PROGRAMMATION FLASH AVEC ACTIONSCRIPT 3.0

par Dominique DOLÉ

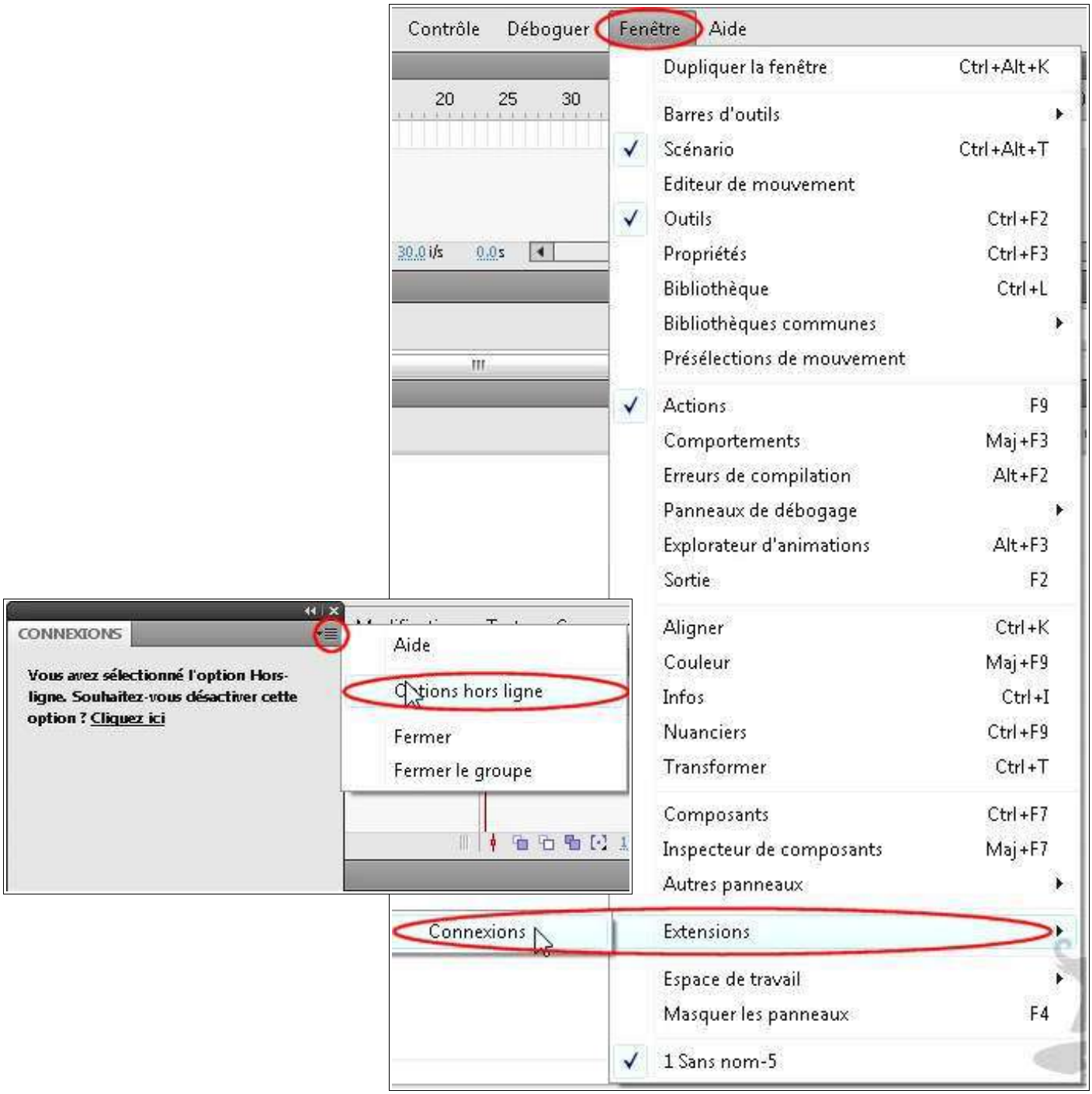
mai 2010

Table des matières

1 La configuration d'Adobe Flash CS4

1.1 Fichier d'aide

Pour utiliser le fichier d'aide en local au lieu de celui sur Internet :



1.2 Organisation d'un projet

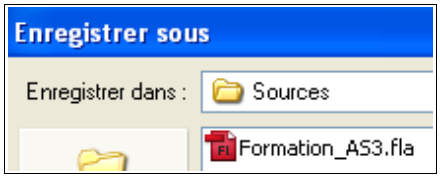
- Il est important de bien structurer les différents éléments relatifs à un projet et notamment de séparer les documents sources de ceux publiés. Pour travailler ainsi, il est conseillé dans le dossier des projets:

- de créer un dossier pour chaque projet (au nom du projet),
- de créer des sous-dossiers (Sources et Publications) pour séparer les sources des publications.

- Placer dans le dossier "Sources" du projet tous les documents qui serviront à le réaliser (images, photos, textes, …).

- Démarrer le programme Flash (Adobe Flash CS4 par exemple), cliquer sur "Fichier Flash (AS 3.0)".

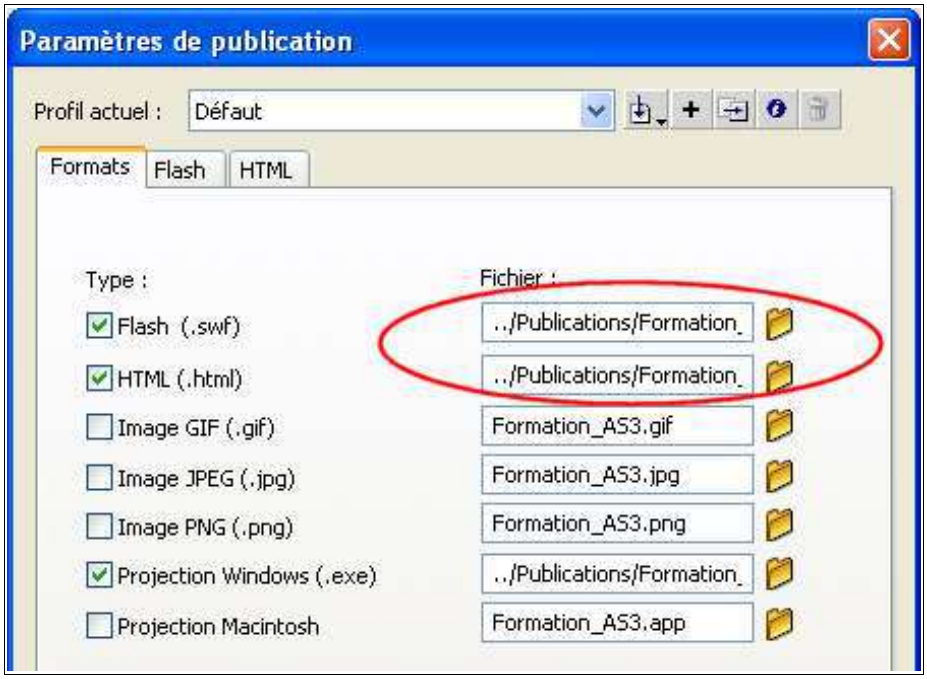
- Cliquer sur "Fichier / Enregistrer sous ..." pour enregistrer le fichier fla dans le sous-dossier "Sources" du dossier du projet précédemment crées.



