5 Les galeries d'images

Introduction

Pour gérer une galerie d'images, vous pouvez naturellement la travailler dans le scénario et jouer l'animation, mais il est préférable d'externaliser les contenus. D'abord, vous optimisez confortablement le poids de l'application. Ensuite, vous en simplifiez la maintenance. Dans ce contexte de galerie d'images externalisées, il suffit alors de remplacer les visuels importés par d'autres du même nom, pour automatiquement mettre à jour les images de l'animation, sans avoir à rééditer le Flash.

Pour construire une galerie d'images, nous devons d'abord apprendre à charger ces images progressivement, puis, à organiser leur affichage en les appelant directement un répertoire externe, et en y associant des informations textuelles, telles que des titres ou des légendes. Pour améliorer la gestion des contenus, nous verrons qu'il peut être intéressant de centraliser ces informations dans un fichier XML de sorte à simplifier la maintenance du projet une fois celui-ci publié.

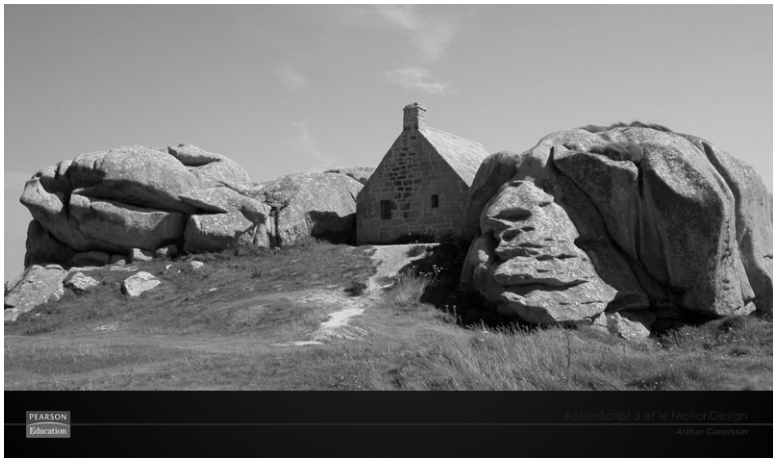
Nous aborderons aussi la manière de rendre ces développements plus pertinents, d'autoriser l'interaction sur chaque objet généré dynamiquement et d'associer à chaque image, un lien, une fonctionnalité de zoom ou toute autre action.

La détection de bogues éventuels pouvant survenir suite à une erreur de chargement ou d'exécution du programme, nous traiterons la manière de placer des contrôles afin d'éradiquer les risques de plantage du programme. Enfin, nous évoquerons l'intégration dynamique des données que vous pourriez extraire d'une base de type MySQL ou de tout autre système d'informations, grâce à vos connaissances acquises sur la gestion d'un flux XML.

Afficher des images externalisées et aléatoires

Dans cette première section, nous abordons le chargement d'images externalisées. Cela nous permet d'alléger grandement le poids des documents SWF que nous créons. Nous abordons aussi la gestion de l'affichage d'une image avec le paramètre `random` de sorte que l'animation publiée affiche un visuel ou un autre et ce, de manière entièrement aléatoire (voir Figure 5.1).

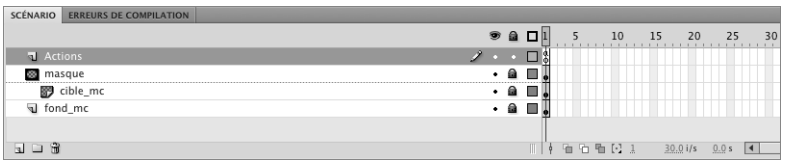
Figure 5.1
Un des aperçus du document à la publication.



Exemples > ch5_galeriesImages_1.fla

Dans la scène du document "ch5_galerieImages_1.fla", au-dessus du calque `fond_mc`, apparaît un calque masqué nommé `cible_mc` (voir Figure 5.2). Ce calque affiche un symbole de type MovieClip du même nom. Ce symbole est vide et sert de conteneur pour importer les images appelées dynamiquement par ActionScript.

Figure 5.2
Aperçu du scénario de la scène principale.



Dans le calque nommé `actions`, nous pouvons lire le code suivant :

```
// CHARGEMENT
var chargeurFond:Loader = new Loader();
var valeurAleatoire:Number=Math.round(Math.random()*2);
var urlFond:URLRequest=new URLRequest("images/coteBretonne/photo"+valeur
➥ Aleatoire+".jpg");
chargeurFond.load(urlFond);

// COMPLETE
chargeurFond.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
  cible_mc.addChild(chargeurFond);
}
```

D'abord, une série de variables initialise le chargement :

```
var chargeurFond:Loader = new Loader();
```

La variable `chargeurFond` réserve l'emplacement mémoire pour la définition d'un chargeur (`:Loader`). Le signe égal (=) affecte une valeur qui désigne, ici, l'instanciation d'un nouvel objet `Loader` (`= new Loader()`).

Plus loin, une variable de type nombre (`valeurAleatoire`) définit une valeur aléatoire comprise entre 0 et 2 inclus. Celle-ci est obtenue grâce à la classe `Math.random()`. La classe `Math.round()` utilisée également permet d'arrondir la valeur obtenue à un chiffre entier. Pour combiner les deux méthodes, nous imbriquons la première dans la seconde :

```
var valeurAleatoire:Number=Math.round(Math.random()*2);
```

C'est cette valeur que nous récupérons plus tard et qui déterminera l'image à charger.

Générer une valeur aléatoire

Pour générer une valeur aléatoire, nous utilisons la classe `Math.random()` multipliée par une valeur. Cette valeur correspond au seuil limite (supérieur) du nombre que l'on souhaite obtenir. Par exemple, `Math.random()*5` renvoie une valeur comprise entre 0 et 5.

Mais attention, la valeur obtenue, même multipliée, reste décimale, car en réalité, `Math.random()` génère une valeur comprise entre 0,00... et 0,9999... En la multipliant, nous agrandissons donc uniquement les valeurs décimales en d'autres valeurs décimales, mais à une échelle plus grande. Par exemple, si l'on considère :

```
Math.random()*5;
```

et que la méthode `Math.random()` renvoie 0.7. Alors, nous obtenons :

```
0,7 x 5 = 3,5
```

Arrondir une valeur

Pour obtenir un chiffre entier, nous utilisons l'une des méthodes suivantes :

- Pour arrondir à l'entier le plus proche, on utilise `Math.round()` : `Math.round(Math.random()*5)`.
- Pour arrondir à l'entier supérieur le plus proche, on utilise `Math.ceil()` : `Math.ceil(Math.random()*5)`.
- Pour arrondir à l'entier inférieur le plus proche, on utilise `Math.floor()` : `Math.floor(Math.random()*5)`.

Ainsi, la méthode `Math.round`, en dessous de `0.5`, arrondit à `0`, au dessus de `0.5`, arrondit à 1. La méthode `Math.ceil`, de `0.01` à `0.9`, arrondit à 1, et la méthode `Math.floor`, de `0.01` à `0.999`, arrondit à `0`... Testez par exemple ceci pour identifier clairement le résultat obtenu par chacune des trois méthodes :

```
trace("floor")
for (var i:Number=0;i<20;i++){
  trace(Math.floor(Math.random()*2));
}
trace("round")
for (var i:Number=0;i<20;i++){
  trace(Math.round(Math.random()*2));
}
trace("ceil")
for (var i:Number=0;i<20;i++){
  trace(Math.ceil(Math.random()*2));
}
```

Une fois la valeur aléatoire définie, une autre variable, `urlFond`, stocke le chemin d'accès à l'image. L'image est localisée dans un dossier nommé "coteBretonne" situé dans le répertoire "images" qui accompagne les documents Flash des exemples du livre :

```
var urlFond:URLRequest=new URLRequest("images/coteBretonne/
photo"+valeurAleatoire+".jpg");
```

Le nom de la photo appelée est construit de manière dynamique. Nous concaténons à la volée notre valeur aléatoire entre la racine du nom et son extension (racine + valeurAleatoire + extension). Selon la valeur obtenue par la variable, le chemin enregistré pour le chargement de l'image sera "images/coteBretonne/photo0.jpg", "images/coteBretonne/photo1.jpg" ou "images/coteBretonne/photo2.jpg".

Définir les chemins pour les requêtes externes. *Lorsque vous appelez un contenu situé à l'extérieur d'un document Flash (une image, un autre fichier SWF, un flux XML, un son, une vidéo, etc., sauf pour les classes .as qui sont compilées à la publication), il est important de déterminer l'emplacement du contenu appelé par rapport à l'emplacement de la page HTML qui exécute le fichier Flash, et non par rapport au fichier Flash lui-même. En effet, si le fichier Flash est enregistré dans un emplacement différent de la page qui le contient, le chemin se référant à ce document Flash risque d'être invalide. Considérez toujours que la page HTML qui exécute le Flash modifie de ce fait la position relative de l'animation Flash, en la définissant par rapport à la page HTML elle-même.*

Une fois le chemin défini pour le chargement de l'image, nous indiquons au chargeur (`chargeurFond`), à l'aide de la méthode `load`, de charger cette image (`urlFond`).

```
chargeurFond.load(urlFond);
```

Plus loin, un écouteur est attaché au chargeur `chargeurFond` et appelle une fonction :

```
// COMPLETE
chargeurFond.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
  cible_mc.addChild(chargeurFond);
}
```

L'écouteur est attaché à l'objet `LoaderInfo` du chargeur et non directement au chargeur lui-même. La propriété `contentLoaderInfo` appartient en effet à l'objet `Loader` et permet d'y lire des informations relatives à la progression du chargement. Pour ajouter des instructions d'affichage suite au chargement, nous devons donc introduire cet objet dans le gestionnaire d'événements, ce qui donne :

```
chargeurFond.contentLoaderInfo.addEventListener
    (Event.COMPLETE,chargementCOMPLET);
```

Lorsque le chargement est complet (`Event.COMPLETE`), c'est-à-dire lorsque l'image appelée avec la méthode `load` est entièrement chargée sur le poste client, l'écouteur exécute une fonction que nous avons ici nommée `chargementCOMPLET` :

```
function chargementCOMPLET(evt:Event) {
  cible_mc.addChild(chargeurFond);
}
```

Cette fonction affiche l'objet chargé directement dans le MovieClip nommé `cible_mc` qui sert de conteneur (ou l'ajoute comme enfant à la liste d'affichage), à l'aide de la méthode `addChild()` :

```
Cible_mc.addChild(chargeurFond);
```

En affichant l'élément chargé dans un symbole de type MovieClip, nous lui faisons bénéficier des propriétés afférentes aux MovieClip. L'objet importé peut donc, par ce conteneur, être animé et contrôlé par ActionScript (position, effets de couleur, affichage, filtres, animations, etc.).

