchez à savoir si un caractère ou une chaîne est contenu dans un texte, testez uniquement l'inégalité avec la valeur -1.

```
var adresseValide:Boolean;
var adressemail:String = "david@yazo.net";
if (adressemail.indexOf("@")!=-1) {
  adresseValide=true;
  trace(adresseValide);
}
```

Remarque

Comme nous l'évoquions précédemment, pour valider correctement une adresse mail, vous devez vous assurer de la présence d'un point. Dans un souci de simplification du code et pour une meilleure compréhension, nous avons simplement vérifié la présence de l'arobase. Consultez éventuellement le chapitre 9 pour de plus amples informations sur les tests conditionnels multiples.

Remplacer un texte par un autre

Dans certains cas, vous aurez besoin de supprimer une portion de texte pour la remplacer par une autre. Cette manipulation s'effectue très simplement à l'aide de la méthode `replaceText()`. Indiquez simplement la plage de caractères à remplacer ainsi que le texte de remplacement, que vous passerez également en paramètre de la fonction.

```
zoneInfo.text = "La pédagogie est une qualité qui s'acquière facilement.";
zoneInfo.replaceText(44,54,"difficilement");
```

La valeur `44` correspond au 44e caractère depuis le début du texte contenu dans l'instance `zoneInfo` en commençant à compter à partir de 0 et non 1. En revanche, le chiffre `54` ne correspond pas au nombre de caractères à remplacer dans l'instance, mais au numéro du caractère de fin de la plage de texte à remplacer. Contrairement au premier paramètre, il faut compter le premier caractère de la chaîne à partir de 1.

Autre méthode

Vous pouvez également utiliser la méthode `replaceSelectedText()` pour remplacer un texte sélectionné.

Mettre en forme un texte avec la classe TextFormat()

Si vous avez lu ce chapitre depuis le début, vous avez pu constater que la classe `TextField()` possède déjà de nombreuses propriétés et méthodes qui permettent de mettre un texte en forme. Cependant, ces différents réglages s'appliquent principalement à l'enveloppe du `TextField` et non au texte contenu dans ce contenant (à l'exception de la propriété `textColor`). Le tableau 14-3 situé à la fin de ce chapitre vous permettra de vérifier cette affirmation.

Pour mettre un texte en forme avec plus de précision (interlettrage, soulignement, retrait de la première ligne d'un paragraphe, alignement, choix de la police, etc.), nous devons faire appel à la classe `TextFormat()`.

Pour commencer, vous devez comparer l'utilisation d'une instance de la classe `TextFormat()` avec une feuille de styles. Le fonctionnement est analogue, car vous définissez des valeurs assignées à des attributs sous une étiquette générale, puis vous l'appliquez avec la méthode `setTextFormat()` de la classe `TextField()`. Le script ci-dessous correspond aux lignes d'instructions minimales que vous devez utiliser pour faire appel à cette classe. Nous instançions la classe, nous définissons une propriété puis nous appliquons le style.

```
var stylePerso:TextFormat = new TextFormat();
stylePerso.size=18;
nomduntextfield.setTextFormat(stylePerso);
```

Cet exemple sous-entend qu'un texte dynamique ou un texte de saisie se trouve sur la scène et s'intitule `nomduntextfield`. Vous pouvez dès lors ajouter autant de propriétés que vous le souhaitez.

Mise en forme d'un texte de saisie

Pour appliquer un style par défaut à un texte saisi par l'utilisateur, utilisez la propriété `defaultTextFormat` comme dans l'exemple ci-dessous :

```
var stylePerso:TextFormat = new TextFormat();
stylePerso.size = 14;
stylePerso.bold = true;
nomutilisateur.defaultTextFormat = stylePerso;
```

Le premier exemple que nous allons aborder, se contente de régler la taille d'un texte et son alignement. La zone de texte possède la même largeur que celle de la scène (500 pixels). Ainsi, en demandant un alignement centré, le texte se trouvera au centre de l'animation.

Fichier de référence : `Chapitre14/texte5.fla`

```
var titreAccueil:TextField = new TextField();
titreAccueil.text = "Accueil";
titreAccueil.y = 20;
titreAccueil.width = 500;

var styleTitreAccueil:TextFormat = new TextFormat();
styleTitreAccueil.align="center";
styleTitreAccueil.size=24;

titreAccueil.setTextFormat(styleTitreAccueil);

addChild(titreAccueil);
```

Ce dernier exemple fait appel à deux propriétés souvent utilisées : la couleur d'un texte et l'interlettrage. Nous n'approfondirons pas, dans cet exemple, l'usage de cette classe, car la fin de ce chapitre fait référence à de nombreuses situations qui gèrent les propriétés les plus utilisées de la classe `TextFormat()`.

Fichier de référence : `Chapitre14/texte6.fla`

```
var titreAccueil:TextField = new TextField();
titreAccueil.text = "Accueil";
titreAccueil.x = 20;
titreAccueil.y = 20;
titreAccueil.width = 200;

var styleTitreAccueil:TextFormat = new TextFormat();
styleTitreAccueil.color=0x474763;
```

```
styleTitreAccueil.letterSpacing =5;

titreAccueil.setTextFormat(styleTitreAccueil);

addChild(titreAccueil);
```

Nous vous conseillons de consulter le dernier développement intitulé Instanciation de classes personnalisées du chapitre 6 ; il présente la gestion de la classe `TextFormat()` à travers une classe externe.

Mettre en forme une plage de caractères

Dans le texte ci-dessous, nous aimerions mettre uniquement le nom de l'auteur de cette citation en gras.

« Un bon maître a ce souci constant : enseigner à se passer de lui. ». André GIDE.

Nous devons donc créer une instance de la classe `TextFormat()` et définir les valeurs des propriétés à utiliser. Nous faisons ensuite appel à la méthode `setTextFormat()` de la classe `TextField()` en ajoutant des paramètres que nous n'avions pas encore utilisés pour le moment : le début et la fin de la mise en forme du texte. Tous deux s'expriment en nombre de caractères (avec une petite différence).

```
citation.setTextFormat(styleAuteur,68,78);
```

La valeur `68` correspond au 68e caractère depuis le début du texte contenu dans l'instance `citation`, en commençant à compter à partir de 0. En revanche, le nombre `78` ne correspond pas au nombre de caractères à mettre en forme depuis la première valeur, mais au numéro du caractère de fin de la plage de texte à mettre en forme. Contrairement au premier paramètre, il faut compter le premier caractère de la chaîne à partir de 1. Voici un exemple complet (un texte dynamique intitulé `citation` a été préalablement crée sur la scène) :

```
citation.text = "'Un bon maître a ce souci constant : enseigner à se passer de lui.'
➥André GIDE.";

var styleAuteur:TextFormat = new TextFormat();
styleAuteur.bold=true;
citation.setTextFormat(styleAuteur,68,78);
```

Vous obtenez : `'Un bon maître a ce souci constant : enseigner à se passer de lui.'` **`André GIDE.`**

Il sera parfois difficile de connaître le numéro d'index à partir duquel la mise en forme doit commencer. Vous pourrez alors utiliser la méthode `indexOf()` qui permet de connaître la valeur que vous recherchez.

```
trace(citation.text.indexOf("André"));
```

Cette ligne d'instruction renverra le nombre `68`, c'est-à-dire l'index à partir duquel ce mot (sa première lettre) est rencontré.

Il est tout à fait possible d'appliquer plusieurs mises en forme à l'aide de plusieurs instances de la classe `TextFormat()` à un même texte dynamique ou de saisie.

```
citation.text = "'Un bon maître a ce souci constant : enseigner à se passer de lui.'
➥André GIDE.";

var styleAuteur:TextFormat = new TextFormat();
styleAuteur.bold=true;
citation.setTextFormat(styleAuteur,68,78);

var styleAdjectif:TextFormat = new TextFormat();
styleAdjectif.italic=true;
styleAdjectif.letterSpacing = 3;
citation.setTextFormat(styleAdjectif,25,34);
```

Encapsuler une police de caractères

Fichier de référence : `Chapitre14/texte3.fla`

Dans les deux premiers exemples que nous venons d'aborder, nous n'avons pas spécifié la police de caractères ; pourtant cela aurait été très simple :

```
var styleTitreAccueil:TextFormat = new TextFormat();
styleTitreAccueil.color=0x474763;
styleTitreAccueil.font = "Futura";
titreAccueil.setTextFormat(styleTitreAccueil);
```

Remarque

`titreAccueil` est le nom d'instance d'un texte dynamique présent sur la scène.

Mais que se passe-t-il lorsqu'un internaute consulte une animation qui utilise une police ne se trouvant pas sur son ordinateur ?

La réponse est malheureusement logique : le lecteur Flash substitue la police manquante par une autre ! Afin que tous les internautes puissent visualiser une animation avec la police que vous avez choisie, vous devez alors l'incorporer. Si vous le réalisez à partir de l'interface en cliquant sur le bouton Intégrer de la palette Propriétés, cela ne pose pas de problème. En revanche, lorsque vous créez, via l'ActionScript, un texte dynamique ou un texte de saisie sur la scène, vous ne pouvez pas préalablement cliquer sur le bouton Intégrer. Voici une procédure que vous pouvez suivre afin qu'au moment de l'exportation de l'animation, la police choisie soit encapsulée dans le fichier SWF.

Rappel

Les textes statiques d'une interface sont automatiquement vectorisés : il est alors inutile d'incorporer la moindre police. La procédure ci-dessous s'applique donc aux textes dynamiques et aux textes de saisie.

1. Commencez par importer une police dans la Bibliothèque de votre animation (via le menu local déroulant accessible en haut à droite de la bibliothèque, commande Nouvelle police…).

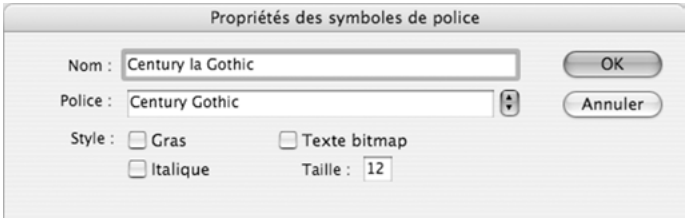


Figure 14-2

Le nom que vous donnez à la police ne sert qu'à être reconnu au sein de la Bibliothèque.

2. Effectuez un clic droit sur la police que vous venez de placer dans la Bibliothèque et sélectionnez la commande Liaison…
3. Cliquez sur la case Exporter pour ActionScript et donnez un nom de classe.

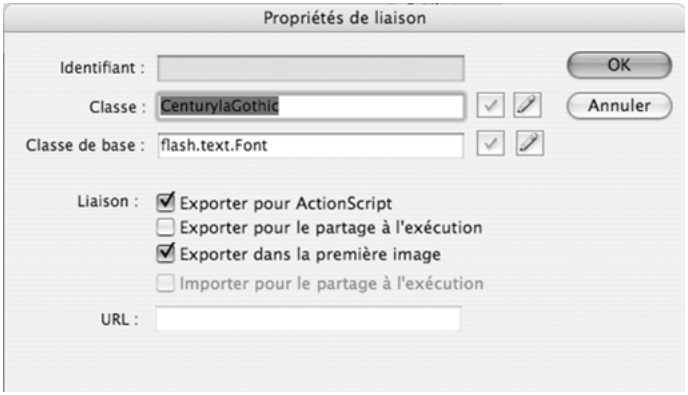


Figure 14-3

Peu importe le nom de la classe que vous donnez à la police, mais vous ne devez pas utiliser de caractères spéciaux ou accentués, ni même d'espace.

4. Fixez la propriété `embedFonts` de votre texte dynamique à `true`.
5. Faites référence à la propriété `font` de la classe `TextFormat()` et passez en paramètre le nom de la police que vous aviez importée.

Voici le script général que vous devriez obtenir pour afficher un texte lisible avec la police de votre choix, sur n'importe quel ordinateur.

```
var titreAccueil:TextField = new TextField();
titreAccueil.text = "Yazo";
titreAccueil.embedFonts = true;

var styleTitreAccueil:TextFormat = new TextFormat();
styleTitreAccueil.font="Century Gothic";
titreAccueil.setTextFormat(styleTitreAccueil);

addChild(titreAccueil);
```

Afin qu'il n'y ait aucune ambiguïté, précisons que le nom Century Gothic est celui de la police qui figure dans la liste des polices accessibles à partir de la palette Propriétés.

Mettre en forme un texte en HTML

Avant d'apprendre à mettre en forme un texte grâce à l'utilisation de balises HTML, nous devons préciser un point très important. À travers tous les exercices abordés depuis le début de ce chapitre, pour placer un contenu dans une instance de `TextField()` nous avons utilisé la propriété `text`. Nous allons maintenant devoir utiliser celle qui s'intitule `htmlText`.

La technique est extrêmement simple car il suffit de placer une balise HTML correcte dans votre texte.

```
citation.htmlText = "'Un bon maître a ce souci constant : enseigner à se passer
➥de lui.' <b>André GIDE</b>.";
```

Examinez l'exemple suivant où de nombreuses balises ont été insérées pour mettre en forme cette citation.

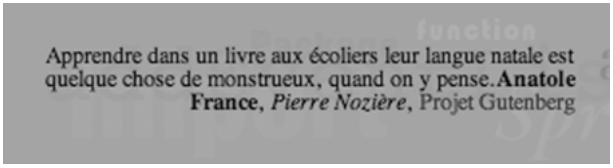


Figure 14-4

Cette mise en forme des caractères a été obtenue à l'aide de balises html uniquement (les deux mots Projet Gutenberg sont en bleu).

Pour aboutir à ce résultat, vous avez le choix entre deux solutions :

- Vous créez le texte dynamiquement sur la scène puis vous insérez votre texte HTML avec les balises nécessaires pour effectuer la mise en forme de certaines parties.
- Vous créez un texte dynamique sur la scène à l'aide de l'outil Texte disponible dans l'interface de Flash, vous effectuez quelques réglages de mise en forme générale (comme le choix de la police) et vous insérez votre texte HTML avec les balises nécessaires pour compléter la mise en forme de certaines parties.

Dans ce dernier cas, vous n'auriez besoin de saisir que la ligne d'instruction suivante :

Fichier de référence : `Chapitre14/texte6.fla`

```
titreAccueil.htmlText = "<p align='justify'>Apprendre dans un livre aux écoliers leur
langue natale est quelque chose de monstrueux, quand on y pense.</p><p align='right'>
<b>Anatole France</b>, <i>Pierre Nozière</i>, <font color='#0000BB'>Projet Gutenberg
</font></p>";
```

Dans le cas où vous souhaiteriez créer votre texte dynamiquement, voici les lignes d'instructions que vous devriez utiliser :

Fichier de référence : `Chapitre14/texte7.fla`

```
var titreAccueil:TextField = new TextField();
titreAccueil.htmlText = "Apprendre dans un livre aux écoliers leur langue natale est quelque chose de monstrueux, quand on y pense.\n<b>Anatole France</b>, <i>Pierre Nozière</i>, <font color='#0000BB'>Projet Gutenberg</font>";
titreAccueil.x = 20;
titreAccueil.y = 20;
titreAccueil.width = 300;
titreAccueil.wordWrap = true;
titreAccueil.multiline = true;

addChild(titreAccueil);
```

Imbriquer des guillemets

Dans le premier exemple que nous avons présenté sur la gestion d'un texte HTML dans un texte dynamique, nous avons été obligé d'utiliser les guillemets simples pour évoquer une citation :

```
'Un bon maître a ce souci constant : enseigner à se passer de lui.' André GIDE.
```

Remarque

Il est possible de placer des guillemets simples à l'intérieur de guillemets doubles et réciproquement : "Il est 'parti' comme il est venu" et 'Il est "parti" comme il est venu'.