1.3 Paramètres de publication

- Dans l'onglet **Format** :

  - Modifier les paramètres de publication pour indiquer le **chemin relatif de publication** :



- Dans l'onglet **Flash** :

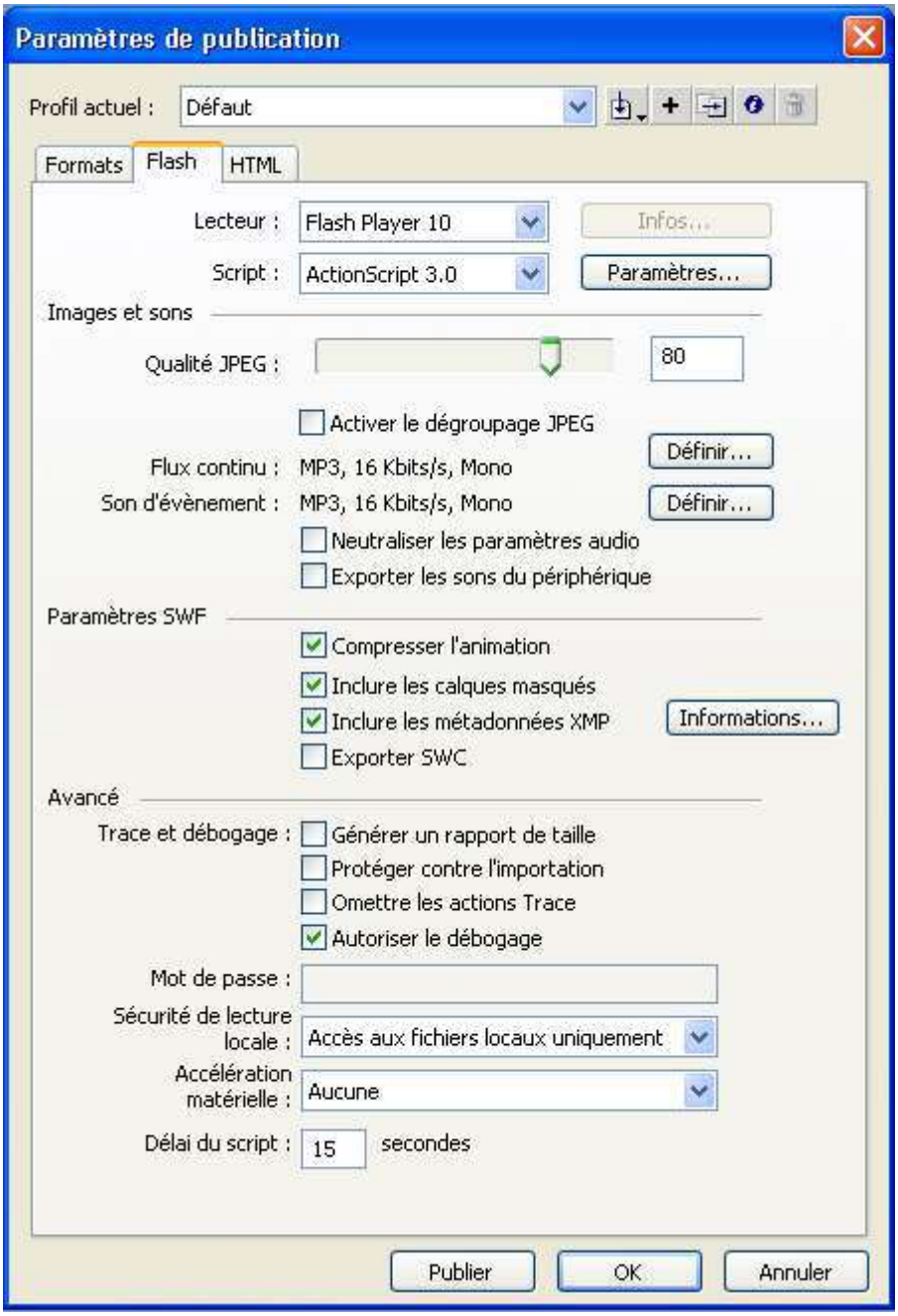
  - Sélectionner **Flash Player 10** comme lecteur.

  - Sélectionner **ActionScript 3.0** comme script.

  - Cocher **Autoriser le débogage** :

  Ce qui permet, en cas d'erreur, d'avoir un message plus explicité (endroit où se situe l'erreur).

  Nota : Cette option "Autoriser le débogage" sera à décocher à la fin de la conception, avant la publication finale.



2 Les variables

2.1 Différents types

string

texte
valeur par défaut : null

number

nombre à virgule flottante
de - $1.79e^{308}$ à $1.79e^{308}$
64 bits
valeur par défaut : NaN (not a number)

int

nombre entier positif ou négatif
de – 2 147 483 ^{648} à + 2 147 483 647

32 bits
valeur par défaut : 0

uint

nombre entier positif
de 0 à 4 294 967 295
32 bits
également utilisé pour les couleurs avec ou sans alpha :
(sans : 0xrrvvbb - avec : 0xaarrvvbb)
valeur par défaut : 0

boolean

booleen
2 valeurs possibles : true et false
valeur par défaut : false

*****

n'importe quel type
peut changer de type au cours du programme
valeur par défaut : undefined

2.2 Syntaxe

2.2.1 Déclaration

```
var nombre1:int = 10;
var texte:String = "bonjour";
var texte:String = "il a : dit \"bonjour\"";
var texte:String = 'bonjour';
const nombre2:int 10; //la valeur ne pourra pas être modifiée par une future affectation
```

2.2.2 Affectation

```
nombre1 = 20;
```

2.2.3 Conversion

```
var texte:String = "2010";
trace (Number(texte));
```

☛ affiche : **2010**

```
var texte:String = "1.26cm";
trace (parseFloat(texte));
```

☛ affiche : **1.26**

2.2.4 Nom dynamique de variable (variable de variable)

Pour créer ou utiliser un nom de variable dynamique :

(ex: txt_1, txt_2, …) utiliser la syntaxe : ['préfixe' + variable]

```
for ( var nb:int = 1; nb < 6 ; nb++ )
{
// remplissage des champs de texte txt_1, txt_2, txt_3, txt_4, txt_5
// remplissage de txt_1 avec Champ 1, txt_2 avec Champ 2, ...
    this['txt_' + nb].text = "Champ " + nb;
}
```

Champ 1
Champ 2
Champ 3
Champ 4
Champ 5

3 Les fonctions

3.1 Généralités

3.1.1 Principe

```
// Déclaration de la fonction
function maFonction (parametres)
{
code à exécuter
}

// Appel de la fonction
maFonction (parametres);
```

3.1.2 Recommandations

- Il est conseillé de faire commencer le nom des paramètres transmis par un "p" afin de mieux différentier les paramètres transmis des variables dans le code de la fonction (ex: pNombre1).
- Comme pour les variables, il est fortement recommandé de spécifier le type des paramètres transmis ainsi que celui de la valeur retournée.
  Ex: function addition (pNombre1:Number, pNombre2:Number):Number

3.2 Différents types

3.2.1 Sans retourner de valeurs

Il faut spécifié le type "void"

```
// Fonction addition
function addition (pNombre1:Number, pNombre2:Number):void
{
trace (pNombre1 + pNombre2);
}
// Appel de la fonction addition
addition(pNombre1,pNombre2);
```

3.2.2 Avec retour de valeurs

La transmission du résultat d'une fonction (retour de paramètre) s'effectue grâce à l'instruction return :

```
// Fonction addition
function addition (pNombre1:Number, pNombre2:Number):Number
{
return pNombre1 + pNombre2;
}

// Appel de la fonction addition et affectation du résultat à la variable "total"
var total:Number = addition (10, 100);
trace (total); // affichage du résultat
```

Remarques :

- Toutes les lignes de code de la fonction placées après la ligne contenant "return" ne seront pas exécutées, car l'exécution de la fonction se termine par l'instruction "return".
- Attention de bien respecter la concordance entre le format transmis et celui de la variable à laquelle on affecte le résultat :

```
function addition (pNombre1:Number, pNombre2:Number):Number
...
var total:Number = addition (10, 100);
```

3.2.3 Paramètres facultatifs

Si on veut affecter une valeur par défaut à un paramètre (en cas d'absence de valeur) :

```
function maFonction (pNombre:Number = 0):void
{
trace (pNombre);
}
maFonction ();          ☛ affiche : 0
maFonction (2);         ☛ affiche : 2
```

Remarques :

- Les paramètres facultatifs se placent toujours **après** les paramètres obligatoires.

```
function maFonction (pNombre1:Number , pNombre2:Number = 0):void
```

3.2.4 Paramètre reste

- Il existe un type de paramètres très particulier : le paramètre "reste", il s'écrit … suivi du nom du paramètre (ex: ...pRest). Il n'est pas nécessaire de typer ce paramètre car il accepte n'importe quel type de données, si on désire quand même le typer il faut lui attribuer le type * (...pRest:*).