Info

Doit-on toujours placer un écouteur sur un chargeur pour activer l'affichage d'un contenu chargé dynamiquement ? Il est possible, lorsque vous réalisez le document localement, de vous passer de l'écouteur attaché au chargeur et d'invoquer directement l'affichage (`addChild`) suite au chargement avec la méthode `load`. Le fichier placé localement est déjà présent sur votre poste et ne requiert pas, dans ce cas, d'être préalablement téléchargé avant d'être affiché. Il est affiché directement, car il est déjà disponible. Mais, pour tout contenu destiné à une publication en ligne, pour un site donc, il convient de passer par l'écouteur qui assurera l'affichage seulement une fois le contenu entièrement chargé. Le cas échéant, les contenus appelés dynamiquement seraient considérés par le lecteur Flash comme inexistants au moment de l'exécution des scripts attachés à ces objets, ce qui pourrait nuire à la bonne exécution de votre programme. Pour placer une action alternative en cas de problème lié au chargement, consultez aussi la section consacrée à la gestion des erreurs au chargement, en fin de chapitre.

À retenir

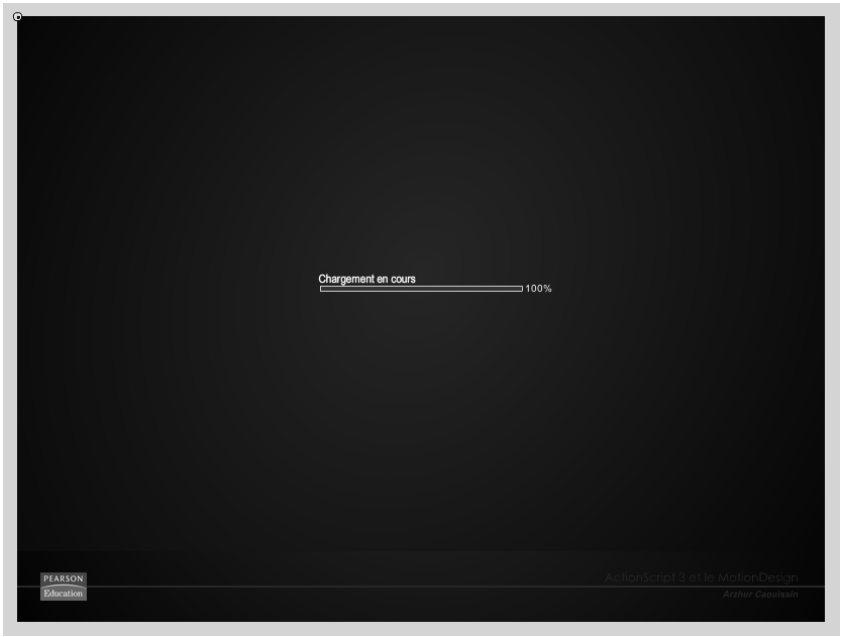
- Pour optimiser le poids d'un document qui affiche des images de grande taille, il est souhaitable d'externaliser ces images à l'aide d'un chargeur.
- Les paramètres du chargeur permettent de définir de manière dynamique les contenus à charger. Nous pouvons de ce fait avoir recours à une variable aléatoire pour construire le chemin de chargement.
- Pour appliquer une valeur aléatoire, nous utilisons la classe `Math.random` et nous arrondissons à un entier la valeur décimale obtenue avec les classes `Math.round`, `Math.ceil` ou `Math.floor`.
- Les chemins des éléments chargés dynamiquement sont relatifs aux pages qui contiennent l'animation Flash.
- Il convient d'utiliser un écouteur pour attendre la fin du chargement d'un contenu avant de l'afficher, à défaut de quoi, l'application développée pourrait ne pas fonctionner normalement.

Réaliser une jauge de chargement

Lorsque le contenu chargé dynamiquement est lourd (plus de 100 Ko), il est souhaitable de signaler la progression du chargement. Pour ce faire, nous utilisons une jauge signalétique afin d'avertir l'utilisateur que l'absence d'affichage ne désigne pas un bogue ou une erreur de sa part, mais qu'il faut attendre la fin du chargement pour en visualiser le contenu.

Dans cette section, nous ajoutons donc une jauge de progression au chargement mis en œuvre à la section précédente (voir Figure 5.3).

Figure 5.3
Aperçu de la jauge de chargement.



Exemples > ch5_galeriesImages_2.fla

Dans la scène du document "ch5_galeriesImages_2.fla", au-dessus du calque `fond_mc`, apparaît un calque masqué nommé `cible_mc`. Ce calque affiche le même symbole `cible_mc` vide, observé précédemment. Au-dessus du masque figure un autre calque, `chargement_mc`, qui affiche une jauge de progression (voir Figure 5.4). À l'intérieur de ce nouveau symbole, nous distinguons, clairement répartis vers les calques (voir Figure 5.5), un autre symbole, `barre_mc`, qui contient une forme graphique rectangulaire et dont le centre géométrique est positionné à gauche (pour que l'objet se déforme vers la droite, dans le sens de la lecture de la progression) (voir Figure 5.6), ainsi qu'un texte dynamique nommé `pourcentage_txt`. Les autres éléments sont purement figuratifs.

Figure 5.4
Aperçu du scénario de la scène principale.

Figure 5.5
Aperçu du scénario du symbole `chargement_mc`.

Figure 5.6
Aperçu du symbole `barre_mc` *dont le centre géométrique est situé à gauche.*

Dans le calque nommé `actions`, nous pouvons lire le code suivant :

```
// CHARGEMENT
var chargeurFond:Loader = new Loader();
var valeurAleatoire:Number=Math.round(Math.random()*2);
var urlFond:URLRequest=new URLRequest("images/coteBretonne/photo"+valeur
➥ Aleatoire+".jpg");
chargeurFond.load(urlFond);

// PROGRESSION
chargeurFond.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➥ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
   var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded / evt.current
   ➥ Target.bytesTotal);
   chargement_mc.barre_mc.scaleX=valeurPourcentage;
   chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+"%";
}

// COMPLETE
chargeurFond.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
   cible_mc.addChild(chargeurFond);
   chargement_mc.visible=false;
}
```

Comme à la section précédente, nous retrouvons d'abord les variables qui activent le chargement. Ensuite, deux écouteurs sont attachés à la même propriété `contentLoaderInfo`.

Le premier écouteur exécute une fonction durant la progression du chargement (`PROGRESS`), alors que le second en exécute une autre une fois ce chargement terminé (`COMPLETE`). Revenons sur le premier écouteur :

```
chargeurFond.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
      ➥ chargementENCOURS);
```

Dans le gestionnaire d'événements, nous pouvons lire `ProgressEvent` et non simplement `Event`. Seule la classe `ProgressEvent` permet de disposer de l'événement `PROGRESS` qui désigne la période durant laquelle le chargement s'effectue et autorise, seul, l'exécution d'une fonction pendant ce chargement.

La fonction appelée pendant le chargement est `chargementENCOURS` :

```
function chargementENCOURS(evt:ProgressEvent) {
   var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded / evt.current
   ➥ Target.bytesTotal);
   chargement_mc.barre_mc.scaleX=valeurPourcentage;
   chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+"%";
}
```

Dans cette fonction, nous commençons par définir une variable de type nombre qui enregistre une valeur correspondant à la progression du chargement. Cette valeur décimale est comprise entre 0 (0 % de progression) et 1 (100 % de progression). Elle est déterminée en calculant, pour l'objet en cours de chargement (`evt.currentTarget`), le nombre de bytes déjà chargés divisé par le nombre de bytes total à charger.

Par exemple, si le fichier pèse 60 Ko (61 440 bytes) et que 15 Ko sont actuellement chargés (15 360 bytes), le pourcentage du chargement effectué est de 15 630/61 440, soit 0,25 (ou 25 %). Lorsque le chargement sera terminé, la valeur sera de 61 440/61 440, soit 1.

Suite à cela, la variable `valeurPourcentage` est utilisée pour définir l'échelle de déformation en X sur l'objet `barre_mc` qui représente la jauge de progression du symbole `chargement_mc` :

```
chargement_mc.barre_mc.scaleX=valeurPourcentage;
```

Plus loin, dans la même fonction, une autre instruction utilise cette valeur pour définir le texte à afficher dans le champ dynamique nommé `pourcentage_txt`, situé à l'intérieur du symbole `chargement_mc` :

```
chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+"%";
```

Le texte résulte de la variable `valeurPourcentage` augmentée et arrondie. Nous concaténons, avec le signe plus (+), le caractère pourcentage (%) pour que le chiffre obtenu soit accompagné à l'écran de cette unité de valeur.