Nous n'aurions pas pu placer des guillemets doubles, car cela aurait été interprété comme la fin du texte à ajouter et une erreur se serait produite, comme dans l'exemple ci-dessous :

```
zoneInfo.htmlText = "Il a dit, je cite : "Il faut apprendre à tout faire" pour être
➥autonome.";
1087: Erreur de syntaxe : caractères excédentaires détectés après la fin du programme.
```

En effet, essayez de retirer le texte placé après les guillemets (`Il faut apprendre à tout faire" pour être autonome."`) et vous découvrirez que cela ne pose plus de problème.

Pour pouvoir insérer certains caractères, vous devez donc les échapper avec l'antislash (\) avant de les saisir. Ainsi, dans l'exemple qui va suivre, il n'y a plus de problèmes.

```
zoneInfo.htmlText = "Il a dit, je cite : \"Il faut apprendre à tout faire\" pour être autonome.";
```

Remarque

Sur Macintosh, ce caractère peut être obtenu avec le raccourci clavier Alt-Shift-:

Quelques exemples supplémentaires

Régler la couleur d'un texte

```
zoneInfo.htmlText = "Le <font color='#0000CC'> bleu </font> est une couleur qui te
➥va bien.";
```

ou encore :

```
zoneInfo.htmlText = "Le <font color='#FF0000'> rouge </font> et le <font color='
➥#00FF00'>vert</font>font du jaune.";
```

Régler la police d'un texte

```
zoneInfo.embedFonts = true;
zoneInfo.htmlText = "<font face='Kino MT'> Il est en forme.</font> ";
```

Dans le cas du réglage du choix d'une police, il est fortement conseillé de l'importer dans l'animation. Consultez le développement Encapsuler une police de caractères, dans ce chapitre, pour découvrir la méthode adéquate.

Régler l'alignement d'un texte

```
zoneInfo.htmlText = "<p align='right'>Janvier, Février, Mars</p> ";
zoneInfo.htmlText += "<p align='left'>Avril, Mai, Juin</p> ";
zoneInfo.htmlText += "<p align='right'>Juillet, Août, Septembre</p> ";
zoneInfo.htmlText += "<p align='left'>Octobre, Novembre, Décembre</p> ";
```

Comme vous pouvez le constater, vous avez la possibilité de définir des réglages différents dans un même texte (plusieurs couleurs, plusieurs alignements différents, etc.), ce qui s'avère bien souvent plus pratique qu'avec la classe `TextFormat()`.

Si vous n'êtes pas habitués à manipuler des balises HTML, ce livre ne peut vous aider à acquérir les bases nécessaires : ce n'est pas l'un de ses objectifs. Vous pouvez, dans ce cas, utiliser un moteur de recherche pour découvrir les nombreux sites qui sont dédiés à l'apprentissage de ce langage.

Pour celles et ceux qui connaissent déjà ce langage, voici une liste des balises reconnues par le lecteur Flash :

Tableau 14-1 Balises HTML reconnues par le lecteur Flash.

Balise	Attributs supportés	Remarques
`<b>`		Des caractères en gras doivent être disponibles dans la police utilisée.
`<i>`		Des caractères en italique doivent être disponibles dans la police utilisée.
`<u>`		
`<font>`	`color, face et size`	
`<textformat>`	`blockindent, indent, leading, leftmargin, rightmargin et tabstops`	

Tableau 14-1 Balises HTML reconnues par le lecteur Flash. *(Suite)*

Balise	Attributs supportés	Remarques
` `		Le texte doit avoir l'attribut `multiligne` activé.
`<p>`	`align et class`	
`<li>`		
`<span>`	`class`	La classe de style CSS doit être définie par un objet `flash.text.StyleSheet`.
`<a>`	`href, event et target. a:link, a:hover et a:active`	Vous pouvez utiliser l'événement `link` pour que le lien exécute une fonction ActionScript dans un fichier SWF.
`<img>`	`src, width, height, align, hspace, vspace, id et checkPolicyFile`	

Mettre en forme un texte avec les CSS

Avant toute chose, il est important de savoir que les styles CSS s'appliquent à une instance de type `TextField()` qui contient du texte au format HTML ou XML. La mise en forme d'un texte à partir de styles CSS (*Cascading Style Sheets*) s'avère très intéressante pour plusieurs raisons :

- De nombreux développeurs connaissent déjà les CSS. La consultation, la création et la mise à jour de tels documents s'en trouvent facilitées.
- La mise à jour d'une animation (au niveau de l'aspect du texte) ne nécessite pas une nouvelle publication du fichier FLA d'origine.
- Une même feuille de styles de type CSS peut être partagé entre une page HTML et un fichier SWF.

Dans l'interface de Flash, la gestion des CSS peut se traduire par deux techniques :

- En créant des styles en ActionScript.
- En important une feuille de styles sous forme de fichier externe.

Créer une feuille de styles en ActionScript

Pour démarrer notre apprentissage des CSS, nous devons commencer par placer un texte dynamique sur la scène que nous intitulons `leMessage`. Soit vous utilisez l'outil Texte disponible dans l'IDE de Flash, soit vous saisissez les lignes d'instructions ci-dessous :

```
var leMessage:TextField = new TextField();

leMessage.x = 20;
leMessage.y = 20;
leMessage.width = 280;
```

```
leMessage.wordWrap = true;
leMessage.multiline = true;

addChild(leMessage);
```

À partir de ce moment-là, nous pouvons créer une feuille de styles. La procédure se décompose en deux temps :

- la création de la feuille de styles ;
- la définition d'un style et son rattachement à la feuille de styles.

```
var styleDuProjet:StyleSheet = new StyleSheet();
var styleCorpsDocument:Object = new Object();
```

Le style `styleCorpsDocument` est maintenant créé ; nous devons à présent définir la valeur des attributs que nous souhaitons utiliser.

```
styleCorpsDocument.letterSpacing = 2;
styleCorpsDocument.color = "#474763";
```

Pour terminer cette procédure, nous devons rattacher le style `styleCorpsDocument` à la feuille de styles `styleDuProjet` et faire appel à la propriété `styleSheet` de la classe `TextField()` pour rattacher la feuille de styles `styleDuProjet` à l'instance de type `TextField()` intitulée `leMessage`.

```
styleDuProjet.setStyle("body", styleCorpsDocument);
leMessage.styleSheet = styleDuProjet;
```

Si nous tentons de placer le texte HTML ci-dessous dans notre instance `leMessage`, nous constatons que la mise en page est effective.

```
eMessage.htmlText = "<body>Le texte qui se trouve à cet endroit précis de la scène ne
peut contenir plus de sens qu'il ne possède de signes typographiques. C'est pourquoi,
il est important de comprendre le sens des mots alors que cette phrase n'a pas de
sens.</body>";
```

Remarque

Faites appel à la propriété `styleSheet` avant de définir le contenu HTML de votre instance de type `TextField()`.

Vous devriez obtenir un script global comme celui-ci :

Fichier de référence : `Chapitre14/texte8.fla`

```
var leMessage:TextField = new TextField();

leMessage.x = 20;
leMessage.y = 20;
leMessage.width = 280;
leMessage.wordWrap = true;
leMessage.multiline = true;

addChild(leMessage);
```

```
var styleDuProjet:StyleSheet = new StyleSheet();
var styleCorpsDocument:Object = new Object();

styleCorpsDocument.letterSpacing = 2;
styleCorpsDocument.color = "#474763";

styleDuProjet.setStyle("body", styleCorpsDocument);
leMessage.styleSheet = styleDuProjet;

leMessage.htmlText = "<body>Le texte qui se trouve à cet endroit précis de la scène ne
peut contenir plus de sens qu'il ne possède de signes typographiques. C'est pourquoi,
il est important de comprendre le sens des mots alors que cette phrase n'a pas de
sens.</body>";
```

Si la procédure que nous venons de suivre ensemble vous paraît un peu trop complexe, essayez de consulter le fichier `texte8bis.fla` qui contient moins de lignes d'instructions, dans la mesure où celles qui permettent de créer le texte sur la scène ont été supprimées.

Fichier externe AS

Fichiers de référence : `Chapitre14/texte9.fla et Styles.as`

```
package {

  import flash.display.Sprite;
  import flash.text.TextField;
  import flash.text.StyleSheet;

  public class Styles extends Sprite {

    function Styles() {

      var leMessage:TextField = new TextField();

      leMessage.x = 20;
      leMessage.y = 20;
      leMessage.width = 280;
      leMessage.wordWrap = true;
      leMessage.multiline = true;

      addChild(leMessage);

      var styleDuProjet:StyleSheet = new StyleSheet();
      var styleCorpsDocument:Object = new Object();

      styleCorpsDocument.letterSpacing = 2;
      styleCorpsDocument.color = "#474763";

      styleDuProjet.setStyle("body", styleCorpsDocument);
      leMessage.styleSheet = styleDuProjet;

       leMessage.htmlText = "<body>Le texte qui se trouve à cet endroit précis de la
scène ne peut contenir plus de sens qu'il ne possède de signes typographiques. C'est
pourquoi, il est important de comprendre le sens des mots alors que cette phrase n'a
pas de sens.</body>";

    }
  }
```

Tableau 14-2 Propriétés CSS et ActionScript

Nom de la propriété CSS	Nom de la propriété en ActionScript	Attributs supportés et commentaires
`color`	`color`	Couleur du texte. Valeur sous forme hexadécimale : `#FF0000` (les noms de couleurs ne sont pas supportés).
`display`	`display`	Valeurs supportées : `inline`, `block` et `none`.
`font-family`	`fontFamily`	Liste des polices à utiliser, séparées par des virgules, classées par ordre de choix décroissant. Tous les noms de familles de polices peuvent être utilisés. Si vous spécifiez un nom de police générique, il est converti dans la police de périphérique appropriée. Les conversions de police suivantes sont disponibles : `mono` est converti en `_typewriter`, `sans-serif` en `_sans` et `serif` en `_serif`.
`font-size`	`fontSize`	Taille du texte. Valeur de type nombre entier (par exemple : 12). Ne pas préciser l'unité (pixels ou points) car le lecteur Flash n'en tient pas compte.
`font-style`	`fontStyle`	Style de texte. Valeurs supportées : `normal` et `italic`.
`font-weight`	`fontWeight`	Style de texte. Valeurs supportées : `normal` et `bold`.
`kerning`	`kerning`	Interlettrage par paire de lettres. Valeurs supportées : `true` et `false`. Le crénage est supporté uniquement pour les polices incorporées. Certaines polices, telles que Courier New, ne supportent pas le crénage. La propriété de crénage n'est supportée que dans les fichiers SWF créés dans Windows, pas dans les fichiers SWF créés sur Macintosh. Cependant, ces fichiers SWF peuvent être lus dans des versions de Flash Player autres que Windows et le crénage s'applique encore.
`leading`	`leading`	Interlignage. Valeur de type nombre entier (par exemple : 3). Ne pas préciser l'unité (pixels ou points) car le lecteur Flash n'en tient pas compte. Une valeur négative resserre les lignes. L'espace est placé après chaque ligne.
`letter-spacing`	`letterSpacing`	Interlettrage. Valeur de type nombre entier (par exemple : 3). Ne pas préciser l'unité (pixels ou points) car le lecteur Flash n'en tient pas compte. Une valeur négative resserre les lettres. L'espace est placé après chaque lettre.
`margin-left`	`marginLeft`	Marge de gauche. Valeur de type nombre entier (par exemple : 3). Ne pas préciser l'unité (pixels ou points) car le lecteur Flash n'en tient pas compte.
`margin-right`	`marginRight`	Marge de droite. Valeur de type nombre entier (par exemple : 3). Ne pas préciser l'unité (pixels ou points) car le lecteur Flash n'en tient pas compte.
`text-align`	`textAlign`	Alignement du texte. Valeurs supportées : `left`, `center`, `right` et `justify`.
`text-decoration`	`textDecoration`	Soulignement du texte. Valeurs supportées : `none` et `underline`.
`text-indent`	`textIndent`	Décalage de la première ligne d'un paragraphe. Valeur de type nombre entier (par exemple : 3). Ne pas préciser l'unité (pixels ou points) car le lecteur Flash n'en tient pas compte.

Travailler avec des classes

Dans l'exemple précédent, nous avons redéfini la balise `body`, mais nous aurions tout autant pu en redéfinir d'autres. Il existe également une autre solution qui consiste à créer un sélecteur de type classe.

Voilà comment nous pourrions adapter l'exemple précédent pour qu'il gère une classe.

Fichier de référence : `Chapitre14/texte10.fla`

```
var styleDuProjet:StyleSheet = new StyleSheet();
var styleMotsCles:Object = new Object();

styleMotsCles.letterSpacing = 2;
styleMotsCles.color = "#BB0000";

styleDuProjet.setStyle(".motCle", styleMotsCles);
leMessage.styleSheet = styleDuProjet;

leMessage.htmlText = "Le texte qui se trouve à <span class='motCle'>cet endroit
précis</span> de la scène ne peut contenir plus de sens qu'il ne possède de <span
class='motCle'>signes typographiques</span>. C'est pourquoi, il est important de
comprendre le sens des mots alors que cette phrase n'a pas de sens";
```

À la lecture de ce script, quelle différence doit-on faire avec la redéfinition d'une balise ? Observez bien la méthode `setStyle()`. Son premier paramètre possède un point pour signifier qu'il s'agit d'une classe. La deuxième et dernière différence se situe au niveau du balisage du texte HTML. Nous avons fait appel à l'attribut `class` pour lequel nous avons indiqué le nom de la classe `motCle`. Nous utilisons la balise `<span>` afin de définir des zones dans le texte.

Importation d'une feuille de styles sous forme de fichier externe

Fichiers de référence : `Chapitre14/texte11.fla` et `styles.css`

Comme nous l'évoquions en introduction à ce développement dédié à la mise en forme à base de styles CSS, il est possible de charger un fichier externe au lieu de créer un objet pour lequel vous définissez des valeurs d'attributs. Voilà comment vous devez procéder.

> **Remarque**
>
> Pour simplifier la compréhension du script qui va suivre, nous avons allégé le code en supprimant les lignes d'instructions chargées de créer le texte dynamique sur la scène. Nous avons directement placé un texte sur la scène que nous avons nommé `leMessage`.

1. Déclarez les instances des classes `StyleSheet()`, `URLLoader()` et `URLRequest()`.Ces deux dernières sont initialisées au moment de la déclaration.

```
var styleDuProjet:StyleSheet;
var chargeur:URLLoader = new URLLoader();
var adresseCSS:URLRequest = new URLRequest("styles.css");
```

Le fichier `styles.css` contient les lignes suivantes :

```
body {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  font-size: 12px;
  color: 474763;
}
.citation {
  font-style: italic;
  color: #FF0000;
}
```

2. Effectuez la demande de chargement et enregistrez un écouteur.

```
chargeur.load(adresseCSS);
chargeur.addEventListener(Event.COMPLETE, cssChargees);
```

3. Nous terminons cette procédure en définissant la fonction de rappel `cssChargees`.

```
function cssChargees(evt:Event) {
}
```

Cette fonction contient tout le code qui va être chargé de :

- Initialiser la classe `StyleSheet()`.
- Placer le contenu du fichier CSS dans l'instance de la classe `StyleSheet()`.
- Appeler la propriété `styleSheet` qui rattache la feuille de styles à l'instance `leMessage` (le texte dynamique sur la scène).

```
  styleDuProjet = new StyleSheet();
  styleDuProjet.parseCSS(evt.target.data);
  leMessage.styleSheet = styleDuProjet;
```

À présent, vous pouvez placer le texte HTML qui contient la balise `<body>` que nous avons redéfinie, ainsi que la classe `citation`.