- Il prendra toutes les valeurs qui restent après le renseignement des premiers paramètres :

```
function maFonction (pNombre1:Number , pNombre2:Number, ...pRest):void
{
trace (pRest);
}
maFonction (2, 3, "toto", new MovieClip(), 255);
```

☛ affiche : **toto, [object MovieClip], 255**

- Pour prendre toutes ces valeurs, il utilise le type Array qui permet de donner des listes de données indexées auxquelles on pourra accéder via un index entre crochets :

```
function maFonction (pNombre1:Number , pNombre2:Number, ...pRest):void
{
trace (pRest[0]);
}
maFonction (2, 3, "toto", new MovieClip(), 255);
```

☛ affiche : **toto**

- On pourra également utiliser le concept des boucles pour parcourir l'ensemble des paramètres.

4 Les variables globales et locales

Si l'utilité d'une variable (ou d'un objet) dépasse le périmètre d'une fonction, il faut alors la déclarer en "**globale**" sinon une déclaration en "**locale**" suffit.

4.1 Variables locales

- Pour déclarer une variable en locale, il faut effectuer sa **déclaration à l'intérieur** de la fonction.
- Cette variable ne sera accessible qu'à l'intérieur de la fonction dans laquelle elle a été déclarée.

```
function test():void
{
var maVariableLocale:String = "déclarée dans une fonction : je suis une variable locale";
trace (maVariableLocale);
}
```

4.2 Variables globales

- Pour déclarer une variable en globale, il faut effectuer sa **déclaration à l'extérieur** de la fonction.
- Cette variable sera alors accessible de n'importe où.

```
var maVariableGlobale:String = "déclarée avant les fonctions : je suis une variable globale";
function test():void
{
...
}
...
trace (maVariableGlobale);
```

5 Les objets

5.1 Organisation

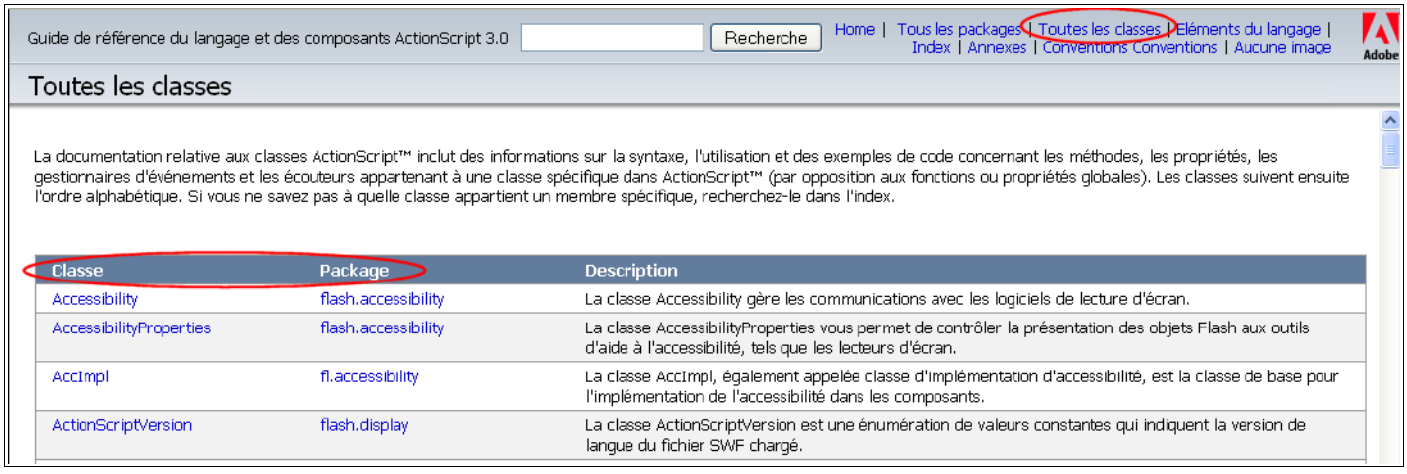
Chaque objet que l'on peut manipuler en ActionScript appartient à un certain type de données.

5.1.1 Symboles et occurrences

- Chaque type de données est crée par des classes.
- La relation qui existe entre une classe et un objet et un peu celle qui existe entre un moule et un objet qui est sorti de ce moule.
- Lorsque l'on a un clip sur la scène, le panneau de propriétés indique que c'est une occurrence de puis le nom du symbole en bibliothèque qui a permis de générer cette occurrence de clip. Si on modifie le clip (symbole), chaque occurrence est automatiquement modifiée.

5.1.2 Packages et classes

- Dans la documentation (par F1) le lien "Toutes les classes" présente l'ensemble des classes, c'est à dire l'ensemble des types de données que nous pouvons manipuler lorsque l'on programme dans Flash.
- Étant donné le nombre de classes possibles, nous devons adopter une certaine organisation pour classer ces classes. On utilise une structure sous forme de dossiers et sous-dossiers que l'on nomme "**package**".



- Les classes sont une sorte de moules qui permettent de créer des occurrences. On pourrait comparer les classes à des boites à outils qui agissent dans un certain contexte d'utilisation. Chaque objet (occurrence) que l'on peut manipuler est donc crée grâce à une classe.

5.2 Programmation sur une occurrence

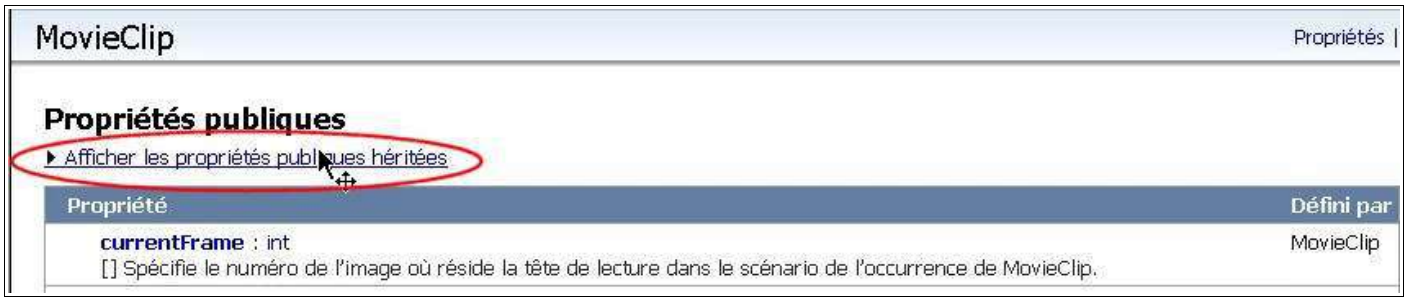
5.2.1 Identification d'une occurrence

- On crée une occurrence du symbole "monClip" que l'on appelle "**clip_1**".
- trace (clip_1); ☛ affiche : [object MovieClip]