Lorsque le chargement atteint la valeur de 100, le texte affiche donc 100 %, le symbole `barre_mc` retrouve son échelle initiale et la fonction `chargementCOMPLET` est alors exécutée :

```
// COMPLETE
chargeurFond.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
   cible_mc.addChild(chargeurFond);
   chargement_mc.visible=false;
}
```

Dans cette fonction, en plus d'afficher le contenu chargé avec `addChild`, nous rendons invisible le symbole `chargement_mc`.

En publiant le document localement sur votre ordinateur, vous ne visualisez probablement pas la progression du chargement, car le fichier appelé est déjà présent sur votre poste et ne requiert donc aucune attente pour être affiché. Pour mieux vous rendre compte de l'efficacité de la jauge de chargement, dans le menu Affichage situé en haut de votre écran et disponible lors de la publication du document (Cmd+Entrée pour Mac ou Ctrl+Entrée pour Windows), activez l'option Simuler le téléchargement. Si le réglage de la bande passante est

correctement configuré, vous pourrez alors visualiser la progression du chargement comme si le fichier était en ligne.

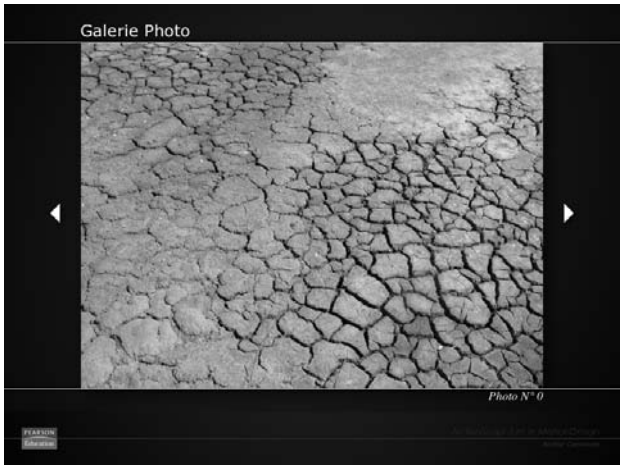
À retenir

- Pour créer une jauge de chargement, nous utilisons `contentLoaderInfo`, une propriété spécifique associée à la classe `ProgressEvent`.
- La valeur obtenue pour déterminer la progression du chargement peut être distribuée à travers différentes propriétés pour redimensionner un symbole (jauge) ou modifier la valeur d'un texte (pourcentage), par exemple.
- Lorsque le chargement est terminé, nous affichons le contenu chargé et nous masquons la jauge de chargement.
- Pour animer l'échelle du symbole qui sert de jauge, il faut bien définir l'emplacement de son centre géométrique de manière à l'étirer correctement.

Réaliser une galerie d'images externalisées

Pour réaliser une galerie d'images à l'aide d'ActionScript, nous utilisons un chargeur. À chaque itération appelée en activant un bouton suivant ou retour, nous remplaçons l'image chargée par une nouvelle. Pour que la gestion de ce dispositif reste simple, nous organisons les fichiers et leurs noms de sorte que le processus puisse être facilement automatisé. Ainsi, si nous rassemblons les images de la galerie dans un même répertoire, et que nous nommons ces images avec des numéros correspondant respectivement à leur ordre d'apparition, et en commençant par zéro, il nous suffit alors d'appeler chaque image en incrémentant ou en diminuant une valeur passée en paramètre de l'URL d'un chargeur de contenu. Nous pouvons aussi utiliser cette valeur pour renseigner l'internaute sur le numéro d'image en cours d'affichage, par exemple. C'est ce que nous présentons dans cette section (voir Figure 5.7).

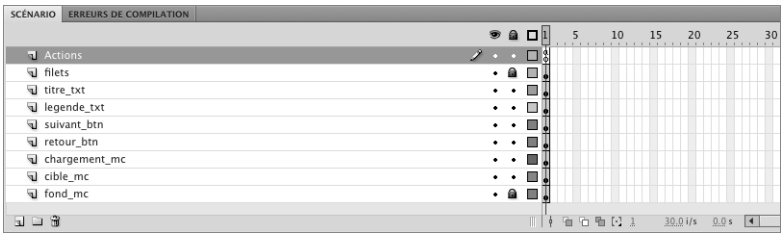
Figure 5.7
Aperçu de la galerie à la publication.



Exemples > ch5_galeriesImages_3.fla

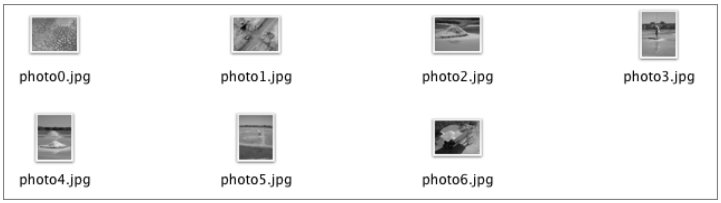
Dans la scène principale du document "ch5_galeriesImages_3.fla", au-dessus du calque `fond_mc`, apparaît le symbole `cible_mc`. Il est vide et sert de conteneur pour les images de la galerie (voir Figure 5.8). Ce symbole possède déjà des propriétés, telles que l'application d'un filtre d'ombre portée. Nous retrouvons la jauge de chargement présentée à la section précédente, puis deux boutons (`retour_btn` et `suivant_btn`) qui servent à appeler les nouvelles images. Au-dessus, deux champs de texte dynamiques permettent d'inscrire pour chaque photo, un titre et une légende en fonction des itérations. Enfin, deux filets soulignent la composition, mais ne sont pas impliqués dans les actions.

Figure 5.8
Aperçu du scénario de la scène principale.



À l'intérieur de notre dossier de travail, le répertoire "images" contient un sous-répertoire nommé "galerie". Celui-ci contient 7 images respectivement nommées photo0.jpg, photo1.jpg, photo2.jpg, etc., de largeurs différentes (voir Figure 5.9).

Figure 5.9
Aperçu des images externalisées de la galerie.



Dans le calque nommé `actions`, nous pouvons lire le code suivant :

```
//-------------------- Chargement initial

// CHARGEMENT
var chargeurPhoto:Loader= new Loader();
var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo0.jpg");
chargeurPhoto.load(cheminPhoto);

// PROGRESSION
chargeurPhoto.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➥ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
   chargement_mc.visible=true;
   var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded /
   ➥ evt.currentTarget.bytesTotal);
   chargement_mc.barre_mc.scaleX=valeurPourcentage;
```

```
   chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+"%";
}

// COMPLETE
chargeurPhoto.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET1);
function chargementCOMPLET1(evt:Event) {
   cible_mc.addChild(chargeurPhoto);
   chargement_mc.visible=false;
   cible_mc.x=(stage.stageWidth/2)-(cible_mc.width/2);
   cible_mc.y=(stage.stageHeight/2)-(cible_mc.height/2)-25;
}

//-------------------- Boutons de navigation

var i:Number=0;
var nombreDePhotos:Number=7;
titre_txt.text="Galerie Photo";
index_txt.text="Les marais salants de Guérande";

// Bouton SUIVANT
suivant_btn.addEventListener(MouseEvent.CLICK,afficherPhotoSuivante);

function afficherPhotoSuivante(Event:MouseEvent){
   i++;
   if (i>=nombreDePhotos) {
      i=0;
   }
   chargerPhoto();
}

// Bouton RETOUR
retour_btn.addEventListener(MouseEvent.CLICK,afficherPhotoRetour);

function afficherPhotoRetour(Event:MouseEvent){
   i--;
   if (i<0) {
      i=nombreDePhotos-1;
   }
   chargerPhoto();
}

function chargerPhoto () {
   var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo"+i+".jpg");
   chargeurPhoto.load(cheminPhoto);
   index_txt.text="Photo N° "+i;
}
```

La première partie de ce code reprend essentiellement le chargement initial d'une image, que nous avons abordé précédemment. L'URL cible simplement un nouveau dossier et affiche la première image de la série, nommée "photo0.jpg" :

```
var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo0.jpg");
```

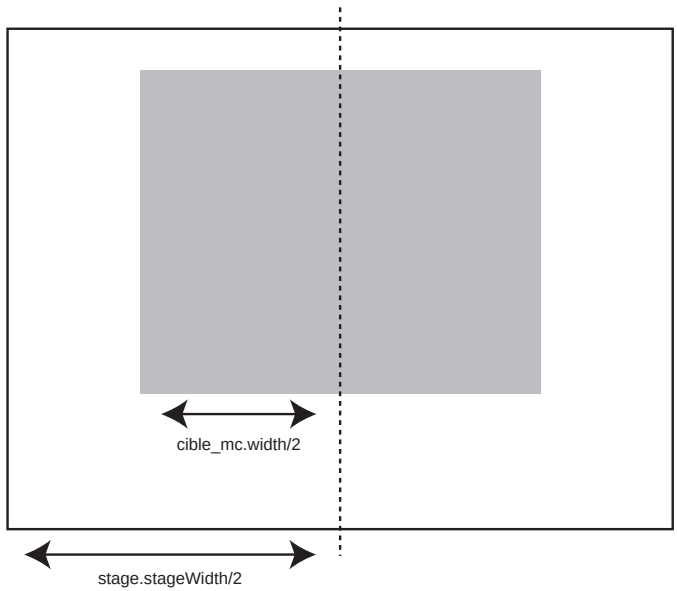
Dans le gestionnaire `PROGRESS`, nous réactivons l'affichage de la jauge de sorte qu'elle puisse réapparaître aussi pour les images suivantes :

```
chargement_mc.visible=true;
```

Enfin, une fois le chargement complet, nous redéfinissons le positionnement du symbole conteneur `cible_mc` en X et Y. Dans notre contexte, cela permet de recentrer l'image dans la page quelle que soit sa largeur. Pour ce faire, nous spécifions, dans le premier groupe de parenthèses, que la position du conteneur correspond à la largeur de la fenêtre du document Flash, divisée par 2. Puis, nous retranchons, dans le deuxième groupe de parenthèses, la largeur du conteneur lui-même, divisée aussi par 2. Si nous ne divisions pas par 2, la cible serait collée à droite de la limite du document, car la valeur alors prise en compte pour caler l'objet serait la largeur de la scène moins celle de l'objet. En divisant par 2 chacune d'entre elles, puisque la moitié d'une largeur équivaut à son centre, nous centrons ce conteneur dans le document (voir Figure 5.10) :

```
cible_mc.x=(stage.stageWidth/2)-(cible_mc.width/2);
```

Figure 5.10
Schéma du positionnement du conteneur.