Le script global devrait ressembler à cela :

```
var styleDuProjet:StyleSheet;
var chargeur:URLLoader = new URLLoader();
var adresseCSS:URLRequest = new URLRequest("styles.css");
chargeur.load(adresseCSS);
chargeur.addEventListener(Event.COMPLETE, cssChargees);

function cssChargees(evt:Event) {
  styleDuProjet = new StyleSheet();
  styleDuProjet.parseCSS(evt.target.data);

  leMessage.styleSheet = styleDuProjet;

   leMessage.htmlText = "<body>Le texte qui se trouve à <span class='citation'>cet
endroit précis</span> de la scène ne peut contenir plus de sens qu'il ne possède de
<span class='citation'>signes typographiques</span>. C'est pourquoi, il est important
de comprendre le sens des mots alors que cette phrase n'a pas de sens</body>";

}
```

Gérer les événements liés au texte

Remarque

Le chapitre 3 aborde un exemple différent de ceux que vous allez découvrir dans les lignes ci-dessous. Un fichier avec une approche orientée objet y est également proposé.

Comme nous l'évoquions au chapitre 3, lorsque vous aurez besoin de contrôler le focus d'un texte de saisie, sachez qu'il existe trois états. Redéfinissons les :

- L'instant ou le champ prend le focus, c'est-à-dire le moment où l'utilisateur clique dans le texte de saisie (`FocusEvent.FOCUS_IN`).
- L'instant ou l'utilisateur change le contenu du champ, c'est-à-dire au moment de l'ajout ou de la suppression d'un caractère (`Event.CHANGE`).
- L'instant ou le champs perd le focus, c'est-à-dire le moment où l'utilisateur clique en dehors du texte de saisie (`FocusEvent.FOCUS_OUT`).

Abordons tout de suite un premier exemple. Lorsque l'utilisateur clique dans un texte de saisie, nous aimerions vider le contenu actuel.

```
var adresseMail:TextField = new TextField();

adresseMail.border = true;
adresseMail.type = TextFieldType.INPUT;
adresseMail.text="Veuillez saisir votre adresse mail";

addChild(adresseMail);

adresseMail.addEventListener(FocusEvent.FOCUS_IN,verifierActivationChamps);

function verifierActivationChamps(evt:FocusEvent) {
  adresseMail.text="";
}
```

Remarque

Un exemple similaire mais plus complet est présenté dans le développement Changer la casse d'un texte, dans ce chapitre.

Dans ce deuxième exemple, nous allons chercher à afficher sur la scène, au cours de la frappe, le nombre de caractères contenus dans un texte de saisie.

```
var message_txt:TextField = new TextField();
var compteur:TextField = new TextField();

message_txt.border = true;
message_txt.type = TextFieldType.INPUT;
message_txt.width= 200;
```

```
addChild(message_txt);
addChild(compteur);
compteur.x=210;

message_txt.addEventListener(Event.CHANGE,compterNbreCaracteres);

function compterNbreCaracteres(evt:Event) {
  compteur.text=message_txt.text.length.toString();
}
```

Apportons quelques commentaires à ce script un peu long.

Nous commençons par créer deux instances de la classe `TextField()` car nous avons besoin de deux zones de texte : une première sert à la saisie du texte, l'autre permet d'afficher le nombre de caractères contenus dans le champ de saisie.

Nous configurons et ajoutons ces deux instances et, enfin, nous créons un objet d'écoute qui fait appel à l'événement `Event.CHANGE`.

Nous sommes obligés d'écrire `message_txt.text.length.toString()` pour les raisons suivantes :

- `toString()` permet de convertir `message_txt.text.length` en une chaîne de caractères.
- La propriété `length` permet d'obtenir le nombre de caractères contenus dans le texte de saisie.
- La propriété `text` permet de faire référence au texte contenu dans l'instance `message_txt`.

Ce script est particulièrement intéressant car il suffit de changer le contenu de la fonction de rappel `compterNbreCaracteres` pour l'adapter à n'importe quel besoin de surveillance d'une saisie d'un texte.

Dans ce troisième et dernier exemple, lorsque l'utilisateur clique en dehors de la zone de texte de saisie, le programme ajoute l'extension `.txt` à la fin du texte.

```
var nomFichier:TextField = new TextField();

nomFichier.border = true;
nomFichier.type = TextFieldType.INPUT;
nomFichier.width= 200;

addChild(nomFichier);

nomFichier.addEventListener(FocusEvent.FOCUS_OUT,verifierExtension);

function verifierExtension(evt:FocusEvent) {
  if (nomFichier.text.substr(nomFichier.length-4,4)!=".txt") {
    nomFichier.appendText(".txt");
  }
}
```

Pour être plus précis, ce script devrait d'abord s'assurer qu'une autre extension n'existe pas déjà, ce que nous n'avons pas écrit, pour ne pas complexifier davantage ces quelques lignes d'instructions.

Contrôler le défilement d'un texte

Défilement vertical

Si vous ne souhaitez pas utiliser un composant pour contrôler le défilement d'un texte, vous pouvez faire appel aux propriétés `scrollV` et `scrollH` de la classe `TextField()`. Elles permettent non seulement de faire défiler des lignes comme vous le feriez avec un ascenseur habituel, mais également de réaliser un ascenseur horizontal.

L'exemple ci-dessous permet de faire défiler les mois de l'année. Deux occurrences de type clip ont préalablement été placées sur la scène, puis nommées `btAvant` et `btApres`.

Nous utilisons la propriété `scrollV` pour faire défiler le texte, mais en ajoutant les lignes une à une. Nous pouvons écrire indifféremment, `listeDeMois.scrollV = listeDesMois.scrollV+1` ou `listeDesMois.scrollV++`

```
var moisAnnee:Array = ["Janvier","Février","Mars","Avril","Mai","Juin","Juillet",
➥"Août","Septembre","Octobre","Novembre","Décembre"];
var listeDesMois:TextField = new TextField();
for each (var entree:String in moisAnnee) {
  listeDesMois.appendText(entree+"\n");
}

listeDesMois.width= 200;
listeDesMois.height= 70;
listeDesMois.selectable = false;

addChild(listeDesMois);
listeDesMois.scrollV=5;

btAvant.addEventListener(MouseEvent.MOUSE_DOWN,baisser);
btApres.addEventListener(MouseEvent.MOUSE_DOWN,monter);

function baisser(evt:MouseEvent) {
  listeDesMois.scrollV++;
}

function monter(evt:MouseEvent) {
  listeDesMois.scrollV--;
}
```

Si nous souhaitions remonter ou abaisser le texte de plusieurs lignes, nous remplacerions `listeDesMois.scrollV++` par `listeDesMois.scrollV+=3`. Pour remonter ou abaisser le texte à une ligne précise, nous pourrions utiliser la ligne d'instruction suivante :

```
listeDesMois.scrollV=5;
```

Rappel

En assignant la valeur `0` à la propriété `scrollV`, vous replacez la première ligne du texte en haut de la zone de texte de saisie.

Défilement horizontal

Dans ce deuxième et dernier exemple, nous faisons défiler un texte de droite à gauche ; nous n'utilisons plus la propriété `scrollV`, mais `scrollH`.

```
var messageDefilant:TextField = new TextField();

messageDefilant.text="Le message qui se trouve dans cette zone sert à vous démontrer
➥qu'il est possible de procéder à un défilement horizontal.";

messageDefilant.width= 200;
messageDefilant.height= 30;

addChild(messageDefilant);

messageDefilant.addEventListener(Event.ENTER_FRAME,defiler);

function defiler(evt:Event) {
  messageDefilant.scrollH++;
}
```

Pour obtenir un affichage qui fait disparaître complètement le texte, vous pouvez insérer des espaces après le dernier caractère de votre chaîne. Vous pouvez effectuer la même opération avant le premier mot afin de faire apparaître progressivement le texte. Le nombre d'espaces dépend de la largeur de votre texte dynamique (ou de saisie).

Gestion des tabulations

La gestion d'une telle mise en page nécessite une préparation du texte à formater : il faut placer des séquences d'échappement (`\t`) entre chaque partie de colonne, pour insérer des tabulations. Vous devez ensuite instancier la classe `TextFormat()`.

Avant de vous présenter le script à générer, voici le résultat attendu :

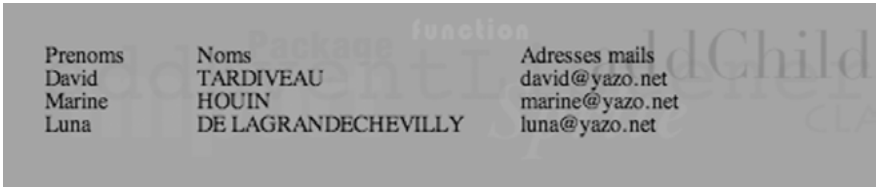


Figure 14-5

Cette mise en forme du texte a été réalisée à l'aide de tabulations et d'une instance de la classe TextFormat.

Comme nous venons de l'évoquer, nous devons commencer par placer du contenu dans un texte dynamique (ou de saisie) ; nous utilisons donc la méthode `appendText()` qui est conseillée par Adobe. Pour séparer les différentes informations qui doivent figurer dans les colonnes, nous utilisons la séquence d'échappement `\t`. Pour finir, nous instancions la classe `TextFormat()` afin de définir les valeurs du tableau assigné à la propriété `tabStops`.

```
var tableauDonnees:TextField = new TextField();

tableauDonnees.appendText("Prenoms\tNoms\tAdresses mails\n");
tableauDonnees.appendText("David\tTARDIVEAU\tdavid@yazo.net\n");
tableauDonnees.appendText("Marine\tHOUIN\tmarine@yazo.net\n");
tableauDonnees.appendText("Luna\tDE LAGRANDECHEVILLY\tluna@yazo.net\n");

tableauDonnees.width= 400;
tableauDonnees.height=300;

addChild(tableauDonnees);

var styleTableau:TextFormat = new TextFormat();
styleTableau.tabStops = [20,80,250];


tableauDonnees.setTextFormat(styleTableau);
```

Remarque

Les valeurs définies dans le tableau correspondent à une mesure précise, le pixel.

Détecter le numéro d'une ligne cliquée

Pour gérer des listes de mots ou de phrases comme dans l'exemple de la figure 14-6, il existe un composant intitulé `List` qui est déjà préprogrammé et livré avec une multitude de propriétés et de méthodes. Comme vous pouvez le constater, il possède un ascenseur par défaut ainsi qu'une surbrillance. Il est tout à fait possible de le configurer pour qu'il n'affiche que le texte et ressemble alors à une instance de type `TextField`.

Figure 14-6

Le composant List permet un défilement du texte et une reconnaissance facile du clic sur une ligne.

Remarque

Consultez le chapitre 15 qui traite de ce composant.

Revenons à notre propos principal, pour découvrir la méthode permettant de détecter le clic sur une ligne précise d'une occurrence de type `TextField`.

Vous allez devoir utiliser la méthode `getLineIndexAtPoint()` qui renvoie, comme son nom l'indique, le numéro de ligne cliquée. Pour cela, vous devez préciser le nom d'une instance de type `TextField` et des coordonnées `x` et `y` du clic. Nous utilisons généralement celles de la souris. Précisons que la première ligne d'un texte renvoie toujours la valeur 0.

Dans les deux exemples ci-dessous, nous aimerions connaître le numéro de ligne cliquée pour pouvoir l'utiliser en tant qu'index d'un tableau. Dans un souci de simplification, nous n'avons pas utilisé un fichier XML, ce qui aurait été plus judicieux et aurait permis de stocker davantage d'informations.

Voici un premier exemple qui présente les lignes d'instructions minimales et nécessaires pour détecter une ligne cliquée. Deux textes dynamiques intitulés `listeParticipants` et `ageParticipant` ont été créés sur la scène.

Fichier de référence : `Chapitre14/texte4bis.fla`

```
var numeroLigneCliquee:Number;
var anneesNaissance:Array = new Array(1945,1963,1902,1967,1965,2006,2002);

function connaitreNumeroLigneCliquee(evt:MouseEvent) {
  numeroLigneCliquee = listeParticipants.getLineIndexAtPoint(listeParticipants.mouseX,
  ➥listeParticipants.mouseY);
  ageParticipant.text = anneesNaissance[numeroLigneCliquee];
}

listeParticipants.addEventListener(MouseEvent.MOUSE_DOWN,connaitreNumeroLigneCliquee);
```

Apportons quelques commentaires : nous commençons par déclarer deux variables sur les deux premières lignes du script. La première va stocker un nombre correspondant au numéro de ligne cliquée, la deuxième contient un tableau dont les valeurs correspondent à des années de naissance. La première entrée du tableau sera associée à la première ligne de notre liste de personnes. Nous créons ensuite un constructeur intitulé `connaitreNumero-LigneCliquee` que nous allons exécuter lorsque l'utilisateur cliquera sur la liste de noms. Justement, lorsqu'un clic se produira, le numéro de la ligne cliquée servira d'index pour rechercher une entrée du tableau.

Voici un deuxième exemple qui gère et génère le contenu de la scène dynamiquement.

Fichier de référence : `Chapitre14/texte4.fla`

```
var listeParticipants:TextField = new TextField();
var ageParticipant:TextField = new TextField();
var numeroLigneCliquee:Number;
var participants:Array = new Array("Robert LEDOUX","Eric DURAND","Jean DUPUIS",
➥"Eva MARTIN","Luc GRANDJEAN","Luna DUPOND","Jules OUVILLE");
var anneesNaissance:Array = new Array(1945,1963,1902,1967,1965,2006,2002);

for each (var nom:String in participants) {
  listeParticipants.appendText(nom+"\n");
```

```
}
addChild(listeParticipants);
addChild(ageParticipant);
ageParticipant.x=120;

listeParticipants.selectable = false;

function connaitreNumeroLigneCliquee(evt:MouseEvent) {
  numeroLigneCliquee = listeParticipants.getLineIndexAtPoint(listeParticipants.mouseX,
  ➥listeParticipants.mouseY);
  ageParticipant.text = anneesNaissance[numeroLigneCliquee];
}

listeParticipants.addEventListener(MouseEvent.MOUSE_DOWN,connaitreNumeroLigneCliquee);
```

En comparaison de l'exemple précédent, nous avons simplement généré dynamiquement le contenu qui avait été créé manuellement à l'aide de l'interface du logiciel.

Récapitulatif des propriétés des classes TextField() et TextFormat()

Voici un rappel sur les différents types utilisés en programmation.

- `String` : chaîne de caractères contenant des lettres et des nombres ;
- `Number` : nombre ;
- `Boolean` : `true` ou `false` ;
- `Array` : un tableau créé avec la classe éponyme ou avec une série de valeurs entre crochets `[]` ;
- `int` : nombre entier compris entre -2 147 483 648 et 2 147 483 647 ;
- `uint` : nombre entier non signé compris entre 0 et 4 294 967 295.

`StyleSheet` et `TextFormat` signifient que les valeurs attendues sont des instances de ces classes.

Les tableaux suivants détaillent le sens de chaque propriété. Certaines descriptions sont celles de l'aide officielle de Flash, d'autres ont été reformulées.