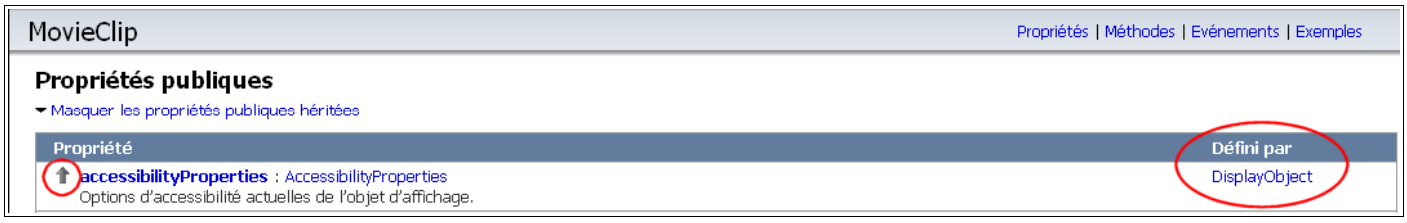
  object pour "occurrence" suivi du nom de la classe (MovieClip)
- trace (MovieClip); ☛ affiche : [class MovieClip]
- On sait donc que l'objet posé sur la scène est un MovieClip.

5.2.2 Propriétés et méthodes d'occurrence

- Dans la documentation en cliquant sur la classe "MovieClip", on peut savoir comment utiliser cet objet.
- Pour connaître toutes les fonctionnalités (notamment celles héritées), il faut cliquer sur "**Afficher les propriétés publiques héritées**" (faire de même pour les méthodes et les événements).



- Chaque fonctionnalité (propriété, méthode ,événement) héritée est alors indiquées par une flèche grise, le nom de la classe mère est également indiquée.



- On devient ainsi autonome en programmation ActionScript, grâce à la documentation.

5.2.3 Syntaxe pointée

- Pour accéder à la propriété d'un objet on utilise le plus souvent la syntaxe pointée (.) : clip_1.x

  Une propriété étant une variable, on se sert de l'opérateur d'affectation pour lui donner une valeur :

```
clip_1.x = 150;
clip_1.rotation = 90;
```

– On peut également affecter une méthode à cet objet.

Une méthode étant une fonction, on applique cette syntaxe :

```
clip_1.startDrag();
```

5.2.4 Syntaxe à crochets

– Dans certains cas particuliers, on peut utiliser la syntaxe à crochet qui consiste à mettre entre crochets une variable contenant le nom d'une propriété :

```
var propriete:String = "rotation";
clip_1[propriete] = 90;
```

– Cette syntaxe est souvent utilisée pour initialiser des objets avec des valeurs externes contenues par exemple dans un fichier XML (en fonction du couple : nom de propriété - valeur).

– Elle est utilisée également pour accéder aux éléments des objets de type Array ou Vector en indiquant leur index entre crochets.

5.2.5 Création d'une occurrence

Nous souhaitons créer sur la scène, par programmation ActionScript, un objet de type "Champ de texte" (comme on le ferait avec l'outil Texte).

– En appliquant la méthode indiquée au chapitre 5.3 (Identification d'une occurrence), nous déterminons que l'objet à créer est du type "TextField".

– Il va falloir déjà créer une **variable** qui va mémoriser l'occurrence créée. Le **type de données** est le nom de la classe qui permet de créer cet objet (TextField).

– Pour créer une occurrence, nous allons utiliser le mot clé "**new**" suivi du **nom de la classe** puis suivi de **()**

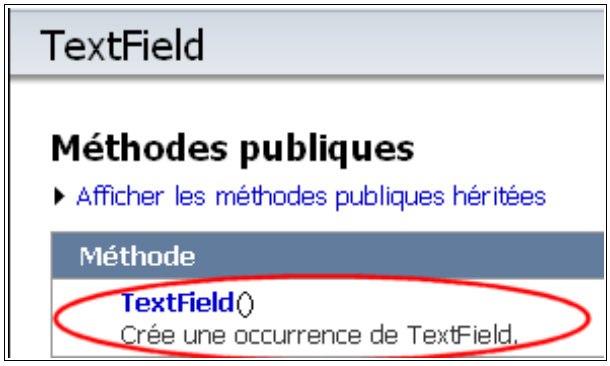
```
var champDeTexte:TextField = new TextField ();
```

– Cette syntaxe "**TextField ()**" nous rappelle la syntaxe d'exécution d'une fonction.

Cette fonction de même nom que la classe s'appelle la fonction **constructeur**.

Elle sert à créer une occurrence de la classe comme nous indique la documentation : La fonction constructeur TextField crée une occurrence de TextField.

Ça sera le cas pour chaque classe dont on peut créer des occurrences.

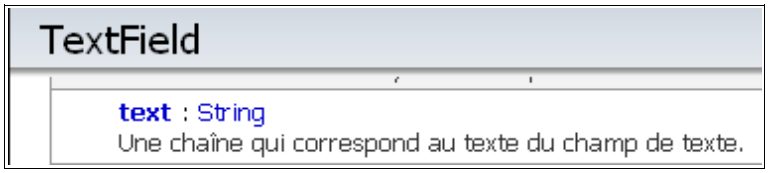


5.2.6 Valeurs d'une occurrence

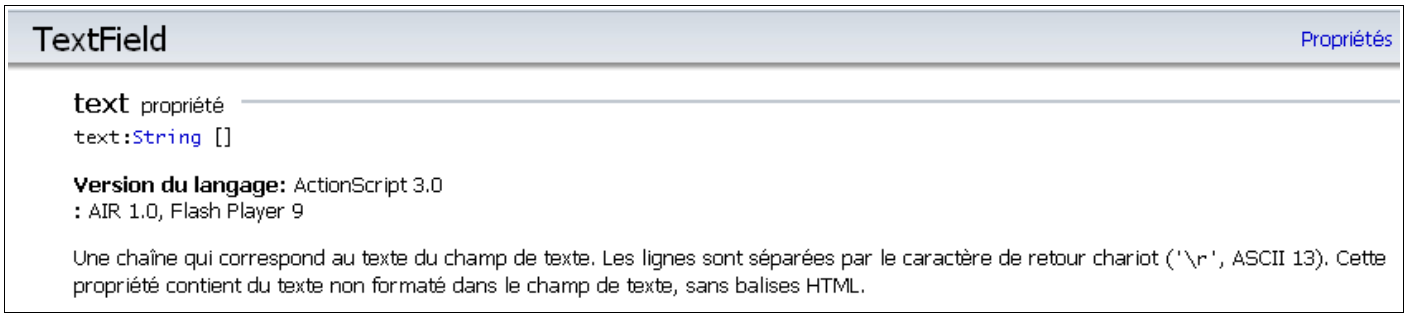
Après avoir crée ce champ de texte, nous voulons écrire du texte (Bonjour) à l'intérieur (toujours en code).

– Nous revenons sur la documentation de la classe TextField et nous recherchons une propriété ou une fonction nous permettant d'écrire dans le champ de texte.