Notez que ce calcul ne peut se faire qu'à partir du moment où les dimensions de `cible_mc` sont connues, c'est-à-dire, uniquement lorsqu'une photo y est chargée. À défaut, c'est sa dimension initiale qui sera lue, soit 0 pixel de large. La position d'un conteneur est déterminée par rapport à son centre géométrique, toujours calé en haut et à gauche dans le cas d'un chargement dynamique. Elle serait effectivement décalée de la moitié de la largeur de l'image importée, si nous l'appliquions avant le chargement de l'image.

Le principe est décliné pour le positionnement vertical, en Y, avec toutefois un décalage de –25 pixels, pour rehausser le conteneur vers le sommet du document :

```
cible_mc.y=(stage.stageHeight/2)-(cible_mc.height/2)-25;
```

Info

Centrer les images avec le composant `UILoader`. *Il est possible de centrer les images chargées dynamiquement sans avoir à calculer leur position. Remplacez simplement le conteneur `MovieClip` `cible_mc` que nous utilisons dans cet exemple par un composant `UILoader`. Ce composant est disponible depuis la fenêtre des composants (Fenêtre > Composants), dans le groupe de composants*

nommé "User Interface". Pour plus d'informations sur ce composant, reportez-vous à l'aide de Flash (F1) et saisissez dans le moteur de recherche de l'aide, le terme `UILoader` ou bien consultez directement l'URL de la notice de ce composant à cette adresse : **http://help.adobe.com/fr_FR/ActionScript/3.0_UsingComponentsAS3/WS5b3ccc516d4fbf351e63e3d118a9c65b32-7f9d.html**.

Plus loin dans le code, au niveau du commentaire Boutons de navigation, nous initialisons quelques valeurs :

```
//-------------------- Boutons de navigation
var i:Number=0;
var nombreDePhotos:Number=7;
titre_txt.text="Galerie Photo";
index_txt.text="Les marais salants de Guérande";
```

En premier lieu, `i` désigne un nombre de valeur 0. À chaque clic sur le bouton suivant ou retour, nous incrémentons ou diminuons cette valeur de sorte qu'elle désigne, dans la liste des images, l'image correspondant à la valeur appelée. Pour démarrer, cette valeur est initialisée à 0 et désigne l'image "photo0.jpg". À chaque clic, cette valeur est donc modifiée. Lorsque la valeur atteint le seuil correspondant à la dernière image de la série, nous l'initialisons à 0, pour redémarrer la boucle. Inversement, lorsque le bouton retour appelle l'image qui précède la première image de la série, la boucle renvoie la valeur qui correspond à la dernière image (`i=nombreDephotos-1`). Nous retranchons 1 à la valeur `nombreDePhotos` : la boucle démarrant à 0, si nous conservions la valeur initiale, nous obtiendrions une itération de trop en regard du nombre d'images disponibles, ce qui provoquerait une erreur d'affichage.

La deuxième variable `nombreDePhotos` indique le nombre de photos disponibles dans la galerie. Vous devez renseigner cette valeur manuellement. Nous verrons, en abordant le XML, que cette valeur peut être renseignée automatiquement. Actuellement, nous comptons 7 photos.

Enfin, nous initialisons les textes des champs dynamiques `titre_txt` et `index_txt` distribués sur la scène. Pour chaque bouton, nous ajoutons à présent un écouteur et une fonction :

```
// Bouton SUIVANT
suivant_btn.addEventListener(MouseEvent.CLICK,afficherPhotoSuivante);
```

Le bouton `suivant_btn` appelle la fonction `afficherPhotoSuivant` :

```
function afficherPhotoSuivante(Event:MouseEvent){
      i++;
   if (i>=nombreDePhotos) {
      i=0;
   }
   chargerPhoto();
}
```

La fonction commence par incrémenter la valeur `i` (`i++`) préalablement initialisée à 0. La valeur `i` devient donc 1, au premier clic, puis 2 au second clic et ainsi de suite.

Ensuite, nous indiquons que si la valeur de `i` dépasse le nombre de photos disponibles (`i>nombreDePhotos`), nous la réinitialisons à 0. Ainsi, lorsque la boucle est terminée, c'est la première image qui est à nouveau affichée et la boucle peut continuer d'être incrémentée jusqu'à ce que la condition l'initialise de nouveau, et ainsi de suite.

À la fin du bloc d'instruction du premier bouton, nous faisons référence à une fonction développée plus loin (`chargerPhoto`). Nous y revenons.

Nous identifions ensuite une nouvelle fonction (`afficherPhotoRetour`) déclinaison de cette même fonction pour le bouton de retour :

```
// Bouton RETOUR
retour_btn.addEventListener(MouseEvent.CLICK,afficherPhotoRetour);
function afficherPhotoRetour(Event:MouseEvent){
   i--;
   if (i<0) {
      i=nombreDePhotos-1;
   }
   chargerPhoto();
}
```

Dans la fonction `afficherPhotoRetour`, nous inversons le sens de l'incrémentation en remplaçant les signes plus par moins (`i--`). La valeur ainsi diminue de 1 à chaque itération. Puis, si cette valeur atteint le seuil limite qui correspond à la première image (`0`), alors, nous l'initialisons à la valeur de la variable `nombreDePhotos` de sorte que l'animation boucle à nouveau sur elle-même.

À la fin du programme, nous plaçons une fonction autonome qui rassemble les instructions communes appelées par les deux boutons. Cela permet de simplifier le codage du projet en évitant les actions redondantes. De plus, cela facilitera la maintenance :

```
function chargerPhoto () {
   var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo"+i+".jpg");
   chargeurPhoto.load(cheminPhoto);
   index_txt.text="Photo N° "+i;
}
```

La fonction `chargerPhoto()` lance le chargement de l'image et modifie la valeur du champ de texte dynamique en fonction de la valeur de l'image chargée, définie par `i`.

Plus précisément, nous redéfinissons l'URL `cheminPhoto` appelée par le chargeur du départ, nommé `chargeurPhoto`. Nous utilisons la valeur de `i` pour déterminer le numéro de l'image à afficher, en intégrant cette valeur au sein de la chaîne de caractères de l'URL. Il est nécessaire, pour le bon fonctionnement du programme, que les images stockées dans le répertoire possèdent la même racine de nom (`photo`), et que la première d'entre elle se termine par le numéro 0 qui correspond à la première valeur de `i`.

Le langage

L'ordre numérique en ActionScript

En ActionScript, la valeur du premier niveau de toute chaîne démarre toujours à 0. Un objet affiché sur la scène sera accessible avec `getChildAt(0)`. Le premier nœud d'un fichier XML sera appelé avec `documentXML.childNodes[0]`. Le premier élément d'un tableau sera accessible avec `nomDuTableau[0]`, et ainsi de suite. Pour homogénéiser les actions et simplifier la gestion d'actions répétitives, nous gérons aussi les images en associant au nom, de la première d'entre elles, la valeur 0. Il est alors plus simple de gérer les images en associant leur nom à une variable d'incrémentation, comme `i`, qui peut en même temps affecter d'autres méthodes ou conteneurs.

Nous rappelons alors le chargeur (`chargementphoto.load(cheminPhoto)`) afin qu'il relance le chargement à partir de l'URL nouvellement désignée.

Il n'est pas nécessaire de rappeler la méthode `addChild` car l'objet `chargeurPhoto` est déjà présent dans la liste d'affichage (donc sur la scène), grâce au premier `addChild` invoqué lors du chargement de la première image. De même, il est inutile de relancer les écouteurs pour les événements `PROGRESS` et `COMPLETE` qui demeurent toujours actifs tant que nous ne les avons pas neutralisés à l'aide de la méthode `removeEventListener`.

En publiant la galerie sur un serveur distant, vous remarquerez que la jauge réapparaît pour chaque nouveau chargement parce que nous avons réactivé la propriété `visible` sur `true` à l'exécution du chargement. De même, la jauge disparaît lorsque vous affichez une deuxième fois chaque image, car celles-ci sont maintenant chargées dans le cache de votre navigateur. Leur chargement devient donc instantané.

Enfin, toujours dans cette fonction, nous modifions le contenu du champ de texte dynamique `index_txt` en y inscrivant le numéro de la photo chargée, selon le même principe que pour la définition de l'URL.