Tableau 14-3 Propriétés de la classe TextField()

Nom de la propriété	Type attendu	Description
`alwaysShowSelection`	`Boolean`	Lorsque cette propriété est définie sur `true` et si le champ texte n'a pas le focus, Flash Player sélectionne le contenu du champ texte en gris.
`antiAliasType`	`String`	Type d'anti-aliasing appliqué à ce champ texte.
`autoSize`	`String`	Redimensionne un texte dynamique ou de saisie. Un alignement est également effectué en fonction du paramètre passé à cette propriété.
`background`	`Boolean`	Active/désactive le remplissage de la couleur d'arrière-plan du texte.
`backgroundColor`	`uint`	Couleur de l'arrière-plan du champ texte.
`border`	`Boolean`	Masque ou affiche un contour du texte.
`borderColor`	`uint`	Couleur du contour du texte.
`bottomScrollV`	`int`	Nombre qui indique la ligne la plus basse visible dans le champ texte spécifié. Cette propriété peut être lue, mais pas modifiée.
`caretIndex`	`int`	Permet de connaître la position du point d'insertion (*caret*). Cette propriété peut être lue, mais pas modifiée.
`condenseWhite`	`Boolean`	Valeur booléenne qui indique si l'espace blanc supplémentaire (espace, saut de ligne, etc.) d'un champ texte HTML doit être supprimé.
`defaultTextFormat`	`TextFormat`	Spécifie le format appliqué par défaut au texte qui vient d'être inséré, tel que le texte inséré avec la méthode `replaceSelectedText()` ou le texte entré par un utilisateur.
`displayAsPassword`	`Boolean`	Active ou désactive l'affichage d'étoiles au lieu de la saisie réelle du texte.
`embedFonts`	`Boolean`	Active ou désactive l'encapsulage d'une police au fichier SWF.
`gridFitType`	`String`	Type d'adaptation à la grille appliqué à ce champ texte.
`htmlText`	`String`	Permet de définir le texte au format HTML qui doit être placé dans l'instance (de type `TextField()`).
`length`	`int`	Permet de connaître le nombre de caractères contenus dans un texte. Cette propriété peut être lue, mais pas modifiée.
`maxChars`	`int`	Le nombre maximal de caractères que le champ texte peut contenir, tels que saisis par un utilisateur.
`maxScrollH`	`int`	Permet de connaître le nombre maximal de pixels qui peuvent défiler avec la propriété `scrollV`. Cette propriété peut être lue, mais pas modifiée.
`maxScrollV`	`int`	Permet de connaître le nombre maximal de lignes de texte qui peuvent défiler avec la propriété `scrollH`. Cette propriété peut être lue, mais pas modifiée.
`mouseWheelEnabled`	`Boolean`	Active ou désactive la possibilité d'utiliser la molette de la souris pour faire défiler le texte.
`multiline`	`Boolean`	Indique si le champ texte est un texte multiligne.
`numLines`	`int`	Définit le nombre de lignes de texte d'un champ multiligne. Cette propriété peut être lue, mais pas modifiée.

Tableau 14-3 Propriétés de la classe TextField() *(Suite)*

Nom de la propriété	Type attendu	Description
`restrict`	`String`	Permet de préciser les caractères autorisés et interdits lors de la saisie du texte par l'utilisateur.
`scrollH`	`int`	Permet de positionner le texte horizontalement par rapport au cartouche dans lequel il se trouve.
`scrollV`	`int`	Permet de positionner le texte verticalement par rapport au cartouche dans lequel il se trouve.
`selectable`	`Boolean`	Indique si le champ texte peut être sélectionné.
`selectionBeginIndex`	`int`	Valeur d'index, basée sur zéro, du premier caractère de la sélection en cours. Cette propriété peut être lue, mais pas modifiée.
`selectionEndIndex`	`int`	Valeur d'index, basée sur zéro, du dernier caractère de la sélection en cours. Cette propriété peut être lue, mais pas modifiée.
`sharpness`	`Number`	Netteté des bords du glyphe dans ce champ texte.
`styleSheet`	`StyleSheet`	Associe une feuille de styles au champ texte.
`text`	`String`	Permet de définir le texte qui doit être placé dans l'instance (de type `TextField()`).
`textColor`	`uint`	Couleur du texte dans un champ texte, au format hexadécimal.
`textHeight`	`Number`	Hauteur du texte en pixels. Cette propriété peut être lue, mais pas modifiée.
`textWidth`	`Number`	Largeur du texte en pixels. Cette propriété peut être lue, mais pas modifiée.
`thickness`	`Number`	Épaisseur des bords du glyphe dans ce champ texte.
`type`	`String`	Type de texte. `dynamic` ou `input` (texte de saisie)
`useRichTextClipboard`	`Boolean`	Spécifie si le formatage du texte peut être copié et collé en même temps que le texte.
`wordWrap`	`Boolean`	Indique si le texte peut revenir à la ligne lorsqu'il atteint le bord droit des limites de l'instance dans laquelle il se trouve.

Tableau 14-4 Propriétés de la classe TextFormat()

Nom de la propriété	Type attendu	Description
`bold`	`Boolean`	Texte en gras.
`italic`	`Boolean`	Texte en italique.
`underline`	`Boolean`	Texte souligné. Ne mélangez pas un texte en gras avec un soulignement.
`bullet`	`Boolean`	Texte avec une puce devant.
`color`	`Hexa ou uint`	Couleur. Valeur exprimée en hexadécimal (0x000000) ou avec un entier.
`font`	`String`	Police de caractère.
`size`	`Number`	Taille du caractère.
`tabStops`	`Array`	Tableau comprenant les arrêts de tabulation.
`align`	`String`	Alignement du paragraphe : `left`, `right`, `center` ou `justify`
`leading`	`Number`	Espacement entre les lignes.
`letterSpacing`	`Number`	Espacement entre les caractères. Attention de ne pas abuser de ce réglage avec de fortes valeurs : le texte devient très rapidement illisible.
`kerning`	`Boolean`	Activer le crénage, c'est-à-dire le rapprochement de paires de lettres ou encore l'espacement entre certaines paires de lettres. À n'utiliser que pour des textes avec des grands corps et de préférence en majuscule. Ne fonctionne pas avec certaines polices de caractères (comme Verdana).
`leftMargin`	`Number`	Espacement de la marge de gauche.
`rightMargin`	`Number`	Espacement de la marge de droite.
`blockIndent`	`Number`	Retrait du paragraphe. Valeur exprimée en pixels. Revient à utiliser `leftMargin` s'il n'y a qu'un seul paragraphe.
`indent`	`Number`	Retrait de la première ligne de base. Valeur exprimée en pixels.
`target`	`String`	La fenêtre dans laquelle doit s'ouvrir le lien spécifié par la propriété `url`.
`url`	`String`	Lien vers une adresse Web.

15

Les composants de type formulaire

Un composant est un symbole préprogrammé qui offre à l'utilisateur une interface de configuration de son contenu initial et de son apparence. Il suffit d'en glisser un sur la scène à partir de la palette Composants, rubrique User Interface, pour découvrir que la palette Paramètres vous propose de renseigner quelques champs pour terminer de configurer l'occurrence de votre composant. Sa configuration est très simple, mais sa programmation l'est un peu moins.

Ce chapitre n'est pas uniquement dédié à l'apprentissage de modules destinés à réaliser des formulaires. Les composants, qualifiés bien souvent d'éléments de formulaires, servent à réaliser toutes sortes d'interfaces et pas uniquement des écrans comprenant des moyens de saisie pour évaluer les réponses d'un utilisateur.

Parmi les nombreux composants proposés dans la palette éponyme, nous avons retenu une sélection de ceux qui sont le plus souvent utilisés. Quel que soit celui que vous choisirez, vous constaterez qu'il existe deux possibilités de création d'une occurrence : soit un glisser-déposer, suivi d'une configuration via la palette Paramètres ; soit des lignes d'instructions. C'est cette dernière solution que nous retiendrons pour chacun des composants. Avant d'aborder chaque spécificité des composants de ce chapitre, il est important de garder à l'esprit que la gestion d'un tel symbole se décompose en trois parties :

- la création du composant (instanciation de la classe) ;
- le remplissage et la configuration du composant ;
- la programmation du composant.

Le composant ComboBox

Fichier de référence : `Chapitre15/ComboBox.fla`

Il s'agit certainement du composant que vous utiliserez le plus souvent, car il permet de sélectionner simplement et rapidement une information dans un menu déroulant.

Pour commencer, voici à quoi ressemble un ComboBox.

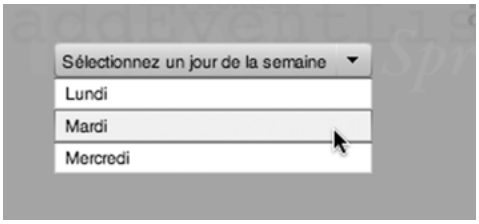


Figure 15-1

Une occurrence de type ComboBox permet de proposer à l'utilisateur un menu déroulant pour y sélectionner une information.

Création d'une instance

Avant de faire appel à la classe `ComboBox()`, vous devez commencer par placer un composant de ce type sur la scène, afin qu'il se retrouve dans la Bibliothèque. Une fois cette manipulation effectuée, vous pouvez garder cette instance ou bien la supprimer pour exécuter la ligne d'instruction ci-dessous.

```
var semaine_comp:ComboBox = new ComboBox();
```

Afin que les membres de cette classe soient accessibles, vous devrez préalablement importer cette dernière.

```
import fl.controls.ComboBox;
```

Vous aurez également besoin de gérer le contenu du ComboBox. Vous devez donc importer la classe `DataProvider`.

```
import fl.data.DataProvider;
```

Une fois la classe `ComboBox` instanciée, et avant de définir son contenu, vous devez configurer son apparence et sa position sur la scène. Utilisez alors les lignes d'instructions suivantes :

```
semaine_comp.dropdownWidth = 200;
semaine_comp.width = 200;
semaine_comp.move(50, 50);
```

La propriété `dropdownWidth` permet de définir la largeur (de la partie basse) du menu, lorsqu'il est déroulé, alors que `width` détermine la largeur de la partie haute du menu, celle qui affiche le texte sélectionné. Vous ajoutez ensuite le composant dans la liste

d'affichage avec la méthode `addChild()`, afin qu'il soit visible sur la scène. Vous devriez obtenir pour l'instant le script ci-dessous :

```
import fl.controls.ComboBox;
import fl.data.DataProvider;

var semaine_comp:ComboBox = new ComboBox();
semaine_comp.dropdownWidth = 200;
semaine_comp.width = 200;
semaine_comp.move(50, 50);
addChild(semaine_comp);
```

À ce niveau-là de la construction de l'occurrence, le menu est vide. Voici comment le remplir.

Le remplissage et la configuration

Comme nous l'évoquions dans le point précédent, vous allez créer une instance de la classe `DataProvider()` afin de définir ses valeurs sous la forme `{label:"Nom de la commande", data:"Valeur"}`. Le label est le texte visible dans le menu, sous forme de commande, alors que la donnée (`data`) correspond à l'information récupérée lorsque le label associé est sélectionné dans le menu.

```
var semaineNoms:DataProvider = new DataProvider();
```

Pour ajouter une commande au menu, vous devez faire appel à la méthode `addItem()` et passer en paramètres un nom d'étiquette et la valeur associée.

```
semaineNoms.addItem({label:"Lundi",data:"Monday"});
```

Vous pouvez ajouter ainsi de nombreuses commandes dans le menu en répétant cette dernière ligne d'instruction. Précisons tout de même qu'il serait plus judicieux de créer un document XML ou un tableau pour les ajouter à l'aide d'une boucle `for()`. En attendant, de façon très simple, voilà comment obtenir un menu suffisamment rempli :

```
var semaineNoms:DataProvider = new DataProvider();
semaineNoms.addItem({label:"Lundi",data:"Monday"});
semaineNoms.addItem({label:"Mardi",data:"Tuesday"});
semaineNoms.addItem({label:"Mercredi",data:"Wenedsay"});
```

N'oubliez pas de terminer le remplissage de votre menu par l'exécution de la ligne d'instruction ci-dessous :

```
semaine_comp.dataProvider = semaineNoms;
```

Elle permet de placer le contenu de votre instance de type `DataProvider` dans l'instance de votre ComboBox pour constituer le menu déroulant.

Nous terminerons cette deuxième étape en ajoutant une dernière ligne d'instruction qui permet d'afficher un texte d'accueil tant que le menu n'a pas été déroulé.

```
semaine_comp.prompt = "Sélectionnez un jour de la semaine";
```

La programmation d'une occurrence

Vous n'avez pas encore programmé l'instance de votre ComboBox : celle-ci ne vous sert donc à rien !

Le script ci-dessous doit être placé à la suite de celui que nous venons de créer pour obtenir une instance de type `ComboBox`.

```
semaine_comp.addEventListener(Event.CHANGE, jourSelectionne);

function jourSelectionne(evt:Event):void {
  trace(evt.target.selectedItem.data);
}
```

Nous faisons référence à l'instance de type `ComboBox` qui s'intitule `semaine_comp` pour enregistrer un écouteur. Nous utilisons l'événement `Event.CHANGE` pour pouvoir exécuter la ligne d'instruction contenue dans la fonction de rappel lorsqu'une commande du menu déroulant sera sélectionnée.

Dans un souci de simplification, l'objectif de cette animation est d'afficher un mot (l'un des jours de la semaine en anglais) dans la fenêtre Sortie de l'IDE de Flash. Nous exécutons donc la commande `trace()` en demandant de lire la valeur de la propriété `data` de l'occurrence utilisée pour la sélection d'un jour.

Rappelons que, dans la ligne d'instruction `trace(evt.target.selectedIndex)`, `evt.target` fait référence à l'occurrence utilisée. De ce fait, vous pourriez ajouter sur la scène autant d'instances de ComboBox que souhaité ; vous n'auriez qu'à enregistrer un écouteur supplémentaire sans être obligé de redéfinir la fonction de rappel (sauf si elle doit effectuer une action différente pour chaque instance).

```
semaine_comp.addEventListener(Event.CHANGE, commandeSelectionnee);
mois_comp.addEventListener(Event.CHANGE, commandeSelectionnee);
annee_comp.addEventListener(Event.CHANGE, commandeSelectionnee);
```

Vous constatez ainsi que la programmation d'un composant de type `ComboBox` est très simple, mais nous terminerons tout de même par un conseil.

Si vous devez manipuler davantage d'informations lors de la sélection d'une commande dans un menu, nous vous conseillons de récupérer non pas la valeur associée à l'étiquette de la commande, mais plutôt le numéro d'index de la commande. Cela vous permet ensuite de pointer vers un nœud XML, dont le niveau dans l'arborescence ou l'index correspond à celui de la commande.

```
trace(evt.target.selectedIndex);
```

Remarque

Le genre du terme ComboBox est à la fois féminin et masculin car on parle d'un composant de type `ComboBox`, et par raccourci de composant `ComboBox` ou encore de ComboBox. Mais il est souvent fait référence à ce composant comme étant une liste déroulante : on dit alors une ComboBox.

Le composant bouton radio

> **Remarque**
> Placez un composant de type `RadioButton` dans la bibliothèque de l'animation.

Vous allez découvrir qu'il devient très vite fastidieux de gérer plusieurs occurrences de composant de ce type, car les différents réglages à effectuer pour en configurer un sont nombreux.

Afin d'avoir une meilleure visibilité de cette configuration, nous vous proposerons un exemple, à la fin de ce développement, qui ne fait pas appel à une classe mais gère plutôt chaque occurrence à base de programmation séquentielle.