– Nous trouvons la propriété "**text**" de type "String" :



– On cliquant sur celle-ci, la documentation nous apporte toutes les précisions nécessaires (notamment qu'elle correspond au texte du champ) :



– Nous pouvons alors appliquer à l'occurrence "**champDeText**" la propriété "**text**" et lui affecter la **valeur** voulue en respectant le type de données (ici String) :

```
champDeText . text = "Bonjour";
```

– Pour permettre l'affichage sur la scène, on doit utiliser la fonction "**addChield**" en lui transmettant le champ à afficher :

```
addChield (champDeTexte);
```

– Soit au final pour la création, la valeur et l'affichage de l'occurrence :

```
var champDeTexte:TextField = new TextField (); // Création de l'occurrence
champDeTexte.text = "Bonjour"; // Texte à afficher
addChild (champDeTexte); // Affichage sur la scène
```

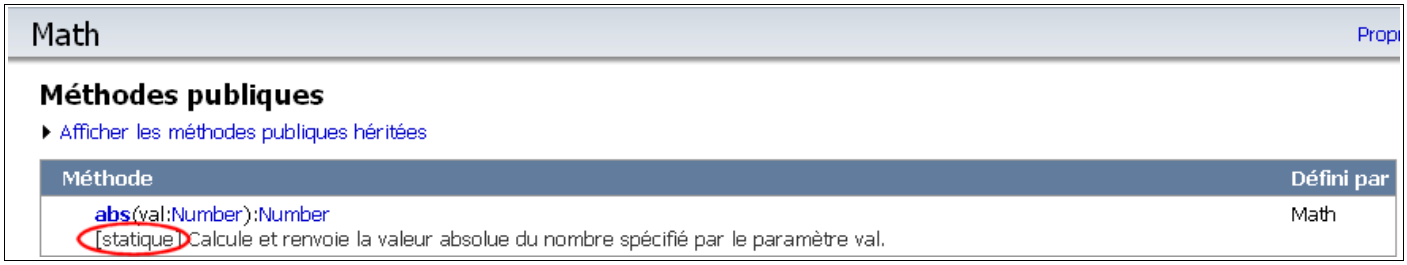
5.3 Programmation sur une classe

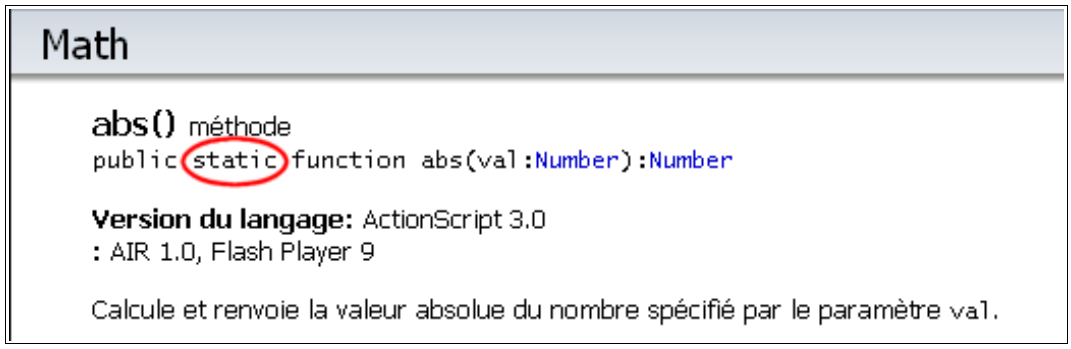
5.3.1 Classe statique

Nous venons de voir que l'on utilise les propriétés ou les méthodes sur les occurrences, mais il y a certains cas où ces propriétés ou ces méthodes s'utilisent directement sur la classe et non plus sur une occurrence.

Là encore la documentation va nous être très utile car elle nous indique pour chaque propriété ou méthode si elle s'applique a une occurrence ou sur la classe directement.

- Si elle s'applique à la classe, elle est identifiée par le mot clé : "**statique**" (ou "**static**") :





Dans ce cas, on a pas besoin de créer d'occurrence :

5.3.2 Classe d'énumération

C'est une classe qui énumère un ensemble de valeurs.

- Par exemple : sur la classe "**Stage**" (qui permet d'agir de manière globale sur le fonctionnement du Flash Player) nous avons une propriété "**Align**" (qui permet de modifier le mode d'alignement du Flash player lorsqu'il est redimensionné) qui attend un certain nombres de valeurs du type String. Les valeurs que nous lui donnerons doivent être reconnues par la classe "Stage". Pour nous faciliter la tâche, il existe une classe qui se nomme "**StageAlign**" qui contient les valeurs possibles pour cette propriété.

Stage

Propriétés | Méth

align propriété

```
align:String []
```

Version du langage: ActionScript 3.0
: AIR 1.0, Flash Player 9

Une valeur de la classe StageAlign qui détermine l'alignement de la scène dans Flash Player ou dans le navigateur. Les valeurs suivantes sont prises en charge :

Valeur	Alignement vertical	Horizontale
StageAlign.TOP	Haut	Centre
StageAlign.BOTTOM	Aligner les bords inférieurs	Centre
StageAlign.LEFT	Centre	Gauche
StageAlign.RIGHT	Centre	Aligner les bords droits
StageAlign.TOP_LEFT	Haut	Gauche
StageAlign.TOP_RIGHT	Haut	Aligner les bords droits
StageAlign.BOTTOM_LEFT	Aligner les bords inférieurs	Gauche
StageAlign.BOTTOM_RIGHT	Aligner les bords inférieurs	Aligner les bords droits

- Ces valeurs sont stockées sous forme de constantes statiques (donc applicables directement sur la classe).

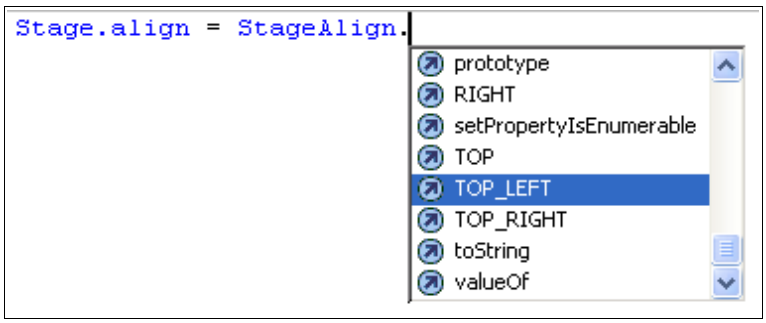
StageAlign

Constantes publiques

Constante	Défini par
BOTTOM : String = "B" [statique] Spécifie que la scène est alignée sur le bas.	StageAlign
BOTTOM_LEFT : String = "BL" [statique] Spécifie que la scène est alignée sur le coin inférieur gauche.	StageAlign
BOTTOM_RIGHT : String = "BR" [statique] Spécifie que la scène est alignée sur le coin inférieur droit.	StageAlign
LEFT : String = "L" [statique] Spécifie que la scène est alignée sur la gauche.	StageAlign
RIGHT : String = "R" [statique] Spécifie que la scène est alignée sur la droite.	StageAlign
TOP : String = "T" [statique] Spécifie que la scène est alignée sur le haut.	StageAlign
TOP_LEFT : String = "TL" [statique] Spécifie que la scène est alignée sur le coin supérieur gauche.	StageAlign
TOP_RIGHT : String = "TR" [statique] Spécifie que la scène est alignée sur le coin supérieur droit.	StageAlign

Cette classe n'a comme seul rôle que de stocker (d'énumérer) ces valeurs.

- Si je souhaite l'utiliser (par exemple : parce que je ne souhaite pas, lorsque je déplace le Flash Player, que les objets se déplacent je souhaite qu'ils gardent leur position en haut à gauche), je vais pouvoir utiliser "l'aide à la saisie" pour choisir parmi les constantes (écrites en majuscule) celle qui correspond à mon choix :




```
Stage.align = StageAlign.TOP_LEFT;
```

5.3.3 Classe dynamique

- Il existe des classes qui sont dites "**dynamiques**" identifiées dans la documentation par le mot-clé "**dynamic**".

- Si une classe est dynamique cela signifie que j'ai le droit de créer, sur une occurrence de cette classe, des propriétés qui n'existent pas dans la classe elle-même.