À retenir

- Il est possible de réaliser une galerie dynamique d'images en jouant sur la manière de nommer les fichiers appelés. Par exemple, en mixant une racine commune avec un nombre qui reflète l'ordre d'apparition.
- Pour appeler différents contenus externalisés ou modifier des valeurs, nous pouvons utiliser une variable de type nombre qu'il suffit d'incrémenter à chaque itération pour générer de nouvelles valeurs.
- Un gestionnaire d'événements est toujours actif tant que celui-ci n'a pas été neutralisé par la méthode `removeEventListener`.
- Il est possible de centrer une image chargée dynamiquement en calculant sa position par rapport aux dimensions de la fenêtre du document Flash (`stage`) ou en utilisant le composant `UILLoader`.

Réaliser une galerie d'images avec XML

Le recours à un fichier XML, pour le développement d'une galerie photo comme nous le proposons ici, permet de centraliser les informations rattachées à chaque visuel dans un seul document, éditable au format texte, et de ne plus avoir à publier à nouveau le document Flash lorsqu'une donnée est modifiée.

La gestion d'une animation Flash avec un flux XML vous permet aussi de travailler plus facilement en équipe avec un développeur back-office par exemple (qui gère des systèmes de gestion de contenu côté serveur, avec les langages PHP/MySQL par exemple, et peut rendre facilement disponibles ses requêtes sous la forme de flux XML).

La structure d'un document XML autorise enfin d'associer, pour chacun des éléments, un nombre indéterminé de données, sans avoir à les traiter toutes systématiquement.

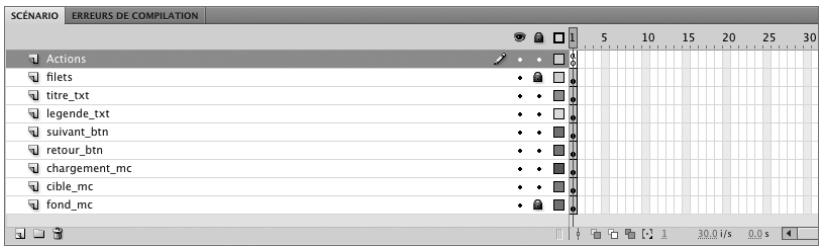
Dans cette section, nous présentons une galerie photo associée à un fichier XML contenant les références aux images, les titres ainsi qu'une légende pour chacune d'entre elles. En cliquant sur les boutons suivant et retour, nous évoluons directement au cœur de l'arborescence du fichier XML alors importé (voir Figure 5.11).

Figure 5.11
Aperçu de la galerie à la publication.

Exemples > ch5_galeriesImages_4.fla

Dans la scène principale du document "ch5_galeriesImages_4.fla", au-dessus du calque `fond_mc`, apparaît le même symbole `cible_mc` vide, qui sert de conteneur pour les images de la galerie. Les autres éléments sont similaires. Deux champs de texte dynamiques `titre_txt` et `legende_txt` sont distribués de part et d'autre de la zone d'affichage `cible_mc`. Deux boutons `suivant_btn` et `retour_btn` permettent de contrôler la navigation. Enfin, le chargeur développé en début de chapitre demeure présent et reste également actif pour chaque nouvelle entrée appelée (voir Figure 5.12).

Figure 5.12
Aperçu du scénario de la scène principale.



Dans le calque nommé actions, nous pouvons lire le code suivant :

```
//-------------------- Chargement initial PHOTO AGRANDIE
// CHARGEMENT
var chargeurPhoto:Loader= new Loader();
var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo0.jpg");
chargeurPhoto.load(cheminPhoto);

// PROGRESSION
chargeurPhoto.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➥ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
   chargement_mc.visible=true;
   var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded / evt.current
   ➥ Target.bytesTotal);
   chargement_mc.barre_mc.scaleX=valeurPourcentage;
   chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+"%";
}

// COMPLETE
chargeurPhoto.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET1);
function chargementCOMPLET1(evt:Event) {
   cible_mc.addChild(chargeurPhoto);
   chargement_mc.visible=false;
   cible_mc.x=stage.stageWidth/2-cible_mc.width/2;
   cible_mc.y=stage.stageHeight/2-cible_mc.height/2-25;
}

//-------------------- Chargement du XML

var i:Number=0;

var chargeurXML:URLLoader = new URLLoader();
chargeurXML.load(new URLRequest("xml/galerie.xml"));
chargeurXML.addEventListener(Event.COMPLETE, XMLComplet);

function XMLComplet(evt:Event){
   var donneesXML:XML=new XML(evt.target.data);
   var NombreDeNoeudsDansLeXML:Number=donneesXML.photo.length();

   // RETOUR
   retour_btn.addEventListener(MouseEvent.MOUSE_DOWN, retourDOWN);
   function retourDOWN(evt:MouseEvent) {
      i--;
      if (i<0) {
         i=NombreDeNoeudsDansLeXML-1;
      }
      distribuerValeurs();
   }
   // SUIVANT
   suivant_btn.addEventListener(MouseEvent.CLICK, suivant);
   function suivant(evt:MouseEvent) {
```

```
            i++;
            if (i==NombreDeNoeudsDansLeXML) {
               i=0;
            }
            distribuerValeurs();
         }
         function distribuerValeurs () {
         titre_txt.htmlText=donneesXML.photo[i].titre.toString();// titre
         legende_txt.htmlText=donneesXML.photo[i].legende.toString();// légende
         cheminPhoto=new URLRequest(donneesXML.photo[i].image.toString());
         ➥ // image agrandie
         chargeurPhoto.load(cheminPhoto);
        }
    }
```

Pour comprendre le principe de la gestion d'un flux XML, nous devons retenir l'idée que les données centralisées dans le fichier XML ne peuvent être distribuées dans le Flash qu'une fois le XML entièrement chargé. C'est pourquoi nous mettons les contenus en forme suite à l'événement `COMPLETE`.

Pour traiter les données issues d'un fichier XML, nous disposons alors de deux méthodes. Soit nous utilisons directement les valeurs chargées par le XML au sein de la fonction liée au chargement du XML. Soit nous créons des variables globales, c'est-à-dire que nous les typons en dehors de toute fonction, pour en modifier ensuite les valeurs avec les données du XML importé. Dans ce cas, nous pouvons exploiter les valeurs modifiées par le XML depuis toute autre fonction présente dans notre programme.

Dans cet ouvrage, nous abordons la première méthode. Pour la seconde, nous vous invitons à consulter des publications plus spécialisées sur l'interfaçage dynamique, comme le livre très pointu de Thibault Imbert (éd. Pearson) ou l'ouvrage plus accessible d'Anne Tasso (éd. Eyrolles). Nous fournissons les références complètes de ces deux livres en fin d'ouvrage.

Dans ce document, nous chargeons le fichier XML relatif à la galerie, qui comporte la structure suivante :

```
<?xml version = '1.0'  encoding="utf-8" ?>
<galerie>
   <photo>
      <titre>GUERANDE</titre>
      <legende>Sol craquelé</legende>
      <image>images/galerie/photo0.jpg</image>
      <vignette>images/galerie/vignettes/photo0-vignette.jpg</vignette>
   </photo>
   <photo>
      <titre>GUERANDE</titre>
      <legende>Gorgue, canal amenant l'eau chargée en sel dans les tables salantes
      ➥ </legende>
      <image>images/galerie/photo1.jpg</image>
      <vignette>images/galerie/vignettes/photo1-vignette.jpg</vignette>
   </photo>
   <photo>
      <titre>GUERANDE</titre>
      <legende>Amas de sel</legende>
      <image>images/galerie/photo2.jpg</image>
      <vignette>images/galerie/vignettes/photo2-vignette.jpg</vignette>
   </photo>
```

```
    <photo>
       <titre>GUERANDE</titre>
       <legende>Cristallisoir</legende>
       <image>images/galerie/photo3.jpg</image>
       <vignette>images/galerie/vignettes/photo3-vignette.jpg</vignette>
    </photo>
    <photo>
       <titre>GUERANDE</titre>
       <legende>Saliculture</legende>
       <image>images/galerie/photo4.jpg</image>
       <vignette>images/galerie/vignettes/photo4-vignette.jpg</vignette>
    </photo>
    <photo>
       <titre>GUERANDE</titre>
       <legende>Rammasage du sel</legende>
       <image>images/galerie/photo5.jpg</image>
       <vignette>images/galerie/vignettes/photo5-vignette.jpg</vignette>
    </photo>
    <photo>
       <titre>GUERANDE</titre>
       <legende>Outils du paludier</legende>
       <image>images/galerie/photo6.jpg</image>
       <vignette>images/galerie/vignettes/photo6-vignette.jpg</vignette>
    </photo>
</galerie>
```

Ce document comporte une structure classique composée d'un nœud principal appelé ici galerie. Cette structure affiche autant de nœuds nommés photo qu'il y a d'images à intégrer dans notre galerie. Chaque nœud `photo`, à son tour, dispose d'entrées respectivement nommées `titre`, `legende`, `image` et `vignette`, véhiculant chacune une valeur que nous distribuons, avec ActionScript, dans les objets mis en scène dans le document Flash. Dans cette section, nous utilisons uniquement les trois premiers éléments de chaque nœud. Le quatrième, vignette, est déployée dans la section suivante.