1. Commencez par importer les deux classes nécessaires à la gestion des boutons radio.

```
import fl.controls.RadioButton;
import fl.controls.RadioButtonGroup;
```

2. Comme vous le savez sûrement, les boutons fonctionnent par groupe ce qui permet d'en décocher un lorsque vous cliquer sur un autre. Nous devons donc définir un groupe à l'aide d'une instance de la classe `RadioButtonGroup()`.

```
var categorieAges:RadioButtonGroup = new RadioButtonGroup("group1");
```

3. Créez un bouton radio à l'aide de la classe `RadioButton()`.

```
var btnRadio1:RadioButton = new RadioButton();
```

4. Définissez les propriétés de l'occurrence que vous venez de créer.

```
btnRadio1.group = categorieAges;
btnRadio1.label = "Moins de 18 ans";
btnRadio1.value = "0018";
btnRadio1.width=200;
```

5. Terminez l'étape de création du bouton radio en l'ajoutant à la liste d'affichage et en le positionnant sur la scène.

```
btnRadio1.move(30,120);
addChild(btnRadio1);
```

6. Pour programmer le bouton, afin qu'il réagisse au clic, nous enregistrons un écouteur qui va gérer l'événement `MouseEvent.CLICK`.

```
btnRadio1.addEventListener(MouseEvent.CLICK, clicSurBouton);
function clicSurBouton(event:MouseEvent):void {
  valeurBoutonSelectionne.text= event.currentTarget.value;
}
```

Dans cet exemple, lorsque l'utilisateur clique sur un bouton radio, la valeur qui lui a été affectée s'affiche dans un texte dynamique qui se trouve sur la scène et s'intitule `valeur-BoutonSelectionne`.

Il ne reste plus qu'à répéter toutes ces étapes pour ajouter autant de boutons radio que nécessaire. Comme nous l'évoquions au début de ce développement, il serait préférable de créer une classe pour gérer la création d'un bouton radio avec tous ces paramétrages.

Bouton radio sélectionné par défaut

Utilisez la propriété `selected` pour définir l'une des occurrences comme bouton radio coché par défaut.

Fichier de référence : `Chapitre15/BoutonRadioCaseaCocher.fla`

```
import fl.controls.RadioButton;
import fl.controls.RadioButtonGroup;

var categorieAges:RadioButtonGroup = new RadioButtonGroup("group1");


var btnRadio1:RadioButton = new RadioButton();
var btnRadio2:RadioButton = new RadioButton();
var btnRadio3:RadioButton = new RadioButton();
var btnRadio4:RadioButton = new RadioButton();

btnRadio1.group = categorieAges;
btnRadio2.group = categorieAges;
btnRadio3.group = categorieAges;
btnRadio4.group = categorieAges;

btnRadio1.label = "Moins de 18 ans";
btnRadio2.label = "De 18 à 30 ans";
btnRadio3.label = "De 31 ans à 50 ans";
btnRadio4.label = "51 ans et plus";

btnRadio1.value = "0018";
btnRadio2.value = "1830";
btnRadio3.value = "3150";
btnRadio4.value = "5199";

btnRadio1.move(30,120);
btnRadio2.move(30,150);
btnRadio3.move(30,180);
btnRadio4.move(30,210);

btnRadio1.width=200;
btnRadio2.width=200;
btnRadio3.width=200;
btnRadio4.width=200;

addChild(btnRadio1);
addChild(btnRadio2);
addChild(btnRadio3);
addChild(btnRadio4);

btnRadio2.selected = true;

btnRadio1.addEventListener(MouseEvent.CLICK, clicSurBouton);
btnRadio2.addEventListener(MouseEvent.CLICK, clicSurBouton);
```

```
btnRadio3.addEventListener(MouseEvent.CLICK, clicSurBouton);
btnRadio4.addEventListener(MouseEvent.CLICK, clicSurBouton);

function clicSurBouton(event:MouseEvent):void {
  valeurBoutonSelectionne.text= event.currentTarget.value;
}
```

Le composant ColorPicker

Remarque

Placez un composant de type `ColorPicker` dans la bibliothèque de l'animation.

Ce composant est particulièrement utile dès que vous devez proposer à l'utilisateur un choix de couleurs à partir de l'interface. Il vous évite de reprogrammer une classe détectant la couleur d'un pixel survolé ou attribuée à une occurrence servant de pastille de couleurs.

1. Commencez par importer les deux classes `ColorPicker()` et `ColorPickerEvent()`.

```
import fl.controls.ColorPicker;
import fl.events.ColorPickerEvent;
```

2. Instanciez la classe `ColorPicker()`.

```
var selecteurDeCouleur_comp:ColorPicker = new ColorPicker();
```

3. Ajoutez l'occurrence obtenue à la liste d'affichage de l'animation.

```
addChild(selecteurDeCouleur_comp);
```

4. Positionnez votre occurrence.

```
selecteurDeCouleur_comp.x=2.5;
selecteurDeCouleur_comp.y=250;
```

À présent, il ne reste plus qu'à programmer l'occurrence afin de pouvoir récupérer la valeur de la couleur sélectionnée. Nous allons utilisez l'événement `ColorPickerEvent` et la constante `CHANGE`.

```
 selecteurDeCouleur_comp.addEventListener(ColorPickerEvent.CHANGE, couleurSelectionnee);

function couleurSelectionnee(evt:ColorPickerEvent) {
  trace(evt.currentTarget.hexValue);
}
```

Dans ce premier exemple, nous nous contentons d'afficher le code couleur sélectionné dans la fenêtre Sortie de l'IDE de Flash. Consultez l'exemple du fichier `ColorPicker2.fla` pour découvrir comment nous appliquons la couleur à un texte.

Fichier de référence : `Chapitre15/ColorPicker1.fla`

```
import fl.controls.ColorPicker;
import fl.events.ColorPickerEvent;
```

```
var selecteurDeCouleur_comp:ColorPicker = new ColorPicker();
addChild(selecteurDeCouleur_comp);
selecteurDeCouleur_comp.x=2.5;
selecteurDeCouleur_comp.y=250;

selecteurDeCouleur_comp.addEventListener(ColorPickerEvent.CHANGE, couleurSelectionnee);

function couleurSelectionnee(evt:ColorPickerEvent) {
  trace(evt.currentTarget.hexValue);
}
```

Changer la couleur d'un texte

Si vous ne connaissez pas la classe `TextFormat()` que nous utilisons dans cet exemple, consultez le chapitre 14 qui traite de son utilisation.

Fichier de référence : `Chapitre15/ColorPicker2.fla`

```
import fl.controls.ColorPicker;
import fl.events.ColorPickerEvent;

var selecteurDeCouleur_comp:ColorPicker = new ColorPicker();
addChild(selecteurDeCouleur_comp);
selecteurDeCouleur_comp.x=2.5;
selecteurDeCouleur_comp.y=250;

selecteurDeCouleur_comp.addEventListener(ColorPickerEvent.CHANGE, couleurSelectionnee);

function couleurSelectionnee(evt:ColorPickerEvent) {
  styleTitre.color = "0x"+evt.currentTarget.hexValue;

  messageDeTest.setTextFormat(styleTitre);
}


var styleTitre:TextFormat = new TextFormat();
styleTitre.color = 0x000000;
styleTitre.size = 20;

messageDeTest.setTextFormat(styleTitre);
```

Le composant List

Nous avons souhaité conclure ce chapitre sur le composant `List` pour vous démontrer les nombreuses similitudes avec la classe `ComboBox()`. Cela vous permettra de ne pas avoir à réapprendre le mécanisme de la gestion d'un composant de type `List` qui est le même que celui d'un `ComboBox`. Pour vérifier notre propos, nous vous invitons à ouvrir le fichier `List1.fla` et à procéder aux modifications suivantes (il s'agit des lignes d'instruction suivies d'un commentaire) :

Remarque

N'oubliez pas de placer un composant de type `List` dans la bibliothèque de votre animation.

```
import fl.controls.ComboBox; // Changez ComboBox en List
import fl.data.DataProvider;

var semaine_comp:ComboBox = new ComboBox(); // Changez ComboBox en List
semaine_comp.dropdownWidth = 200; // Placez cette ligne en commentaire
semaine_comp.width = 200;
semaine_comp.move(50, 50);
addChild(semaine_comp);

var semaineNoms:DataProvider = new DataProvider();
semaineNoms.addItem({label:"Lundi",data:"Monday"});
semaineNoms.addItem({label:"Mardi",data:"Tuesday"});
semaineNoms.addItem({label:"Mercredi",data:"Wenedsay"});
semaine_comp.dataProvider = semaineNoms;

semaine_comp.prompt = "Sélectionnez un jour de la semaine"; // Placez cette ligne
                                                              en commentaire

semaine_comp.addEventListener(Event.CHANGE, jourSelectionne);

function jourSelectionne(evt:Event):void {
  trace(evt.target.selectedItem.data);
  //trace(evt.target.selectedIndex)
}
```

Le constat est flagrant : le résultat revient à créer une occurrence de type `List` car ces deux classes héritent des mêmes classes parentes. Voici tout de même un exemple qui contient des propriétés propres à une occurrence de type `List`.

Fichier de référence : `Chapitre15List1.fla`

```
import fl.controls.List;
import fl.data.DataProvider;

var semaine_comp:List = new List();
semaine_comp.width = 200;
semaine_comp.move(50, 50);
addChild(semaine_comp);

var semaineNoms:DataProvider = new DataProvider();
semaineNoms.addItem({label:"Lundi",data:"Monday"});
semaineNoms.addItem({label:"Mardi",data:"Tuesday"});
semaineNoms.addItem({label:"Mercredi",data:"Wenedsay"});
semaineNoms.addItem({label:"Jeudi",data:"Thursday"});
semaineNoms.addItem({label:"Vendredi",data:"Friday"});
semaineNoms.addItem({label:"Samedi",data:"Saterday"});
semaineNoms.addItem({label:"Dimanche",data:"Sunday"});
```

```
semaine_comp.dataProvider = semaineNoms;

semaine_comp.rowCount = 3;
semaine_comp.rowHeight = 15; // 20 pour que rowCount soit juste

semaine_comp.addEventListener(Event.CHANGE, jourSelectionne);

function jourSelectionne(evt:Event):void {
  trace(evt.target.selectedItem.data);
  //trace(evt.target.selectedIndex)
}
```

16

La création de classes personnalisées

Dans le chapitre 1 de ce livre, nous avons expliqué succinctement le fonctionnement de la programmation orientée objet à travers l'interface de Flash, mais nous nous sommes appuyés sur l'utilisation d'une classe du document. Nous n'avons pas eu à instancier une classe en particulier.

Vous serez souvent amené à utiliser des lignes d'instructions qui se trouvent dans de nombreux programmes différents, mais la méthode du copier-coller n'est pas la meilleure solution pour optimiser votre production. Vous allez plutôt devoir créer des classes que vous pourrez ensuite appeler autant de fois que nécessaire.

Prenons l'exemple de l'affichage du temps écoulé lorsque vous écoutez un son ou diffusez une vidéo dans une animation. En aucun cas, vous ne pouvez obtenir l'affichage de la position de la tête de lecture sur la scène (dans un texte dynamique) sous la forme HH:MM:SS (comme 00:02:34 pour 2 minutes et 34 secondes). Vous pouvez connaître le nombre de secondes ou de millisecondes écoulées depuis le début de la lecture d'un son ou d'une vidéo, mais vous devez traiter les valeurs renvoyées par les propriétés `position` (classe `SoundChannel()`) et `playheadTime` (classe `FLVPlayback()`) pour obtenir ce type de formatage.

Créer une classe dans un fichier .as

Fichiers de référence : `Chapitre16/ClassePersonnalisee.fla` et `Chapitre16/Convert.as`

> **Note**
>
> Si vous n'avez pas encore lu, dans le chapitre 1, la partie dédiée à la programmation orientée objet, faites-le avant de poursuivre votre lecture.

1. Commencez par créer un fichier ActionScript que vous nommez `Convert.as`.
2. Saisissez le code ci-dessous dans le fichier `Convert.as`.

```
package {

  import flash.display.Sprite;

  public class Convert {

    var tempsEcoule:Number;
    var minutes:*;
    var secondes:*;
    var tempsFinal:String;

    public function Convert(temps:Number=0) {
      tempsEcoule=temps;
    }

    public function affichageTemps() {
      minutes=Math.floor(tempsEcoule / 60);
      secondes=Math.floor(tempsEcoule % 60);
      minutes=minutes <= 9?"0" + minutes:minutes;
      secondes=secondes <= 9?"0" + secondes:secondes;
      tempsFinal=minutes + ":" + secondes;
      return tempsFinal;
    }
  }
}
```

Analysons ensemble quelques détails de ce script.

- Les premières lignes concernant la structure du document (le package et la classe) ont été expliquées dans la partie dédiée à la programmation orientée objet du chapitre 1 ; nous ne reviendrons donc pas dessus.
- Le type des variables (appelées aussi les propriétés de la classe) `minutes` et `secondes` peut vous surprendre car il s'agit d'une étoile ! Comme vous le savez peut-être, afin de faciliter le déboguage d'un programme et surtout pour éviter des erreurs de typage, nous spécifions le type d'une propriété ; cela se fait via un mot (précédé de deux points) qui indique la nature du contenu stocké dans une variable. L'étoile que nous utilisons dans cet exemple permet d'éviter l'utilisation de deux variables et l'appel de la classe `Number()`, car les propriétés `minutes` et `secondes` vont stocker successivement deux types d'informations de type `Number` puis `String`.

Rappel

Il est très dangereux de typer systématiquement une variable avec l'étoile car cela revient à ne pas typer et entraîne, de ce fait, plus de risques d'erreurs de typage.

- Le constructeur `Convert()` sert uniquement à récupérer puis passer le paramètre indiqué lors de l'instanciation de la classe.

- L'unique méthode `affichageTemps()` de la classe `Convert()` se termine par l'instruction `return` afin de renvoyer le résultat du calcul sous la forme HH:MM:SS (heures:minutes:secondes).

Instancier une classe

Comme nous le précisions en introduction à ce chapitre, nous allons apprendre maintenant à instancier une classe. Pour cela, nous devons faire référence au mot `Convert`, suivi d'une paire de parenthèses.

```
Convert()
```

Commencez par définir un nom d'instance (ou nom de variable) suivi du typage correspondant à la classe que nous allons instancier.

```
var tempsChanson1:Convert
```

Ajoutez ensuite la classe à laquelle vous voulez faire référence. Précisez, par la même occasion, la valeur à passer en paramètres entre les parenthèses. Pour convertir et obtenir l'affichage de trois durées, voici comment procéder.

```
var tempsChanson1:Convert = new Convert(189);
var tempsChanson2:Convert = new Convert(257);
var tempsChanson3:Convert = new Convert(345);

affichageDureeChanson1.text = tempsChanson1.affichageTemps();
affichageDureeChanson2.text = tempsChanson2.affichageTemps();
affichageDureeChanson3.text = tempsChanson3.affichageTemps();
```

Précisons qu'`affichageDureeChanson1` est le nom d'occurrence d'un texte dynamique présent sur la scène.

Remarque

Dans notre exemple, nous convertissons des secondes, mais nous pourrions tout aussi bien convertir des millisecondes ; il faudrait alors adapter le code de votre classe de la façon suivante :

```
minutes=Math.floor(tempsEcoule  / 1000/ 60);
secondes=Math.floor(tempsEcoule / 1000 % 60);
```

Rattacher une classe à un symbole

Fichiers de référence : `Chapitre16/JeuCarte.fla` et `Chapitre16/ModeleCarte.as`

Nous venons d'aborder, dans le développement précédent, la technique qui consiste à créer puis appeler une classe. Nous allons à présent découvrir, dans l'exemple qui suit, qu'un symbole peut posséder sa propre classe sans qu'un appel soit nécessaire.