MovieClip

Package	flash.display
Classe	public dynamic class MovieClip
Héritage	MovieClip → Sprite → DisplayObjectContainer → InteractiveObject → DisplayObject → EventDispatcher → Object
Sous-classes	LivePreviewParent

```
var monClip:MovieClip = new MovieClip (); // Création de l'occurrence
monClip.toto = 2; // Affectation de la valeur 2 à la propriété "toto"
trace (monClip.toto); // Affichage de la valeur de la propriété "toto"
```

- Il existe quand même un inconvénient à ce type de classe, c'est que le compilateur ne peut pas gérer les erreurs du fait qu'il est possible de créer ce que l'on veut.

  Si l'on utilise quelque chose (fonction) qui n'existe pas, l'erreur ne sera donc pas vue à la compilation mais uniquement à l'exécution (par le Flash Player).

5.3.4 Les classes Array et Vector

Ces classes nous permettent de créer et de gérer des listes d'objets.

- Lorsque nous créons une occurrence de la classe **Array**, nous pouvons transmettre deux types de paramètres :
  - soit un paramètre numérique qui va créer une liste de x éléments vides .
  - soit tout autres paramètres (par exemple une chaine de caractères) qui créera alors une liste contenant ces différents éléments.

```
var liste1:Array = new Array (3);
var liste2:Array = new Array ("tata", "titi", "toto");
```

- La classe **vector** a le même rôle que la classe Array. Cependant les éléments auront tous le même type, celui-ci sera spécifié lors de la déclaration et vérifié lorsque l'on ajoutera un élément.
- La classe **vector** nécessite Flash Player 10.
- La syntaxe de la classe **vector** est un peu particulière :

```
// Création de l'occurence
var liste1:Vector.<MovieClip> = new Vector.<MovieClip>();

// Ajout d'un élément
liste1.push (new MovieClip());
```

- Avec la classe **vector**, nous pouvons fixer et limiter le nombre d'éléments :

```
// Création de l'occurence
var liste1:Vector.<MovieClip> = new Vector.<MovieClip>(10, true);
```

Dans cet exemple, la liste est créée avec 10 éléments et ne dépassera jamais les 10 éléments.

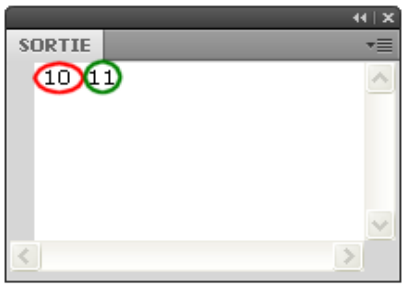
5.4 Type primitif et type complexe

5.4.1 Définition

- Les types **Number**, **int**, **uint**, **String** et **Boolean** sont des types **primitifs** (ou simples).
- Tous les autres sont des types **complexes**.

5.4.2 Comportement

```
var monNombre:int = 10; // Déclaration et affectation de la variable monNombre
var monClip:MovieClip = new MovieClip (); // Création de l'occurrence monClip
monClip.x = 10  // Affectation de la propriété x de monClip

function test ( pValeur1:int, pValeur2:MovieClip ):void
{
    pValeur1 += 1;      // Incrémentation de pValeur1
    pValeur2.x += 1;    // Incrémentation de x de pValeur2
}

test ( monNombre, monClip ); // Exécution de la fonction test

trace ( monNombre, monClip.x ); // Affichage de monNombre et monClip.x
```



- On s'aperçoit que :
  - monNombre (type primitif) garde sa valeur d'origine.
  - MonClip.x (type complexe) a été modifié par la fonction.

5.4.3 Principe

Lorsque l'on donne à un paramètre d'une fonction le type :

- **primitif** : l'objet qui est reçu par la fonction n'est qu'une **copie** de cet objet.

- **complexe** : l'objet qui est reçu par la fonction n'est pas une copie mais est la **référence** vers l'objet d'origine. L'objet d'origine sera donc modifié par la fonction !

6 Les conditions

- Chaque condition que l'on va écrire va retourner un Booléen (true ou false).

6.1 Les opérateurs

- Égalité : **==**
- Inégalité : **!=**
- Supériorité : **>**
- Supériorité ou égalité : **>=**
- Infériorité : **<**
- Infériorité ou égalité : **<=**
- Est : **is** ex: if (maVariable **is** String)

6.2 Les opérateurs logiques

- ET logique : **&&** ex: if (nombre > 10 **&&** nombre < 100)
- OU logique : **||** ex: if (reponse == solution1 **||** reponse == solution2)
- PAS (NOT) : **!**

```
if ( clip1.hitTestObject (clip2) )
{
trace ( "collision" );
}
```

```
if ( ! clip1.hitTestObject (clip2) )
{
trace ( "pas de collision" );
}
```

6.3 L'instruction if

```
if ( condition 1 )
{
    code exécuté si condition 1 est vraie
}
else if ( condition 2 )
{
    sinon code exécuté si condition 2 est vraie
}
else
{
    sinon code exécuté
}
```

Remarque : il est conseillé de toujours finir par l'instruction "**else**" pour être sur que toutes les conditions ont été testées.

```
if ( champTexte.text == "" )
{
    messageUtilisateur.text = "Merci de répondre";
}
else if ( champTexte.text == bonneReponse )
{
    messageUtilisateur.text = "Bravo !";
}
else
{
    messageUtilisateur.text = "Perdu !";
}
```

6.4 L'instruction switch

```
switch ( Capabilities.language )
{
    case "fr" :
        trace ( "La langue est le français" );
    break;

    case "en" :
        trace ( "La langue est l'anglais" );
    break;

    default :
        trace ( "Cette langue n'est pas prise en compte" );
}
```

- L'instruction "**default**" équivaut à l'instruction "else" (sinon)

```
switch ( Capabilities.language )
{
    case "fr" :
    case "en" :
        trace ( "La langue est prise en charge" );
    break;

    default :
        trace ( "Cette langue n'est pas prise en charge" );
}
```

- Nous avons la possibilité de prolonger une condition sur une autre en supprimant une instruction "break" :