Créer un fichier XML. *Pour créer un fichier XML, utilisez n'importe quel éditeur de texte et enregistrez le code au format texte avec l'extension xml. Puis, dans la première ligne, spécifiez un encodage de type UTF-8 pour éviter les problèmes d'accents mal interprétés si vous éditez depuis un Macintosh : sur Mac, les fichiers texte sont nativement codés en ISO Mac. Vous devez donc enregistrer le texte avec une option qui permette d'exporter en UTF-8. L'instruction ajoutée dans le code affiche donc :*

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

Dans la suite Adobe, Dreamweaver permet de réaliser facilement des documents XML clairement indentés et propose des assistants à la saisie. Pour plus d'informations sur la structure d'un fichier XML, consultez les ouvrages de Howard Goldberg ou Florent Nolot aux éditions Pearson.

Dans le Flash, depuis la fenêtre Actions, nous activons le chargement d'une première image dans la zone cible, comme vu précédemment :

```
//-------------------- Chargement initial PHOTO AGRANDIE
// CHARGEMENT
var chargeurPhoto:Loader= new Loader();
```

```
var cheminPhoto:URLRequest=new URLRequest("images/galerie/photo0.jpg");
chargeurPhoto.load(cheminPhoto);

// PROGRESSION
chargeurPhoto.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➥ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
   chargement_mc.visible=true;
   var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded /
   ➥ evt.currentTarget.bytesTotal);
   chargement_mc.barre_mc.scaleX=valeurPourcentage;
   chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+"%";
}

// COMPLETE
chargeurPhoto.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET1);
function chargementCOMPLET1(evt:Event) {
   cible_mc.addChild(chargeurPhoto);
   chargement_mc.visible=false;
   cible_mc.x=stage.stageWidth/2-cible_mc.width/2;
   cible_mc.y=stage.stageHeight/2-cible_mc.height/2-25;
}
```

Ensuite, nous activons la gestion du flux XML à partir duquel nous allons définir les contrôles de navigation sur les boutons suivant et retour :

```
//-------------------- Chargement du XML
var i:Number=0;
var ecartEntreVignettes:Number=90;

var chargeurXML:URLLoader = new URLLoader();
hargeurXML.load(new URLRequest("xml/galerie.xml"));
chargeurXML.addEventListener(Event.COMPLETE, XMLComplet);

function XMLComplet(evt:Event) {
   // gestion des données XML
}
```

Avant de charger le fichier XML, nous initialisons un nombre `i`, qui permet de mémoriser l'ordre d'affichage de l'image courante. Cette valeur est initialisée à 0, ce qui correspond de nouveau à l'image chargée par défaut, "photo0.jpg".

Puis, nous importons le fichier XML. Pour ce faire, nous déclarons d'abord un nouveau chargeur (`chargeurXML`), en utilisant cette fois une instance de la classe `URLLoader`, comme nous le ferions pour importer une image. Ce chargeur active le chargement d'un fichier défini à travers l'objet URLRequest, passé directement en paramètre de la méthode `load()`. Cela équivaut à définir l'URL séparément avec `var monChemin:URLRequest=new URLRequest("monURL")`. Les deux méthodes conviennent.

Différence entre `Loader` et `URLLoader`. *La classe* `Loader` *est utilisée pour charger un fichier image ou SWF. Tandis que la classe* `URLLoader` *est employée pour charger un fichier au format texte. La classe* `URLLoader` *est donc la méthode qu'il faut utiliser pour la gestion de données dynamiques.*

Le chargeur `chargeurXML` est associé à un écouteur (avec l'action `addEventListener`) qui, à travers la classe `Event` et l'événement `COMPLETE`, appelle la fonction nommée `XMLComplet` une fois le chargement effectué.

À l'intérieur de cette fonction, nous créons une variable `donneesXML` qui enregistre l'ensemble de l'arbre XML importé. Cela permettra, par la suite, de s'y référer aisément en invoquant cet objet pour distribuer chaque donnée qu'il véhicule, plus loin dans le code.

L'objet créé est un objet de type XML. Il se réfère à la classe `Event` active, *via* l'identifiant `evt`, et y cible les données courantes (`target.data`), données alors aspirées par l'écouteur auquel cette classe se rattache.

```
function XMLComplet(evt:Event) {
   var donneesXML:XML=new XML(evt.target.data);
   var NombreDeNoeudsDansLeXML:Number=donneesXML.photo.length();
}
```

Enfin, nous déclarons une première valeur à partir de l'objet XML que nous venons de créer, pour identifier le nombre de nœuds disponibles dans ce fichier. Pour cela, nous nous référons à l'objet XML avec `donnesXML`. Puis, nous spécifions le nom du nœud de premier niveau (photo) et détectons la longueur (la quantité de nœuds) présente dans le document, grâce à la méthode `length()`.

Lire le XML. *Pour lire les données contenues dans un fichier XML, selon le type de données relevées, procédez comme suit :*

- *Pour atteindre un nœud précis, dont vous connaissez l'ordre d'apparition, passez son numéro entre crochets, comme ceci :* `donneesXML.photo[numeroDuNoeud].toString()`. *L'ordre d'affichage des nœuds dans l'arborescence du fichier XML démarre toujours à la valeur 0. Pour atteindre le premier nœud, nous inscrivons* `nomDuXML.nomDuNoeuds[0]`.
- *Si la structure comporte plusieurs nœuds imbriqués, ciblez chaque niveau individuellement, comme ceci :*
  `donneesXML.photo[numeroDuNoeud].image[numeroDuNoeudDeSecondNiveau]toString()` ;
- *Si le nœud porte la structure suivante :* `<photo largeur="600" hauteur="450"> <image>images/photo0.jpg</image> </photo>`, *pour cibler la valeur contenue dans l'attribut* `largeur`, *utilisez la syntaxe* `donneesXML.photo[numeroDuNoeud].@largeur.toString()`.
- *Si le nœud porte cette structure-ci :* `<photo largeur="600" hauteur="450"> <image>images/photo0.jpg</image> </photo>`, *pour cibler la valeur contenue entre les balises* `<image>` *et* `</image>`, *utilisez la syntaxe* `donneesXML.photo[numeroDuNoeud].image.toString()`. *Attention toutefois, toute* `XMLListe` *se comporte comme un XML si elle ne contient qu'un seul élément. Dans le cas contraire, il faudrait écrire* `donneesXML.photo[numeroDuNoeud].image[0].toString()`.
- *Si le nœud doit comporter des balises HTML, enveloppez le HTML dans une description de type CDATA :* `<photo> <legende> <![CDATA[Marais salants de <b>GUERANDE</`

b>]]> </legende> </photo>, et ciblez la valeur contenue entre les balises <legende > et </legende>, avec la syntaxe donneesXML.photo[numeroDuNoeud].legende.toString().

- Pour cibler enfin plus spécifiquement un nœud dont vous connaissez la valeur d'un attribut : <photo largeur="600" hauteur="450"> <image>images/photo0.jpg</image> </photo>, utilisez la syntaxe suivante : donneesXML.photo[numeroDuNoeud].(@largeur=="600").toString().
- Pour enfin convertir le contenu d'une balise en chaîne de caractères, spécifiez .toString(), qui demeure facultatif.

Une fois ces informations définies, nous pouvons distribuer les données à travers les objets du document Flash :

```
// RETOUR
retour_btn.addEventListener(MouseEvent.MOUSE_DOWN, retourDOWN);
function retourDOWN(evt:MouseEvent) {
   i--;
   if (i<0) {
      i=NombreDeNoeudsDansLeXML-1;
   }
   distribuerValeurs();
}
```

Dans la fonction XMLComplet, nous définissons successivement les écouteurs et les fonctions rattachées aux boutons suivant et retour de l'interface, ceci afin de directement pouvoir y exploiter les valeurs du XML.

Dans la fonction retourDOWN, activée lorsque l'utilisateur clique sur le bouton retour_btn, nous commençons par réduire la valeur de i, initialement portée à 0 (i--). Si cette valeur est inférieure à 0 (i<0), une fois diminuée, elle est aussitôt ramenée à la valeur correspondant au seuil limite de la galerie, c'est-à-dire au nombre de nœuds présent dans le fichier XML, en l'occurrence 7 moins un (i=nombreDeNoeudDansLeXML-1).

À la fin de la fonction, nous appelons une autre fonction (distribuerValeurs) qui rassemble les instructions communes aux deux boutons, comme vu précédemment.

Nous déclinons ensuite l'écouteur et la fonction pour le bouton suivant_btn, avant de refermer la fonction principale associée au flux XML :

```
// SUIVANT
suivant_btn.addEventListener(MouseEvent.CLICK, suivant);
function suivant(evt:MouseEvent) {
   i++;
   if (i==NombreDeNoeudsDansLeXML) {
      i=0;
   }
   distribuerValeurs();
}
```

Pour le bouton suivant_btn, nous inversons les valeurs définies pour la condition if, en spécifiant que la valeur de i doit revenir à 0 si celle-ci atteint la valeur correspondant au nombre de nœuds dans le XML (donc, lorsque i vaut 7). Ainsi, nous passons directement de la valeur 6 à 0, ce qui nous permet de créer un effet de boucle sur le déroulement des images pour notre galerie.

Enfin, la fonction `distribuerValeurs` qui termine le programme rassemble les instructions communes pour les deux boutons :

```
function distribuerValeurs () {
   titre_txt.htmlText=donneesXML.photo[i].titre.toString();// titre
   legende_txt.htmlText=donneesXML.photo[i].legende.toString();// légende
   cheminPhoto=new URLRequest(donneesXML.photo[i].image.toString());// image agrandie
   chargeurPhoto.load(cheminPhoto);
}
```

Une fois la gestion du nombre `i` assurée, il reste à passer les valeurs contenues dans les éléments du XML vers les objets de l'interface Flash. Pour chacune des trois instructions qui ponctuent la fonction, nous faisons référence à des nœuds de l'objet `donneesXML`, en y ciblant plus spécifiquement les balises `titre`, `legende` et `image`. Pour convertir toutefois les valeurs en chaîne de caractères, et éviter d'importer les balises qui enveloppent notre sélection dans le fichier XML, nous utilisons la méthode `toString()`.