Vous devez réaliser un jeu de cartes où l'utilisateur doit effectuer des glisser-déposer d'occurrences (les cartes) sur la scène. Afin de ne pas devoir programmer la mobilité de

chaque occurrence du symbole que vous aurez défini comme carte modèle, nous créons un fichier ActionScript avec le contenu ci-dessous :

Script du fichier `ModeleCarte.as` :

```
package {

  import flash.display.Sprite;
  import flash.events.MouseEvent;

  public class ModeleCarte extends Sprite {

    public function ModeleCarte() {
      addEventListener(MouseEvent.MOUSE_DOWN,rendreMobile);
      addEventListener(MouseEvent.MOUSE_UP,arreterMobilite);
    }
    public function rendreMobile(evt:MouseEvent) {
      startDrag();
    }
    public function arreterMobilite(evt:MouseEvent) {
      stopDrag();
    }

  }
}
```

Nous devons ensuite rattacher ce script au symbole, afin que toutes les occurrences issues de ce dernier puissent être déplacées sur la scène.

1. Créez un symbole dans une animation que vous devez enregistrer dans le même dossier que le fichier `ModeleCarte.as`.

   Lorsque vous allez effectuer le raccourci clavier F8 pour convertir votre sélection en symbole, cliquez ensuite sur le bouton Avancé (figure 16-1). Si ce dernier porte le nom d'étiquette Options de base, c'est que vous avez accès à la partie inférieure de la fenêtre qui nous intéresse (figure 16-2).

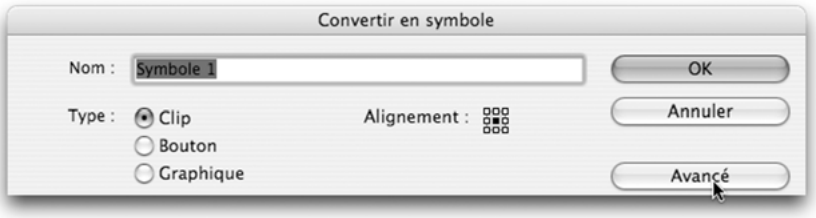


Figure 16-1

Le bouton Avancé permet d'accéder aux réglages de liaison d'un symbole.

2. Cochez la case Exporter pour ActionScript afin d'activer la case de saisie Classe.
3. Saisissez le nom du fichier ActionScript que vous venez de créer.

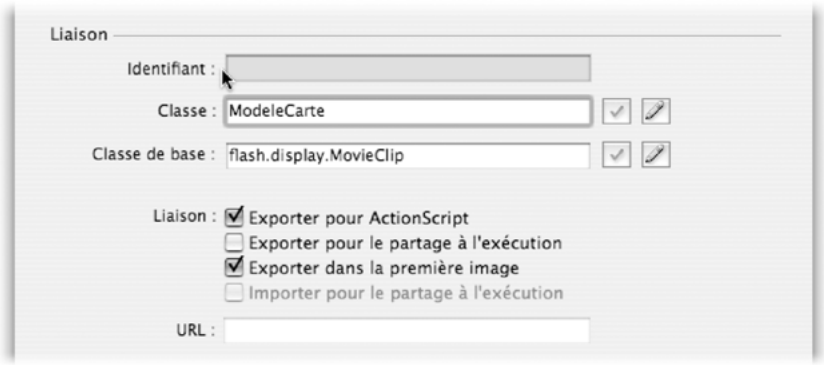


Figure 16-2

Cochez la case Exporter pour ActionScript et précisez le nom de classe souhaité.

4. Validez.

Comme vous pouvez le constater, la procédure est extrêmement simple. Il ne reste plus qu'à analyser les fonctionnalités communes à plusieurs occurrences issues d'un même symbole.

Nous avons utilisé la fonction `addEventListener()` dans la classe `ModeleCarte`. Allons-nous pouvoir la réutiliser dans notre classe du document ?

La réponse est oui ! En effet, nous pourrions penser qu'en redéfinissant un nouveau gestionnaire d'événement dans le fichier principal, la fonction `addEventListener()` pourrait s'annuler. Il n'en est rien, car il est tout à fait possible d'en définir plusieurs, les gestionnaires se cumulent.

Dans l'exemple qui suit, nous allons démontrer deux points :

- L'initialisation d'une occurrence peut être effectuée à partir du fichier de classe d'un symbole (nous souhaitons par exemple que chaque occurrence issue du symbole que nous allons travailler, possède déjà un effet de relief, *emboss*).
- L'attribution d'un gestionnaire d'événement ne constitue pas une redéfinition lorsqu'un gestionnaire précédent a déjà été défini, mais un ajout de gestionnaire d'événement.

Nous avons préparé initialement deux fichiers comme nous l'avions fait dans l'exemple précédent :

- Le fichier `SymboleAvecClasse.fla` possède un symbole dont la classe de liaison est `BoutonReactif`. Deux occurrences intitulées `sommaire` et `quitter` ont également été disposées sur la scène. Nous reviendrons sur ce détail un peu plus loin dans nos explications.
- Le fichier `BoutonReactif.as` possède les lignes d'instructions suivantes :

```
package {

  import flash.display.Sprite;
```

```
  import flash.events.MouseEvent;
  import flash.text.TextField;
  import flash.filters.BevelFilter;

  public class BoutonReactif extends Sprite {

    public function BoutonReactif() {

      var serieFiltres:Array = new Array();
      var biseau:BevelFilter = new BevelFilter();
      biseau.distance = 3;
      biseau.blurX=7;
      biseau.blurY=7;

      serieFiltres.push(biseau);
      filters=serieFiltres;

      nomEtiquette.text=name.toUpperCase();

      addEventListener(MouseEvent.MOUSE_DOWN,clicSurLeBouton);

    }
    public function clicSurLeBouton(evt:MouseEvent) {

      trace("Oui");
    }

  }
}
```

Jusqu'à ce point, la procédure est la même que celle que nous avons exposée dans l'exemple précédent. Ce qui diffère réside dans le fait que nous allons maintenant préciser un nom de classe du document. Nous nous plaçons donc sur le fichier `SymboleAvecClasse.fla` et dans la palette Propriétés (après avoir cliqué en dehors de la scène), nous spécifions le nom suivant : `main`. Voici le contenu du fichier.

```
package {

  import flash.display.Sprite;
  import flash.events.MouseEvent;
  import flash.display.MovieClip;

  public class main extends Sprite {

    public function main() {
      sommaire.addEventListener(MouseEvent.MOUSE_DOWN,actionClic);
      sommaire.addEventListener(MouseEvent.MOUSE_UP,actionRelacherClic);
    }
    public function actionClic(evt:MouseEvent) {
      evt.currentTarget.scaleX=0.8;
      evt.currentTarget.scaleY=0.8;
    }
    public function actionRelacherClic(evt:MouseEvent) {
      evt.currentTarget.scaleX=1;
      evt.currentTarget.scaleY=1;
    }

  }
}
```

Comme vous pouvez le constater, nous redéfinissons un gestionnaire d'événement pour l'occurrence `sommaire`. Ainsi, lorsque vous cliquerez sur cette occurrence, les deux gestionnaires exécuteront les lignes d'instructions qu'ils gèrent. Vous pouvez ainsi contrôler plusieurs interactivités à des niveaux différents.

Dans le fichier `BoutonReactif.as`, nous initialisons l'apparence des occurrences issues du symbole dont nous avons défini une classe de liaison, mais nous précisons surtout et également que le message `Oui` doit être affiché dans la fenêtre Sortie de l'IDE de Flash lorsqu'un clic est effectué.

Dans le fichier `main.as`, nous donnons une autre instruction qui concerne toujours cette même occurrence : au clic, sa taille doit diminuer avant de reprendre son échelle initiale au moment où l'utilisateur relâchera le bouton de la souris.

Afin que le code que nous avons présentés dans ce dernier exemple puisse vous servir, voici quelques explications.

Pourquoi aucun nom d'occurrence n'est spécifié devant les membres d'une classe ?

Vous l'aurez peut-être remarqué, les lignes d'instructions ci-dessous, extraites des scripts que nous venons de voir, ne contiennent pas de nom d'instance en début de ligne.

```
addEventListener(MouseEvent.MOUSE_DOWN,rendreMobile);
startDrag();
filters=serieFiltres;
```

Commençons par rappeler que le langage ActionScript utilise une syntaxe pointée. De ce fait, une méthode ou une propriété doit toujours être précédée du nom d'une occurrence. Et pourtant, nous découvrons dans ces lignes que ce nom est absent.

Dans l'exemple ci-dessous, l'occurrence `curseur` est rendue mobile :

```
curseur.startDrag();
```

Dans celui-ci, le symbole dont la classe de liaison sera `ModeleCarte` génèrera des occurrences mobiles.

```
startDrag();
```

Précision

La ligne d'instruction ci-dessus qui n'est composée que d'une seule méthode est la 17e ligne du fichier `ModeleCarte.as`.

Si vous aviez oublié cette spécificité, rappelons également que le principe d'une classe est de contenir un code qui va s'appliquer à toutes les instances qui en sont issues. Nous devons, de ce fait, rester neutre et ne faire aucune référence.

Le schéma de la figure 16-3 illustre les relations qui existent entre un symbole, ses occurrences et les lignes d'instructions contenues dans la classe à laquelle elle est rattachée.

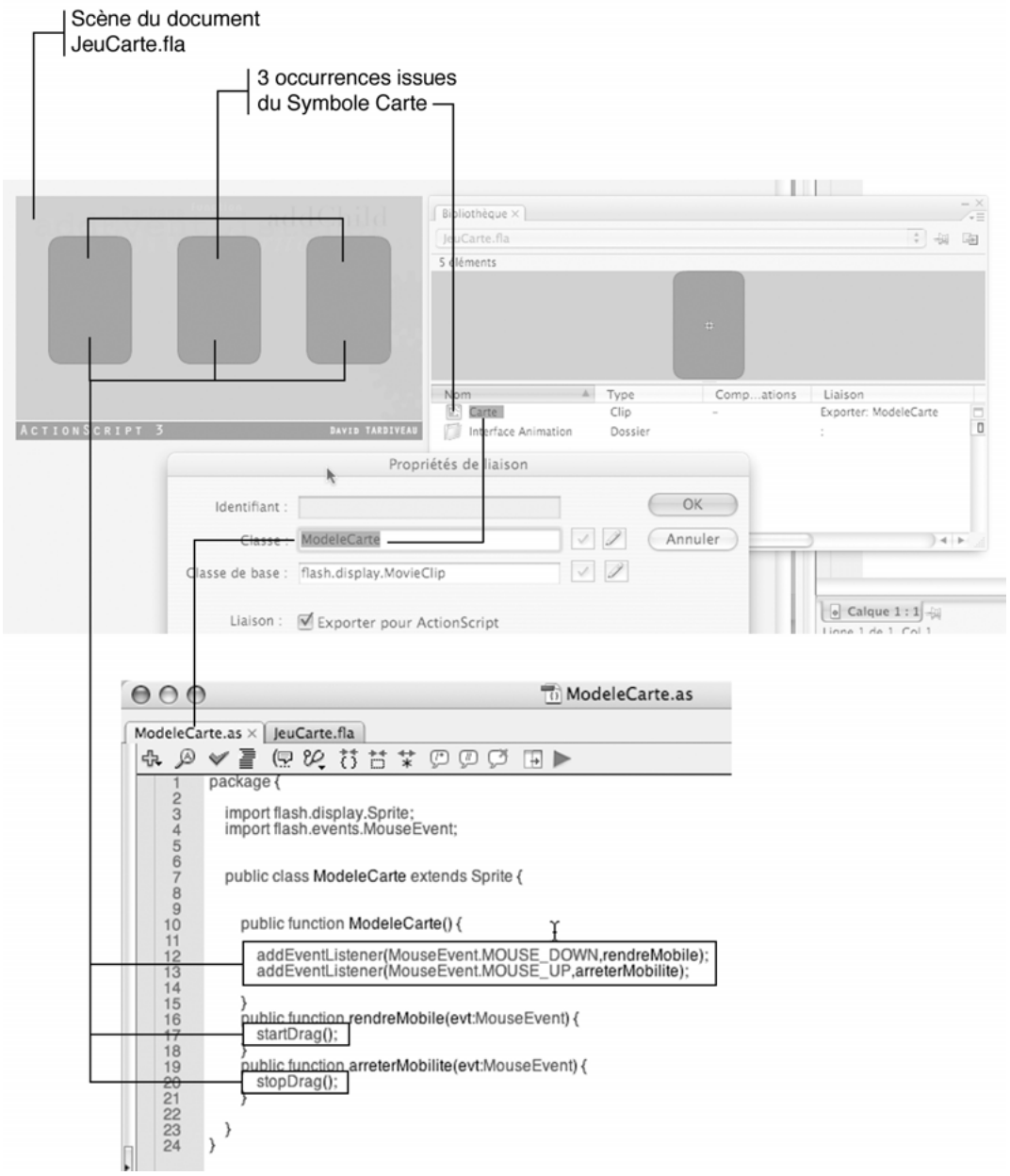


Figure 16-3

Les propriétés, méthodes et événements d'une classe s'appliquent aux occurrences d'une classe.

Explications sur le script du fichier BoutonReactif.as

La première série de lignes d'instructions ci-dessous a peu d'effet : elle permet simplement de créer un filtre sans pour autant l'affecter à une occurrence. Nous aurions pu nous contenter de ne saisir que les deux première lignes, car les trois dernières ne sont pas indispensables ; elles permettent uniquement de configurer le style du filtre.

```
var serieFiltres:Array = new Array();
var biseau:BevelFilter = new BevelFilter();
biseau.distance = 3;
biseau.blurX=7;
biseau.blurY=7;
```

Attention

Pour que le code ci-dessus fonctionne correctement, nous devons impérativement importer la classe `BevelFilter` en début de package : `import flash.filters.BevelFilter`.

Une occurrence ne possède pas obligatoirement un seul filtre. De ce fait, nous devons créer un tableau qui va permettre de stocker d'éventuels filtres supplémentaires. Pour terminer, comme vous le montre la dernière ligne ci-dessous, nous appliquons la propriété `filters` à chaque occurrence en précisant la valeur `serieFiltres`.

```
serieFiltres.push(biseau);
filters=serieFiltres;
```

Si vous ouvrez le fichier que nous fournissons en exemple, vous noterez que les occurrences qui se trouvent sur la scène sont nommées `sommaire` et `quitter`. Nous pouvons ainsi les utiliser comme valeur pour la propriété `text` du texte dynamique que nous avons placé dans le symbole `BoutonReactif`. Afin que le titre de l'occurrence apparaisse en majuscule, nous utilisons la méthode `toUpperCase()` qui a pour effet de basculer chaque caractère de la chaîne en majuscules.

```
nomEtiquette.text=name.toUpperCase();
```

Pour finir, nous définissons un gestionnaire d'événement pour les occurrences qui seront issues de cette classe.

```
addEventListener(MouseEvent.MOUSE_DOWN,clicSurLeBouton);
```

Le paramètre `clicSurLeBouton` est le nom de la fonction appelée au moment du clic.

Explications sur le script du fichier main.as

```
public class main extends Sprite {

    public function main() {
      sommaire.addEventListener(MouseEvent.MOUSE_DOWN,actionClic);
      sommaire.addEventListener(MouseEvent.MOUSE_UP,actionRelacherClic);
    }
```

```
    public function actionClic(evt:MouseEvent) {
      evt.currentTarget.scaleX=0.8;
      evt.currentTarget.scaleY=0.8;
    }
    public function actionRelacherClic(evt:MouseEvent) {
      evt.currentTarget.scaleX=1;
      evt.currentTarget.scaleY=1;
    }

  }
```

Remarque

Pour les non-anglophones, le nom de la classe est `main` pour évoquer la classe principale (qui se traduit par *main* en anglais).

Ce script est des plus simples car il contient simplement deux gestionnaires d'événements accompagnés des fonctions qu'ils appellent. Dans la mesure où ce fichier ActionScript est celui d'un document FLA, nous devons spécifier, cette fois-ci, les occurrences auxquelles nous souhaitons rattacher les gestionnaires.