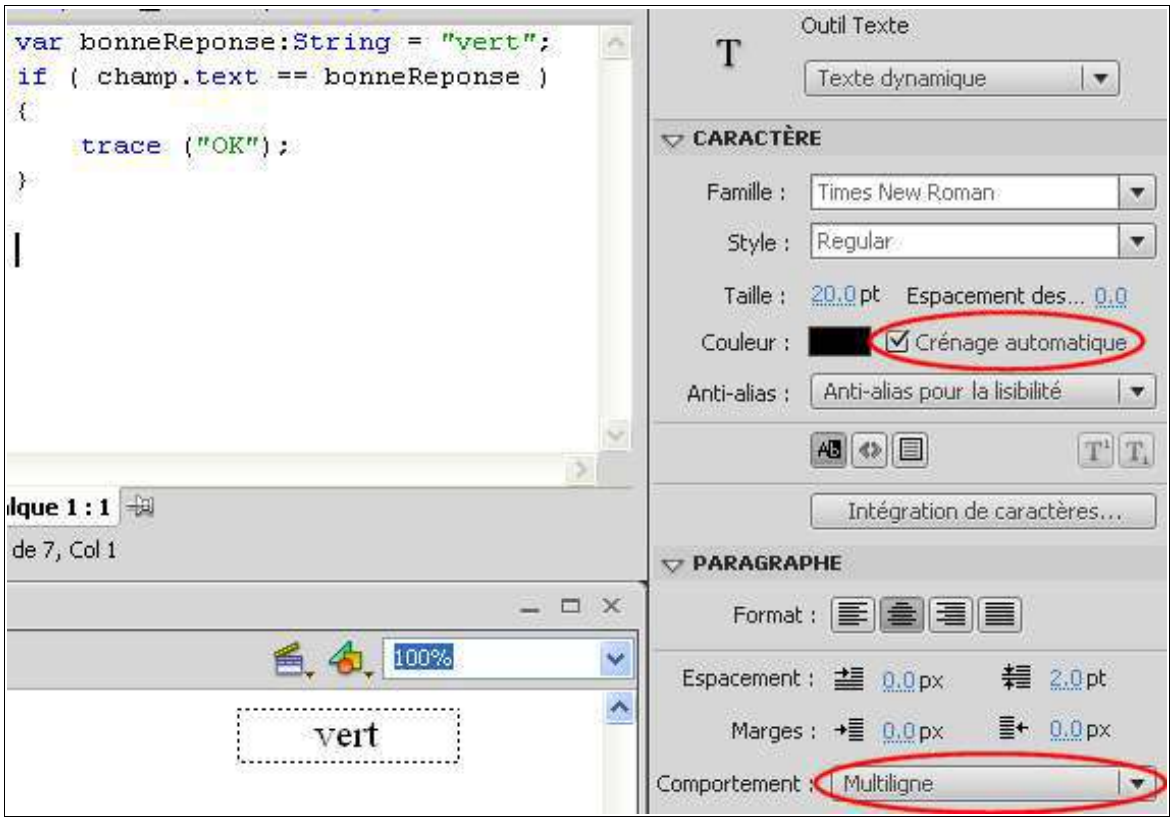
6.5 L'opérateur ternaire

- Cet opérateur permet de glisser une condition au sein d'une instruction.

```
var v = ( condition ) ? code exécuté si vrai : code exécuté si faux;
```

```
var messageUtilisateur:String = ( champTexte ==  bonneReponse ) ? "Bravo !" : "Perdu !";
```

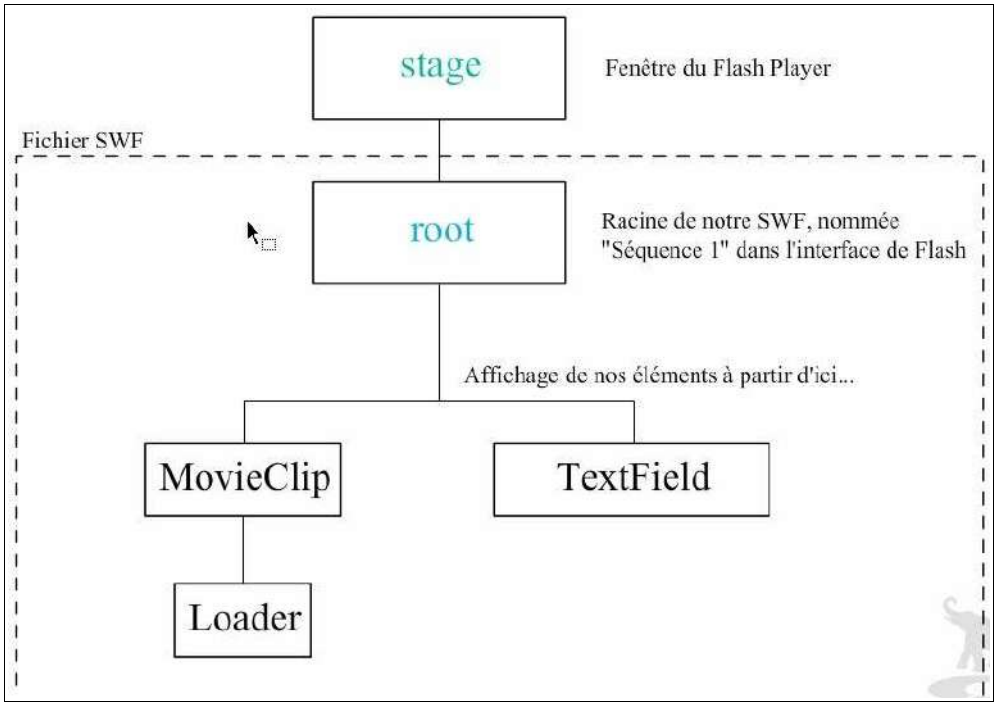
6.6 Problématique sur les champs de texte



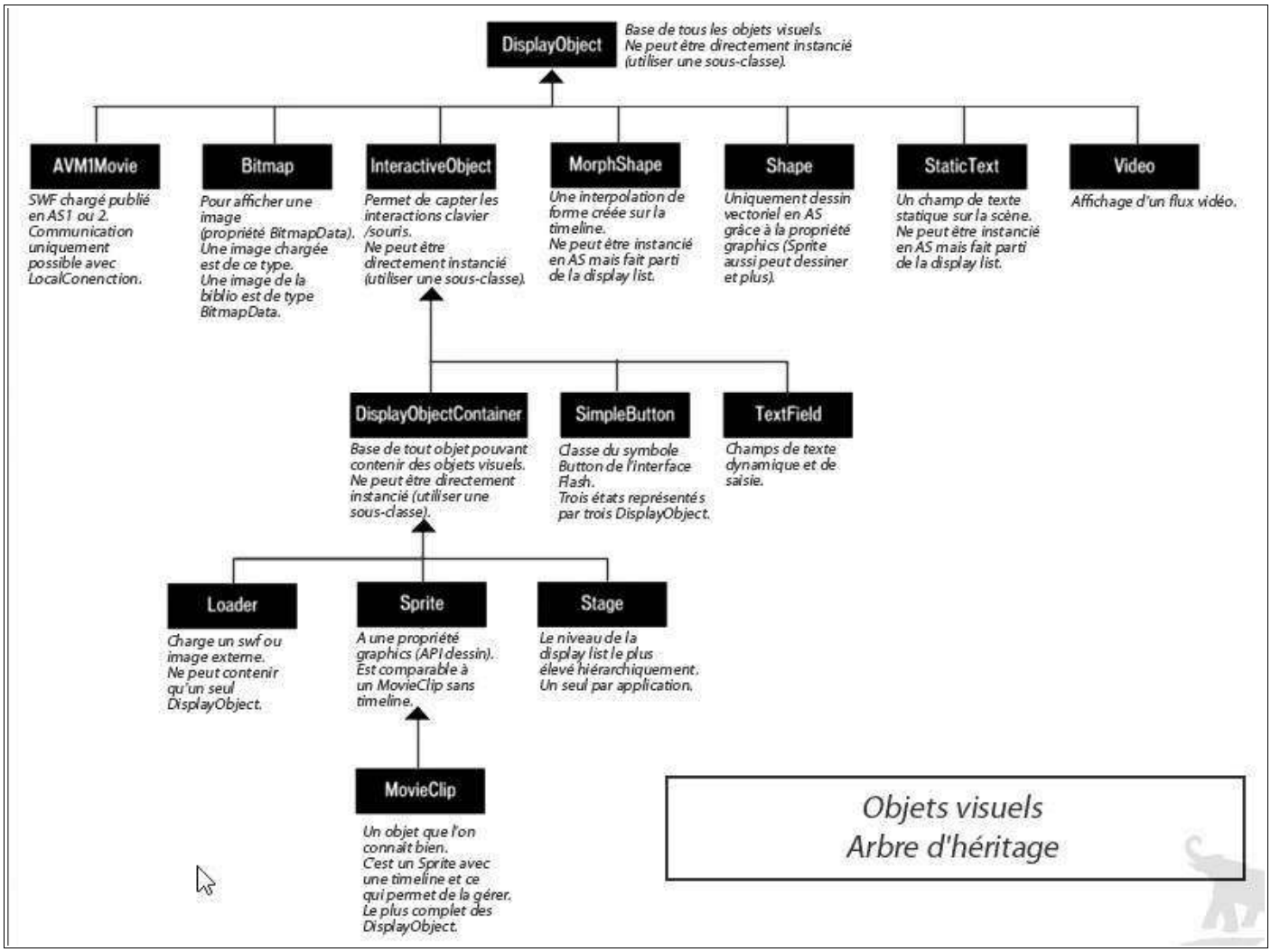
- Il existe une problématique sur les champs texte liée aux **paramètres par défaut,** notamment à la combinaison de "**crénage automatique**" et de "**comportement multiligne**" :
- Lorsque nous activons ces 2 options, Flash rajoute dans le texte du code Html qui rend notre condition inefficace.
- Pour éviter cette contrainte, il suffit de décocher le "**crénage automatique**" et la condition devient efficace. Notons au passage que n'ayant pas besoin d'un champ multiligne, nous aurions aussi choisir comportement "Une seule ligne".