Vous remarquez que le chemin qui appelle les images est enregistré dans le document XML. Nous aurions très bien pu utiliser aussi une référence au fichier image en passant la valeur de `i` directement dans l'URL de l'image à importer, comme suit :

```
cheminPhoto=new URLRequest("images/galerie/photo"+i+".jpg");
```

Mais en appelant l'entrée contenue dans le XML, nous évitons de devoir publier à nouveau le document Flash si une modification devait être apportée dans l'organisation des fichiers. Nous permettons en outre de centraliser la gestion des données dans un seul document, le XML.

À retenir

- *Pour optimiser la gestion des données dans un document Flash, il est plus confortable de les isoler dans un fichier XML. Cela évite de publier le document Flash à chaque modification des données.*
- *Les données isolées dans un fichier XML ne peuvent être distribuées que lorsque le XML est entièrement chargé. Il faut donc, soit les distribuer dans la fonction exécutée à l'issue du chargement du XML, soit les stocker dans des variables globales.*
- *Pour éviter que l'incrémentation d'une variable nombre (`i`) ne risque de faire référence à un nœud inexistant, nous pouvons interrompre ou réinitialiser l'incrémentation à l'aide d'une simple instruction `if` conditionnelle.*
- *Nous ciblons le nœud d'un document XML en nous référant à l'arbre racine préalablement identifié, puis en ciblant chacun des nœuds descendant de l'arborescence du fichier XML avec la méthode pointée. Il est également possible de cibler ponctuellement un attribut et de vérifier sa valeur. Vous pouvez enfin atteindre un nœud en fonction de son ordre d'apparition dans l'arborescence XML avec une valeur passée en paramètre du nœud ciblé, entre crochets comme `[i]`.*

Interactivité sur les objets dynamiques

Nous savons comment associer des actions sur un objet placé sur la scène, mais lorsque l'objet ou les instructions traitent des éléments gérés dynamiquement, il faut savoir identifier ces objets et ces données afin de permettre toute interaction avec eux.

Dans cette section, nous présentons une déclinaison de notre galerie en plaçant dynamiquement des vignettes dans une zone d'affichage constituée par un symbole de type `MovieClip`. En cliquant sur les vignettes générées dynamiquement, une action effectue un zoom sur un autre symbole et y charge l'image agrandie, correspondant à la vignette activée. Les champs de texte dynamiques sont mis à jour également à chaque action (voir Figure 5.13).

Figure 5.13
Aperçu de la galerie à la publication.

Exemples > ch5_galeriesImages_5.fla

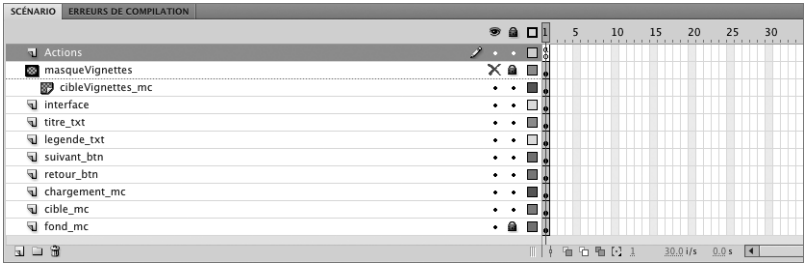
Dans la scène principale du document "ch5_galeriesImages_5.fla", au-dessus du calque `fond_mc`, le symbole `cible_mc` sert de conteneur pour les images agrandies que nous allons charger en cliquant sur les vignettes du bas. Ce conteneur contient aussi un autre symbole, `carreBlanc_mc`. Ce symbole est masqué par défaut avec la propriété `visible` passée sur `false` *via* ActionScript. Ce carré blanc, lorsqu'il est visible, sert de signalétique pour permettre à l'utilisateur d'identifier l'objet cliqué. À chaque clic, nous prévoyons de le redimensionner et de le repositionner sous l'objet cliqué pour le souligner et reconstituer dynamiquement un effet cliqué.

Les autres éléments sont similaires à ceux des sections précédentes. Nous retrouvons la jauge de chargement, les boutons suivant et retour, les champs de texte dynamiques, ainsi qu'un calque qui affiche des éléments graphiques de l'interface nommé interface. Nous

avons ajouté ici, au-dessus des autres calques, un symbole `cibleVignettes_mc`, vide, et un masque, pour y placer les vignettes qui sont gérées dynamiquement. Le masque sert à restreindre la zone d'affichage de ce calque pendant son déplacement, lorsque nous cliquons sur les boutons suivant et retour (voir Figures 5.14 et 5.15).

Figure 5.14
Aperçu de la scène principale.

Figure 5.15
Aperçu du scénario de la scène principale.



Dans le calque nommé `actions`, nous pouvons lire le code suivant :

```
//-------------------- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;


var i:int=0;
var ecartEntreVignettes:Number=90;
var positionCibleVignettesInit:Number=cibleVignettes_mc.x;
cibleVignettes_mc.carreBlanc_mc.x=-2;
cibleVignettes_mc.carreBlanc_mc.y=-2;
cibleVignettes_mc.carreBlanc_mc.visible=false;

//-------------------- Chargement initial PHOTO AGRANDIE
```

```
// CHARGEMENT
var chargeurPhoto:Loader= new Loader();
var cheminPhoto:URLRequest;

// PROGRESSION
chargeurPhoto.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➥ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
  chargement_mc.visible=true;
  var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded / evt.current-
  ➥ Target.bytesTotal);
  chargement_mc.barre_mc.scaleX=valeurPourcentage;
  chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+"%";
}

// COMPLETE
chargeurPhoto.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
  cible_mc.addChild(chargeurPhoto);
  chargement_mc.visible=false;
}

//-------------------- Chargement du XML

var chargeurXML:URLLoader = new URLLoader();
chargeurXML.load(new URLRequest("xml/galerie.xml"));
chargeurXML.addEventListener(Event.COMPLETE, XMLComplet);

function XMLComplet(evt:Event) {
  var donneesXML:XML=XML(evt.target.data);
  var NombreDeNoeudsDansLeXML:Number=donneesXML.photo.length();

  for (i; i<NombreDeNoeudsDansLeXML; i++) {
    // chargement des vignettes
    var chargeurVignette:Loader= new Loader();
    var vignette:String=donneesXML.photo[i].vignette.toString();
    var cheminVignette:URLRequest=new URLRequest(vignette);
    chargeurVignette.load(cheminVignette);
    chargeurVignette.x=ecartEntreVignettes*i;
    chargeurVignette.name=String(i);
    cibleVignettes_mc.addChild(chargeurVignette);

    // actions sur vignettes
    cibleVignettes_mc.addEventListener(MouseEvent.CLICK,clickVignettes);
    function clickVignettes(evt:MouseEvent) {
      cibleVignettes_mc.carreBlanc_mc.visible=true;
      titre_txt.htmlText=donneesXML.photo[evt.target.name].titre.toString();

legende_txt.htmlText=donneesXML.photo[evt.target.name].legende.toString();
      cheminPhoto=new URLRequest("images/galerie/photo"+evt.target.name+".jpg");
      chargeurPhoto.load(cheminPhoto);
      TweenMax.to(cibleVignettes_mc.carreBlanc_mc, 1,
➥ {x:(ecratEntreVignettes*(evt.target.name-1))-2, ease:Strong.easeOut});
// transition carreBlanc
      cible_mc.scaleX=0.1;
      cible_mc.scaleY=0.1;
      cible_mc.x=mouseX;
      cible_mc.y=mouseY;
```

```
        TweenMax.to(cible_mc, 3, {x:100, y:50, scaleX:1, scaleY:1, ease:Strong.easeOut});
      }
    }
  }

  //---------------------- Boutons NAVIGATION

  retour_btn.addEventListener(MouseEvent.MOUSE_DOWN, retourDOWN);
  function retourDOWN(evt:MouseEvent) {
    if (cibleVignettes_mc.x<=positionCibleVignettesInit*2) {
      TweenMax.to(cibleVignettes_mc, 1,
➥ {x:cibleVignettes_mc.x+ecratEntreVignettes, ease:Elastic.easeOut});
    }
  }
  suivant_btn.addEventListener(MouseEvent.CLICK, suivant);
  function suivant(evt:MouseEvent) {
    if (cibleVignettes_mc.x>=0) {
      TweenMax.to(cibleVignettes_mc, 1, {x:cibleVignettes_mc.x-ecratEntre-
      ➥ Vignettes, ease:Elastic.easeOut});
    }
  }
```

Le code est structuré en quatre étapes. Il fonctionne de la manière suivante : nous commençons par importer les classes nécessaires et nous instancions le module de chargement d'images, sans activer de chargement. Ainsi, nous plaçons les fonctions liées au chargement et à la jauge de progression indépendamment de la gestion des données XML. Dans un deuxième temps, nous traitons les données importées par le XML, à travers la fonction `XMLComplet`. Nous permettons ainsi de charger les vignettes dynamiquement et les distribuer dans notre interface avec un positionnement précis. Dans un troisième temps, dans la fonction `XMLComplet`, nous définissons les actions au clic sur les vignettes. Nous terminons notre développement en spécifiant quelques actions, indépendantes des données XML, pour contrôler le déplacement du conteneur de vignettes dans notre dispositif global de navigation. Il n'y a pas de relation entre les données XML importées et ce dispositif de navigation si ce n'est que la répercussion indirecte de la quantité de vignettes chargées dans le conteneur, qui aidera à définir les limites de son positionnement.