Rappel

Le paramètre `evt.currentTarget` permet de faire référence à l'occurrence à laquelle la fonction est rattachée par le biais de la fonction `addEventListener()`.

Annexe

Les packages et classes

Comme vous avez pu le remarquer, avant de pouvoir faire référence à l'une des méthodes ou propriétés d'une classe (même à un événement), vous devez importer cette dernière en précisant son chemin, c'est-à-dire le dossier dans lequel elle se trouve. Il existe principalement deux classifications de packages : le `flash` et le `fl` ; tous deux contiennent de nombreuses classes et d'autres packages. Lorsque vous codez dans la fenêtre Actions de l'IDE de Flash, l'une des difficultés est de savoir quelles classes doivent être importées pour pouvoir faire appel à telle ou telle méthode ou propriété.

Remarque

L'un des avantages de la solution de développement FlashDevelop (`http://www.flashdevelop.org/community/`), est d'intégrer automatiquement les lignes d'instructions en début de programme qui contiennent les directives `import`.

Pour vous aider à surmonter cette difficulté, retenez que les classes du package `fl` servent à gérer principalement les composants, les animations à base de code (`Motion`), les interpolations à base de code (`Tween`) et les transitions à base de code (`Transitions`). Il y a de fortes chances que vos premières animations ne fassent donc pas appel à de telles fonctionnalités. De ce fait, vous aurez davantage besoin d'importer les classes du package `flash`.

Ces dernières servent à gérer toutes les autres actions telles que la gestion de l'affichage à l'écran, les événements, le texte, les filtres, les bitmap, les médias, les transferts de données, les informations système, et quelques autres.

Remarque

Il existe les classes de niveau supérieur qui ne se trouvent pas dans les packages que nous venons d'évoquer. Elles servent à gérer la date (`Date`), les nombres (`Number`), les tableaux (`Array`), les chaînes de caractères (`String`) et les calculs mathématiques (`Math`). Vous n'avez donc pas besoin de faire appel à la directive `import` pour spécifier le chargement d'une de ces classes.

Les packages souvent utilisés

Dans la plupart de vos programmes, vous aurez sûrement besoin de faire appel aux mêmes classes, ces dernières se trouvant sûrement dans l'un des packages du tableau A-1.

Tableau A-1 Les principaux packages utilisés dans un programme

Nom du package	Contenu ou rôle du package
`flash.events`	À utiliser dés que vous gérez des événements dans une animation.
`flash.display`	À utiliser dès que vous gérez un affichage sur la scène (`Sprite`, `MovieClip`, `Loader`, `Stage`, `Shape`, `Bitmap`, `BitmapData`, etc.).
`flash.text`	À utiliser dès que vous manipulez du texte sur la scène (`TextField`, `TextFormat`, `StyleSheet`, `FontTextFieldAutoSize`, etc.).
`flash.net`	Contient des classes permettant d'envoyer et recevoir des données à partir du réseau, telles que le téléchargement d'URL et Flash Remoting.
`flash.ui`	Contient des classes d'interface utilisateur, telles que les classes permettant d'interagir avec la souris et le clavier.
`fl.controls`	Classes de composant de niveau supérieur à utiliser avec `List`, `Button` et `ProgressBar`.
`fl.events`	À utiliser pour gérer les événements propres aux composants.
`fl.video`	Classes à utiliser avec les composants `FLVPlayback` et `FLVPlaybackCaptioning`.

Si le fonctionnement et l'utilisation de la directive `import` vous semblent encore abstraits, prenons l'exemple d'un script dans lequel vous programmez l'interactivité d'un bouton. Vous aurez besoin d'écrire la ligne d'instruction ci-dessous afin que le compilateur de Flash sache qu'il doit d'abord apprendre à gérer les événements (`events`) relatifs à la souris (`MouseEvent`).

```
import flash.events.MouseEvent;
```

La directive `import` provoque la lecture des lignes d'instructions contenues dans un fichier de votre ordinateur afin que le compilateur de Flash sache ce qu'il doit écrire dans le SWF final. Si vous ne saisissez pas préalablement cette ligne d'instruction, vous ne pourrez pas exécuter proprement le code suivant (cela générera une erreur) :

```
nomOccurrMonBouton.addEventListener(MouseEvent.CLICK.nomFonctionAppelee);
```

Dans le tableau A-1, les trois premiers packages auxquels nous avons fait référence sont `flash.events`, `flash.display` et `flash.text`, voici respectivement leurs classes :

> **Remarque**
>
> Certaines définitions contenues des les tableaux ci-dessous sont extraites de la documentation officielle de Flash CS3.

Tableau A-2 Les classes du package flash.events

Nom de la classe	Instant du déclenchement de l'événement
`MouseEvent`	Importez cette classe dès que vous avez besoin d'utiliser l'une des constantes suivantes : `MouseEvent.CLICK`, `MouseEvent.DOUBLE_CLICK`, `MouseEvent.MOUSE_DOWN`, `MouseEvent.MOUSE_MOVE`, `MouseEvent.MOUSE_OUT`, `MouseEvent.MOUSE_OVER`, `MouseEvent.MOUSE_UP`, `MouseEvent.MOUSE_WHEEL`, `MouseEvent.ROLL_OUT` et `MouseEvent.ROLL_OVER`. Pour plus de détails sur la gestion des événements, consultez le chapitre 3 de ce livre, qui est dédié à cette spécificité technique.
`NetStatusEvent`	Dès que vous établirez des connexions entre deux machines avec les classes `NetConnection`, `NetStream` ou `SharedObject`, vous aurez besoin de faire appel à `NetStatusEvent` pour connaître l'état d'une connexion.
`TextEvent`	Cette classe devra être importée dès que vous aurez besoin de gérer un texte sur la scène, notamment avec l'une des constants suivantes : `TextEvent.LINK` et `TextEvent.TEXT_INPUT`. Consultez également la classe `FocusEvent` pour gérer l'état d'édition d'un texte de saisie.
`FocusEvent`	Dès que l'utilisateur déplace le focus sur un objet dans la liste d'affichage, utilisez l'une des constantes suivantes pour gérer ce changement de sélection : `FocusEvent.FOCUS_IN`, `FocusEvent.FOCUS_OUT`, `FocusEvent.KEY_FOCUS_CHANGE` et `FocusEvent.MOUSE_FOCUS_CHANGE`.
`TimerEvent`	Importez cette classe pour pouvoir gérer les deux événements liés aux deux étapes qui accompagnent la classe `Timer()` : le passage d'un intervalle temps (`TimerEvent.TIMER`) et la fin d'un timer (`TimerEvent.TIMER_COMPLETE`)
`DataEvent`	Flash Player distribue des objets `DataEvent` lorsqu'il a terminé de charger des données brutes.
`ActivityEvent`	Flash Player distribue un objet `ActivityEvent` chaque fois qu'une caméra ou un microphone signale qu'il est devenu actif ou inactif.
`AsyncErrorEvent`	Flash Player distribue un événement `AsyncErrorEvent` lorsqu'une exception est renvoyée par le code asynchrone natif (`LocalConnection`, `NetConnection`, `SharedObject` ou `NetStream`).
`ContextMenuEvent`	Flash Player distribue des objets `ContextMenuEvent` lorsqu'un utilisateur génère ou utilise le menu contextuel.
`ErrorEvent`	Flash Player distribue des objets `ErrorEvent` lorsqu'une erreur entraîne l'échec d'une opération réseau.
`Event`	La classe Event est utilisée comme classe de base pour la création des objets événements, transmis aux écouteurs d'événements en tant que paramètres lorsqu'un événement se produit.
`EventDispatcher`	La classe `EventDispatcher` implémente l'interface `IEventDispatcher` et est la classe de base de la classe `DisplayObject`.
`EventPhase`	La classe `EventPhase` fournit des valeurs à la propriété `eventPhase` de la classe `Event`.
`FullScreenEvent`	Flash Player distribue un objet `FullScreenEvent` chaque fois que la scène passe en mode d'affichage plein écran ou quitte ce mode.
`HTTPStatusEvent`	Flash Player distribue des objets `HTTPStatusEvent` lorsqu'une requête réseau renvoie un code d'état HTTP.
`IMEEvent`	Flash Player distribue des objets `IMEEvent` lorsqu'un utilisateur saisit du texte à l'aide d'un IME (éditeur de méthode d'entrée).
`IOErrorEvent`	Flash Player distribue un objet `IOErrorEvent` lorsqu'une erreur entraîne l'échec d'une opération d'envoi ou de chargement.

Tableau A-2 Les classes du package flash.events *(Suite)*

Nom de la classe	Instant du déclenchement de l'événement
`KeyboardEvent`	Flash Player distribue des objets `KeyboardEvent` en réponse à une saisie utilisateur au clavier.
`ProgressEvent`	Flash Player distribue les objets `ProgressEvent` lorsqu'une opération de chargement a commencé ou qu'un socket a reçu des données.
`SecurityErrorEvent`	Flash Player distribue des objets `SecurityErrorEvent` pour signaler une erreur de sécurité.
`StatusEvent`	Flash Player distribue des objets `StatusEvent` lorsqu'un périphérique, tel qu'une caméra ou un microphone, ou un objet comme `LocalConnection` publie son état.
`SyncEvent`	Flash Player distribue des événements `SyncEvent` lorsqu'une occurrence de `SharedObject` distante a été mise à jour par le serveur.

Tableau A-3 Les classes du package flash.display

Nom de la classe	Description de la classe
`ActionScriptVersion`	La classe `ActionScriptVersion` est une énumération de valeurs constantes qui indiquent la version du langage du fichier SWF chargé.
`AVM1Movie`	`AVM1Movie` est une classe simple représentant les clips AVM1, qui utilisent ActionScript 1.0 ou 2.0.
`Bitmap`	La classe `Bitmap` représente les objets d'affichage qui représentent les images bitmap.
`BitmapData`	La classe `BitmapData` vous permet d'utiliser les données (pixels) d'un objet Bitmap.
`BitmapDataChannel`	La classe `BitmapDataChannel` est une énumération de valeurs constantes qui désignent le canal à utiliser : rouge, bleu, vert ou transparence alpha.
`BlendMode`	Classe qui fournit des valeurs constantes relatives aux effets de mode de fondu visuels.
`CapsStyle`	La classe `CapsStyle` est une énumération de valeurs constantes qui spécifient le style d'extrémité à utiliser pour tracer les lignes.
`DisplayObject`	La classe `DisplayObject` constitue la classe de base de tous les objets susceptibles d'être insérés dans la liste d'affichage.
`DisplayObjectContainer`	La classe `DisplayObjectContainer` est la classe de base de tous les objets susceptibles de servir de conteneurs d'objets d'affichage dans la liste d'affichage.
`FrameLabel`	L'objet `FrameLabel` contient des propriétés qui spécifient un numéro d'image et le nom d'étiquette correspondant.
`GradientType`	La classe `GradientType` fournit les valeurs du paramètre type dans les méthodes `beginGradientFill()` et `lineGradientFill()` de la classe `flash.display.Graphics`.
`Graphics`	La classe `Graphics` contient un ensemble de méthodes permettant de créer une forme vectorielle.
`InteractiveObject`	La classe `InteractiveObject` correspond à la classe abstraite de l'ensemble des objets d'affichage avec lesquels l'utilisateur peut interagir à l'aide de la souris et du clavier.
`InterpolationMethod`	La classe `InterpolationMethod` transmet des valeurs au paramètre `interpolationMethod` par l'intermédiaire des méthodes `Graphics.beginGradientFill()` et `Graphics.lineGradientStyle()`.

Tableau A-3 Les classes du package flash.display *(Suite)*

Nom de la classe	Description de la classe
`JointStyle`	La classe `JointStyle` est une énumération de valeurs constantes qui spécifient le style de liaison à utiliser pour tracer les lignes.
`LineScaleMode`	La classe `LineScaleMode` fournit des valeurs pour le paramètre `scaleMode` de la méthode `Graphics.lineStyle()`.
`Loader`	La classe `Loader` permet de charger des fichiers SWF ou des fichiers d'image (JPG, PNG ou GIF).
`LoaderInfo`	La classe `LoaderInfo` fournit des informations relatives à un fichier SWF ou à un fichier d'image (JPEG, GIF ou PNG) chargé.
`MorphShape`	La classe `MorphShape` représente les objets `MorphShape` figurant dans la liste d'affichage.
`MovieClip`	La classe `MovieClip` hérite des classes suivantes : `Sprite`, `DisplayObjectContainer`, `InteractiveObject`, `DisplayObject` et `EventDispatcher`.
`PixelSnapping`	La classe `PixelSnapping` est une énumération de valeurs constantes destinées à la définition des options d'accrochage aux pixels par le biais de la propriété `pixelSnapping` d'un objet `Bitmap`.
`Scene`	La classe `Scene` contient des propriétés destinées à identifier le nom, les étiquettes et le nombre d'images d'une séquence.
`Shape`	La classe `Shape` sert à créer des formes légères par le biais de l'interface de programmation d'applications (API) de dessin ActionScript.
`SimpleButton`	La classe `SimpleButton` vous permet de contrôler toutes les occurrences de symboles de bouton dans un fichier SWF.
`SpreadMethod`	La classe `SpreadMethod` fournit les valeurs du paramètre `spreadMethod` dans les méthodes `beginGradientFill()` et `lineGradientStyle()` de la classe `Graphics`.
`Sprite`	La classe `Sprite` est un bloc constitutif de base de la liste d'affichage : un nœud de liste d'affichage qui permet d'afficher des images et peut également contenir des enfants.
`Stage`	La classe `Stage` représente la zone de dessin principale.
`StageAlign`	La classe `StageAlign` fournit les valeurs constantes à utiliser pour la propriété `Stage.align`.
`StageDisplayState`	La classe `StageDisplayState` fournit des valeurs pour la propriété `Stage.displayState`.
`StageQuality`	La classe `StageQuality` fournit les valeurs de la propriété `Stage.quality`.
`StageScaleMode`	La classe `StageScaleMode` fournit les valeurs de la propriété `Stage.scaleMode`.
`SWFVersion`	La classe `SWFVersion` est une énumération de valeurs constantes qui indiquent la version du format d'un fichier SWF chargé.

En fonction de la nature de vos projets, certains packages pourraient vous sembler indispensables alors qu'ils ne figurent pas dans le tableau A-1. En vous présentant subjectivement l'ensemble de ces tableaux, nous essayons simplement de mettre en évidence auprès des publics de développeurs Flash néophytes, les packages fréquemment utilisés qui nécessitent donc un appel de classes et/ou de packages.

Tableau A-4 Les classes du package flash.text

Nom de la classe	Description de la classe
`AntiAliasType`	La classe `AntiAliasType` fournit les valeurs d'anti-aliasing de la classe `flash.text.TextField`.
`CSMSettings`	La classe `CSMSettings` contient des propriétés à utiliser avec la méthode `TextRenderer.setAdvancedAntiAliasingTable()` pour mettre en place une modulation continue du trait (CSM).
`Font`	La classe `Font` permet de gérer des polices intégrées dans les fichiers SWF.
`FontStyle`	La classe `FontStyle` fournit les valeurs de la classe `TextRenderer`.
`FontType`	La classe `FontType` contient les constantes énumérées `embedded` et `device` pour la propriété `fontType` de la classe `Font`.
`GridFitType`	La classe `GridFitType` définit des valeurs pour l'adaptation à la grille dans la classe `TextField`.
`StaticText`	Cette classe représente les objets `MorphShape` figurant dans la liste d'affichage.
`StyleSheet`	La classe `StyleSheet` permet de créer un objet feuille de styles contenant des règles de formatage de texte pour la taille et la couleur de la police ainsi que d'autres styles de formatage.
`TextColorType`	La classe `TextColorType` fournit des valeurs de couleur pour la classe `flash.text.TextRenderer`.
`TextDisplayMode`	La classe `TextDisplayMode` regroupe les valeurs qui contrôlent l'anti-aliasing des sous-pixels du système d'anti-aliasing avancé.
`TextField`	La classe `TextField` permet de créer des objets d'affichage et de saisie de texte.
`TextFieldAutoSize`	La classe `TextFieldAutoSize` énumère les valeurs constantes utilisées lors de la définition de la propriété `autoSize` de la classe `TextField`.
`TextFieldType`	La classe `TextFieldType` énumère les valeurs constantes utilisées lors de la définition de la propriété type de la classe `TextField`.
`TextFormat`	La classe `TextFormat` représente les informations de mise en forme de caractères.
`TextFormatAlign`	La classe `TextFormatAlign` fournit des valeurs pour l'alignement du texte de la classe `TextFormat`.
`TextLineMetrics`	La classe `TextLineMetrics` contient des informations sur la position du texte et les unités de mesure d'une ligne de texte dans un champ texte.
`TextRenderer`	La classe `TextRenderer` permet d'exploiter la fonction avancée d'anti-aliasing des polices incorporées.
`TextSnapshot`	Les objets `TextSnapshot` permettent de travailler avec du texte statique dans un clip.

Les autres packages

Dans un souci d'exhaustivité, nous vous présentons ci-dessous, tous les packages qui ne figuraient pas dans le tableau A-1. Consultez l'aide en ligne de Flash pour découvrir les différentes classes contenues dans ces packages.

Tableau A-5 Les classes des packages utilisés ponctuellement

Nom du package	Contenu ou rôle du package
`adobe.utils`	Fonction (MMExecute) et classes utilisées (CustomActions et XMLUI) par les développeurs d'outils de programmation de Flash.
`fl.accessibility`	Classes qui permettent de prendre en charge l'accessibilité des composants de l'IDE de Flash.
`fl.containers`	Classes qui chargent du contenu ou d'autres composants.
`fl.controls.dataGridClasses`	Classes utilisées par le composant `DataGrid` pour y organiser les informations contenues dans un tableau (une grille).
`fl.controls.listClasses`	Classes utilisées par les composants de type `List` pour y organiser les informations contenues dans une occurrence.
`fl.controls.progressBarClasses`	Classes dédiées au composant `ProgressBar`.
`fl.core`	Classes liées à tous les composants.
`fl.data`	Classes servant à gérer les données associées à un composant.
`fl.events`	Classes d'événement dédiées aux composants.
`fl.lang`	L'unique classe Locale de ce package prend en charge le texte multilingue.
`fl.livepreview`	Classes dédiées au comportement d'aperçu en direct d'un composant dans l'IDE (l'environnement de programmation) de Flash.
`fl.managers`	Classes servant à gérer les relations entre un composant et un utilisateur.
`fl.motion`	À utiliser pour gérer les interpolations de mouvement.
`fl.motion.easing`	Classes à utiliser pour gérer les variations de vitesses (accélération, décélération) dans un mouvement obtenu grâce à la classe `Motion`.
`fl.transitions.easing`	Classes à utiliser avec les classes `fl.transitions` pour créer des effets d'accélération.
`fl.transitions`	Classes qui vous permettent d'utiliser l'ActionScript pour créer des effets d'animation.
`flash.accessibility`	Classes qui permettent de prendre en charge l'accessibilité du contenu et des applications Flash.
`flash.errors`	Classes d'erreur couramment utilisées.
`flash.external`	L'unique classe `ExternalInterface` de ce package permet de communiquer avec le conteneur de Flash Player.
`flash.filters`	Classes pour les effets de filtrage de bitmaps.
`flash.media`	Classes permettant de manipuler des ressources multimédia, telles que des sons et des vidéos.
`flash.printing`	Classes permettant d'imprimer le contenu Flash.
`flash.system`	Classes permettant d'accéder aux fonctionnalités de niveau système, telles que la sécurité, le contenu multilingue, etc.
`flash.utils`	Classes d'utilitaires, telles que des structures de données comme `ByteArray`.
`flash.xml`	Classes servant à prendre en charge de l'ancien code XML.
`flash.geom`	Classes géométriques, telles que les points, les rectangles et les matrices de transformation, pour prendre en charge la classe `BitmapData` et la fonctionnalité d'interception de bitmaps.

Intégrer des tables de caractères dans une animation

Vous aurez parfois besoin d'intégrer des polices de caractères à certains textes dynamiques ou de saisie, mais vous ne savez pas quels sont les glyphes contenus dans les tables ci-dessous ; faisons toute la lumière sur ces dernières.

> **Rappel**
>
> L'intégration d'une table de caractères dans une animation Flash est nécessaire lorsque vous souhaitez que les textes dynamiques et de saisie s'affichent correctement (suivant la police que vous avez choisie au moment de l'intégration de vos zones de textes sur la scène). Par ailleurs, gardez à l'esprit qu'une telle intégration augmente le poids de vos publications.

Intégrer des caractères dans une animation

Pour information et rappel, voici la technique qui permet d'intégrer une table de caractères à un texte dynamique ou de saisie.

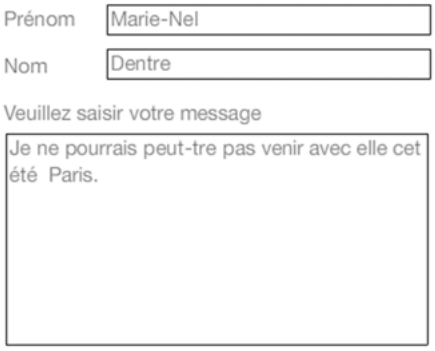
1. Sélectionnez le texte dynamique ou de saisie à traiter sur la scène.
2. Cliquez sur le bouton Intégrer de la palette Propriétés.
3. Cliquez sur la ou les tables de caractères à intégrer de la liste figurant sur la copie d'écran de la figure A-2 (maintenez la touche Commande (Mac) ou Ctrl (Windows) pour sélectionner plusieurs tables).

Généralement, il faut intégrer les tables Majuscules, Minuscules, Chiffres, Ponctuation, Latin basique et Latin I pour les caractères accentués, pour être sûr d'afficher tous les glyphes (signes) d'un texte dynamique ou de saisie. Même si certains caractères accentués ne nécessitent pas la table Latin I pour pouvoir être visibles, la plupart d'entre eux se trouvent tout de même dans cette table. Pour des textes de saisie où l'utilisateur devra simplement renseigner un code postal ou un numéro de téléphone, il n'est pas nécessaire d'intégrer les majuscules et minuscules.

> **Remarque**
>
> Si vous omettez d'intégrer une table de caractères dans un texte dynamique ou de saisie, les caractères faisant partie de cette description (la table de caractères) ne seront tout simplement pas visibles comme le démontre la copie d'écran de la figure A-1.

Figure A-1

L'utilisateur à saisi les informations suivantes : Marie-Nöel Denêtre avec le message Je ne pourrais peut-être pas venir cet été à Paris. Certains caractères ne sont pas visibles car seules les majuscules et les minuscules ont été intégrées.



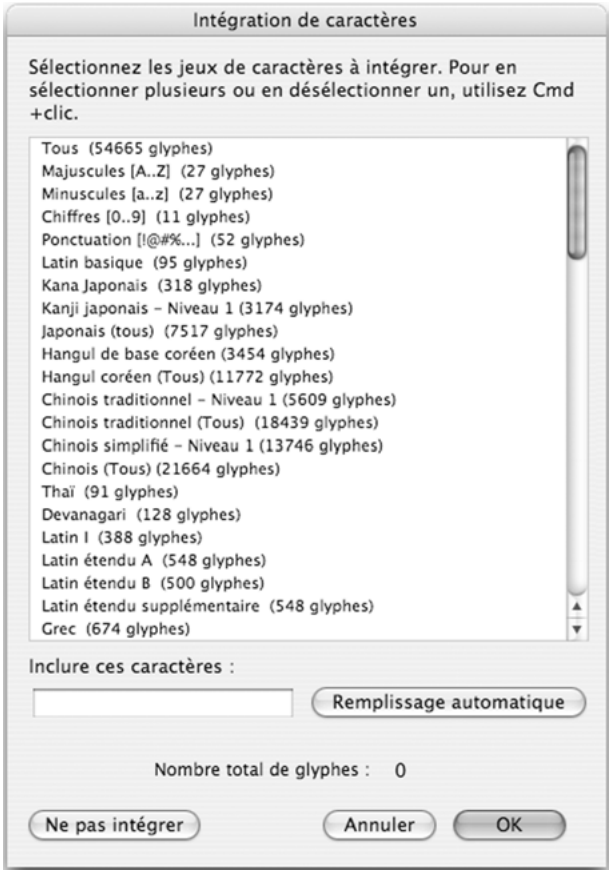


Figure A-2

Les tables de caractères intégrées servent à afficher correctement les textes dynamique et de saisie sur l'écran d'un internaute qui ne possède pas la police que vous utilisez dans votre animation.

Tables des caractères Unicode

Comme l'exprime très bien le titre de ce développement, les tables des 128 signes auxquelles Flash fait référence lorsque vous vous apprêtez à intégrer un ensemble de caractères font appel à une nomenclature précise. La codification d'un signe se fait sur la base suivante :

Latin basique

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000																
001																
002		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
003	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
004	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
005	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
006	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
007	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Latin I

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
008																
009																
00A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
00B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
00C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É		Ë	Ì	Í	Î	Ï
00D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
00E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
00F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Latin étendu A

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	Ā	ā	Ă	ă	Ą	ą	Ć	ć	Ĉ	ĉ	Ċ	ċ	Č	č	Ď	ď
11	Đ	đ	Ē	ē	Ĕ	ĕ	Ė	ė	Ę	ę	Ě	ě	Ĝ	ĝ	Ğ	ğ
12	Ġ	ġ	Ģ	ģ	Ĥ	ĥ	Ħ	ħ	Ĩ	ĩ	Ī	ī	Ĭ	ĭ	Į	į
13	İ	ı	Ĳ	ĳ	Ĵ	ĵ	Ķ	ķ	ĸ	Ĺ	ĺ	Ļ	ļ	Ľ	ľ	Ŀ
14	ŀ	Ł	ł	Ń	ń	Ņ	ņ	Ň	ň	ŉ	Ŋ	ŋ	Ō	ō	Ŏ	ŏ
15	Ő	ő	Œ	œ	Ŕ	ŕ	Ŗ	ŗ	Ř	ř	Ś	ś	Ŝ	ŝ	Ş	ş
16	Š	š	Ţ	ţ	Ť	ť	Ŧ	ŧ	Ũ	ũ	Ū	ū	Ŭ	ŭ	Ů	ů
17	Ű	ű	Ų	ų	Ŵ	ŵ	Ŷ	ŷ	Ÿ	Ź	ź	Ż	ż	Ž	ž	ſ

Latin étendu B

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
018	ƀ	Ɓ	Ƃ	ƃ	Ƅ	ƅ	Ɔ	Ƈ	ƈ	Ɖ	Ɗ	Ƌ	ƌ	ƍ	Ǝ	Ə
019	Ɛ	Ƒ	ƒ	Ɠ	Ɣ	ƕ	Ɩ	Ɨ	Ƙ	ƙ	ƚ	ƛ	Ɯ	Ɲ	ƞ	Ɵ
01A	Ơ	ơ	Ƣ	ƣ	Ƥ	ƥ	Ʀ	Ƨ	ƨ	Σ	ƪ	ƫ	Ƭ	ƭ	Ʈ	Ư
01B	ư	Ʊ	Ʋ	Ƴ	ƴ	Ƶ	ƶ	Ʒ	Ƹ	ƹ	ƺ	ƻ	Ƽ	ƽ	ƾ	ƿ
01C	ǀ	ǁ	ǂ	ǃ	DŽ	Dž	dž	LJ	Lj	lj	NJ	Nj	nj	Ǎ	ǎ	Ǐ
01D	ǐ	Ǒ	ǒ	Ǔ	ǔ	Ǖ	ǖ	Ǘ	ǘ	Ǚ	ǚ	Ǜ	ǜ	ǝ	Ǟ	ǟ
01E	Ǡ	ǡ	Ǣ	ǣ	Ǥ	ǥ	Ǧ	ǧ	Ǩ	ǩ	Ǫ	ǫ	Ǭ	ǭ	Ǯ	ǯ
01F	ǰ	DZ	Dz	dz	Ǵ	ǵ	Ƕ	Ƿ	Ǹ	ǹ	Ǻ	ǻ	Ǽ	ǽ	Ǿ	ǿ
020	Ȁ	ȁ	Ȃ	ȃ	Ȅ	ȅ	Ȇ	ȇ	Ȉ	ȉ	Ȋ	ȋ	Ȍ	ȍ	Ȏ	ȏ
021	Ȑ	ȑ	Ȓ	ȓ	Ȕ	ȕ	Ȗ	ȗ	Ș	ș	Ț	ț	Ȝ	ȝ	Ȟ	ȟ
022	Ƞ	ȡ	Ȣ	ȣ	Ȥ	ȥ	Ȧ	ȧ	Ȩ	ȩ	Ȫ	ȫ	Ȭ	ȭ	Ȯ	ȯ
023	Ȱ	ȱ	Ȳ	ȳ	ȴ	ȵ	ȶ									

Index

ActionScript 3

Programmation séquentielle et orientée objet

David Tardiveau

Enseignant à GOBELINS, l'école de l'image, et développeur Flash depuis 1999, David Tardiveau assure des cours et des productions dans le domaine du multimédia off et on-line. Il est aussi le fondateur et l'animateur du site www.yazo.net, site bien connu de la communauté Flash qui propose des tutoriaux sur l'ActionScript et des explications sur la partie animation du logiciel.

Une version qui marque un tournant

L'ActionScript 3, basé sur l'ECMAScript (révision 3 de l'ECMA-262), est un langage totalement orienté objet qui autorise cependant une syntaxe de programmation séquentielle. Très proche du JavaScript, il ressemble également fortement au langage Java. Caractérisé par une syntaxe différente de l'ActionScript 1 et 2, avec de nouvelles classes, propriétés, méthodes et événements, il est plus complexe dans son architecture, mais plus efficace et rapide à maîtriser.

Une approche complète et précise de l'ActionScript 3

Cet ouvrage très pédagogique détaille les nouveaux concepts de l'ActionScript 3, de la displayList aux écouteurs, en passant par la nouvelle syntaxe utilisée en XML. Toutes les notions de base y sont expliquées, mais également les propriétés et manipulations élémentaires d'occurrences, le contrôle des différents médias (texte, image, son, vidéo), ou encore la gestion du XML. Il s'adresse aussi bien aux développeurs utilisant une programmation séquentielle qu'aux habitués de la programmation orientée objet : de nombreux exemples y sont en effet proposés dans les deux modes de programmation. Sur l'extension Web du livre, le lecteur pourra télécharger 180 animations Flash correspondant aux exemples des différents chapitres.

À qui s'adresse cet ouvrage ?

– Aux développeurs web
– Aux graphistes Flash qui maîtrisent l'interface du logiciel

Au sommaire

Sur le site www.yazo.net/eyrolles
– Téléchargez les 180 scripts de l'ouvrage
– Dialoguez avec l'auteur

www.editions-eyrolles.com
Groupe Eyrolles | Diffusion Geodif | Distribution Sodis

Code éditeur : G12282
ISBN : 978-2-212-12282-4

9 782212 122824

Conception : Nord Compo

EYROLLES