Tout d'abord, nous commençons par appeler les classes requises pour les animations `TweenMax` :

```
//-------------------- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;
```

Puis, nous initialisons un certain nombre de variables :

```
var i:Number=0;
var ecartEntreVignettes:Number=90;
var positionCibleVignettesInit:Number=cibleVignettes_mc.x;
cibleVignettes_mc.carreBlanc_mc.x=-2;
cibleVignettes_mc.carreBlanc_mc.y=-2;
cibleVignettes_mc.carreBlanc_mc.visible=false;
```

La première, le nombre `i`, permet de définir l'index (la position courante) de l'image active, chargée dans la zone d'agrandissement `cible_mc`.

La variable `ecartEntreVignettes` sert à déterminer en une seule fois la valeur qui sépare chaque point d'origine de chaque vignette que le script va placer dans `cibleVignettes_mc`. Cette valeur (90) est utilisée à plusieurs reprises. Pour simplifier la gestion du code, nous la véhiculons à travers une variable.

La variable `positionCibleVignetteInit` sert à mémoriser la position de départ du conteneur de vignettes pour être exploitée plus tard dans le calcul des limites du défilement du conteneur de vignettes `cibleVignettes_mc`.

Puis, les trois instructions suivantes affectent le symbole `carreBlanc_mc`, placé dans le symbole `cibleVignettes_mc` :

```
cibleVignettes_mc.carreBlanc_mc.x=-2;
cibleVignettes_mc.carreBlanc_mc.y=-2;
cibleVignettes_mc.carreBlanc_mc.visible=false;
```

Nous déterminons la position de départ du carré blanc en le calant en X et en Y à 2 pixels plus haut et à gauche de l'origine du clip qui le contient (celle du symbole `cibleVignettes_mc`), car le carré blanc mesure 4 pixels de large de plus que les vignettes. En le décalant de 2 pixels en haut et à gauche, nous centrons ce carré blanc. Nous formalisons ainsi un contour signalétique de 2 pixels qui souligne chaque vignette cliquée. À chaque clic de souris sur l'une d'entre elles, l'objet est repositionné sous la vignette. Par défaut, le carré blanc est rendu invisible (`visible=false`), car aucune image n'est encore affichée dans la zone d'agrandissement. Mais il est de nouveau visible en cliquant sur l'une ou l'autre des vignettes.

Puis, nous mettons en place le chargement d'images et la jauge de progression, sans activer toutefois de chargement sur une première image ni l'ajouter sur la scène :

```
//-------------------- Chargement initial PHOTO AGRANDIE

// CHARGEMENT
var chargeurPhoto:Loader= new Loader();
var cheminPhoto:URLRequest;

// PROGRESSION
chargeurPhoto.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
➥ chargementENCOURS);
function chargementENCOURS(evt:ProgressEvent) {
  chargement_mc.visible=true;
  var valeurPourcentage:Number = (evt.currentTarget.bytesLoaded / evt.current-
  ➥ Target.bytesTotal);
  chargement_mc.barre_mc.scaleX=valeurPourcentage;
  chargement_mc.pourcentage_txt.text=(Math.ceil(valeurPourcentage*100))+"%";
}

// COMPLETE
chargeurPhoto.contentLoaderInfo.addEventListener(Event.COMPLETE,
➥ chargementCOMPLET);
function chargementCOMPLET(evt:Event) {
  cible_mc.addChild(chargeurPhoto);
  ➥ chargement_mc.visible=false;
}
```

Une fois les fonctions de gestion du chargement définies, nous importons le document XML, avec le chargeur `chargeurXML`, la méthode `load` qui active le chargement du contenu, et l'écouteur qui appelle la fonction une fois le chargement effectué. Dans cette fonction, nous retrouvons la définition de l'arbre XML véhiculé par la variable `donneesXML` et la longueur de l'arbre, désignée par `nombreDeNoeudsDansLeXML` :

```
//-------------------- Chargement du XML

var chargeurXML:URLLoader = new URLLoader();
chargeurXML.load(new URLRequest("xml/galerie.xml"));
chargeurXML.addEventListener(Event.COMPLETE, XMLComplet);

function XMLComplet(evt:Event) {
  var donneesXML:XML=XML(evt.target.data);
  var NombreDeNoeudsDansLeXML:Number=donneesXML.photo.length();
}
```

À la suite de la définition des premières variables, nous pouvons identifier l'utilisation d'une boucle `for` :

```
for (i; i<NombreDeNoeudsDansLeXML; i++) {
   // actions à répéter autant de fois que d'itérations induites par la boucle.
}
```

Une boucle `for` permet de rassembler, dans un seul constructeur, un ensemble d'instructions répétitives, dont seul un ou quelques paramètres changent. En l'occurrence, seule la valeur véhiculée par le nombre `i` change pour charger, positionner et définir des actions et interactions avec les vignettes.

Dans cette boucle, nous indiquons de considérer la valeur de `i` telle que définie initialement (`i` vaut 0 tel que nous l'avons défini en amont), puis, si `i` reste inférieur au nombre de nœuds présents dans le XML (`i<nombreDeNoeudsDansLeXML`), alors, nous l'incrémentons (`i++`).

Mécanisme d'une boucle `for`. *Une boucle* `for` *présente la structure suivante :*

```
for(valeur de référence ; condition à respecter ; modification de la valeur de
référence){

// instructions à répéter autant de fois que d'itérations dans la boucle.
}
```

À l'intérieur de la boucle `for`, nous plaçons les instructions à reproduire :

```
// chargement des vignettes
var chargeurVignette:Loader= new Loader();
var vignette:String=donneesXML.photo[i].vignette.toString();
var cheminVignette:URLRequest=new URLRequest(vignette);
chargeurVignette.load(cheminVignette);
chargeurVignette.x=ecartEntreVignettes*i;
chargeurVignette.name=String(i);
cibleVignettes_mc.addChild(chargeurVignette);

// actions sur vignettes
cibleVignettes_mc.addEventListener(MouseEvent.CLICK,clickVignettes);
function clickVignettes(evt:MouseEvent) {
```

```
    cibleVignettes_mc.carreBlanc_mc.visible=true;
    titre_txt.htmlText=donneesXML.photo[evt.target.name].titre.toString();
    legende_txt.htmlText=donneesXML.photo[evt.target.name].legende.toString();
    cheminPhoto=new URLRequest("images/galerie/photo"+evt.target.name+".jpg");
    chargeurPhoto.load(cheminPhoto);
    TweenMax.to(cibleVignettes_mc.carreBlanc_mc, 1, {x:(ecratEntreVignettes*
    ➥(evt.target.name-1))-2, ease:Strong.easeOut});
    // transition carreBlanc
    cible_mc.scaleX=0.1;
    cible_mc.scaleY=0.1;
    cible_mc.x=mouseX;
    cible_mc.y=mouseY;
    TweenMax.to(cible_mc, 3, {x:100, y:50, scaleX:1, scaleY:1, ease:Strong.easeOut});
}
```

La première série d'instructions à reproduire définit le chargement des vignettes et utilise, en paramètre de l'URL, la valeur incrémentée de `i`. Ainsi, à chaque itération, chaque image définie dans le XML sera chargée successivement jusqu'à la dernière, stoppée par la condition de la boucle `for` (`i<nombreDeNoeudsDansLeXML`) :

```
// chargement des vignettes
var chargeurVignette:Loader= new Loader();
var vignette:String=donneesXML.photo[i].vignette.toString();
var cheminVignette:URLRequest=new URLRequest(vignette);
chargeurVignette.load(cheminVignette);
chargeurVignette.x=ecartEntreVignettes*i;
chargeurVignette.name=String(i);
cibleVignettes_mc.addChild(chargeurVignette);
```

Pour permettre, par la suite, d'attacher un écouteur à chaque objet placé dynamiquement, nous devons pouvoir identifier chaque objet individuellement. Pour ce faire, nous utilisons la propriété `name`, que nous attachons à chaque objet chargé (`chargeurVignette.name=String(i)`) en affectant, pour valeur de nom, la valeur correspondant à l'ordre d'affichage de chacune de ces vignettes. Ainsi, la vignette 0 se nomme 0, la vignette 1 se nomme 1, et ainsi de suite. Pour cibler chaque objet plus tard, il suffira d'invoquer son nom en utilisant de nouveau la propriété `name`.

Dans ces instructions de chargement des vignettes, nous relevons aussi le positionnement dynamique est également orchestré par la propriété `x`. Nous déterminons cette valeur en multipliant l'écart défini plus haut entre les vignettes (90 pixels) par la valeur de `i` avec `chargeurVignette.x=ecartEntreVignettes*i`. Donc, chaque vignette ajoutée à toute nouvelle itération est placée à 90 pixels de plus que la précédente. Comme chacune d'entre elles mesure précisément 80 pixels de large. Nous obtenons une marge de 10 pixels entre chaque vignette.

Une fois les vignettes affichées, nous ajoutons un écouteur sur l'objet conteneur global `cibleVignettes_mc`. Pour connaître l'objet sur lequel l'utilisateur a cliqué, nous le ciblons grâce à la propriété `name` associée à l'expression `evt.target`. Cette propriété permet d'identifier un objet lorsqu'il est cliqué au sein du conteneur auquel est rattaché l'écouteur :

```
cibleVignettes_mc.addEventListener(MouseEvent.CLICK,clickVignettes);
function clickVignettes(evt:MouseEvent) {
    // actions à exécuter
    trace (evt.target.name)

}
```