7 L'affichage (1ère partie)

7.1 Arborescence d'affichage

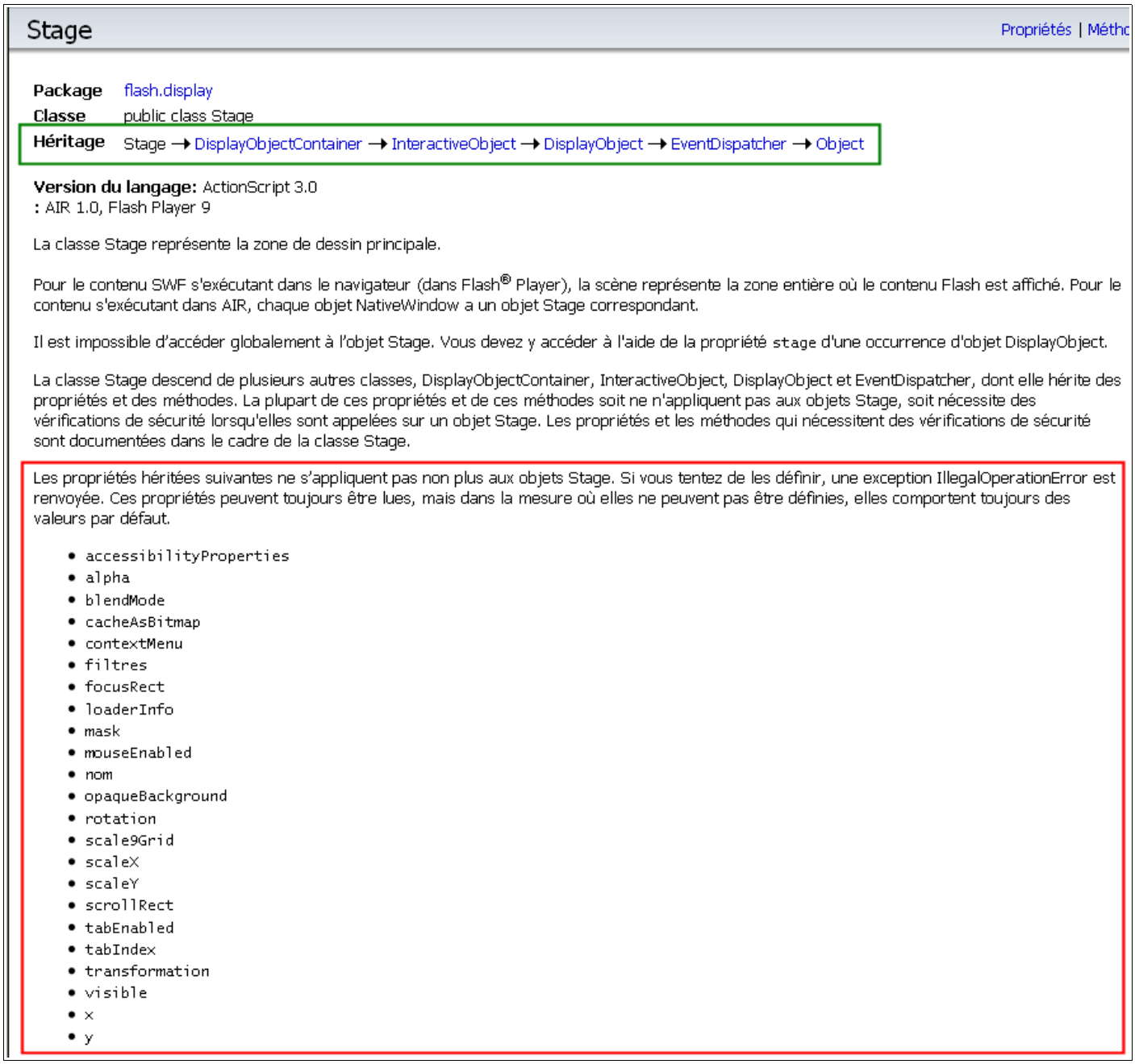


7.2 Héritage



- Les différentes classes héritées sont également mentionnées dans la documentation.
- Dans la majorité des cas, une propriété ou une méthode héritée est utilisable sur la classe qui en hérite.

  Il existe des exceptions, celle-ci sont indiquées dans la documentation (ex: classe Stage) :



Stage

Propriétés | Méthc

Package flash.display
Classe public class Stage
Héritage Stage → DisplayObjectContainer → InteractiveObject → DisplayObject → EventDispatcher → Object

Version du langage: ActionScript 3.0
: AIR 1.0, Flash Player 9

La classe Stage représente la zone de dessin principale.

Pour le contenu SWF s'exécutant dans le navigateur (dans Flash® Player), la scène représente la zone entière où le contenu Flash est affiché. Pour le contenu s'exécutant dans AIR, chaque objet NativeWindow a un objet Stage correspondant.

Il est impossible d'accéder globalement à l'objet Stage. Vous devez y accéder à l'aide de la propriété `stage` d'une occurrence d'objet DisplayObject.

La classe Stage descend de plusieurs autres classes, DisplayObjectContainer, InteractiveObject, DisplayObject et EventDispatcher, dont elle hérite des propriétés et des méthodes. La plupart de ces propriétés et de ces méthodes soit ne n'appliquent pas aux objets Stage, soit nécessite des vérifications de sécurité lorsqu'elles sont appelées sur un objet Stage. Les propriétés et les méthodes qui nécessitent des vérifications de sécurité sont documentées dans le cadre de la classe Stage.

Les propriétés héritées suivantes ne s'appliquent pas non plus aux objets Stage. Si vous tentez de les définir, une exception IllegalOperationError est renvoyée. Ces propriétés peuvent toujours être lues, mais dans la mesure où elles ne peuvent pas être définies, elles comportent toujours des valeurs par défaut.

- `accessibilityProperties`
- `alpha`
- `blendMode`
- `cacheAsBitmap`
- `contextMenu`
- `filtres`
- `focusRect`
- `loaderInfo`
- `mask`
- `mouseEnabled`
- `nom`
- `opaqueBackground`
- `rotation`
- `scale9Grid`
- `scaleX`
- `scaleY`
- `scrollRect`
- `tabEnabled`
- `tabIndex`
- `transformation`
- `visible`
- `x`
- `y`

7.3 Affichage d'un objet

- Pour afficher un objet en ActionScript, il faut toujours avoir en tête les schémas relatifs à l'arborescence d'affichage et à l'arbre d'héritage.
- De plus, il est important de se rappeler que l'objet racine de mon swf : "**root**" est un **MovieClip**.
- en ActionScript, pour désigner l'objet sur lequel on est en train d'écrire du code ("root"), on utilise le mot clé "**this**".
- On peut demander l'affichage d'un objet, soit directement sur la scène, soit dans un conteneur :

```actionscript
var champTexte:TextField = new TextField();
champTexte.text = "Mon texte";

// Affichage du champ de texte sur la scène principale
this.addChild (champTexte);
```

7.4 Exporter un élément de la bibliothèque pour ActionScript

7.4.1 Importer une image dans la bibliothèque

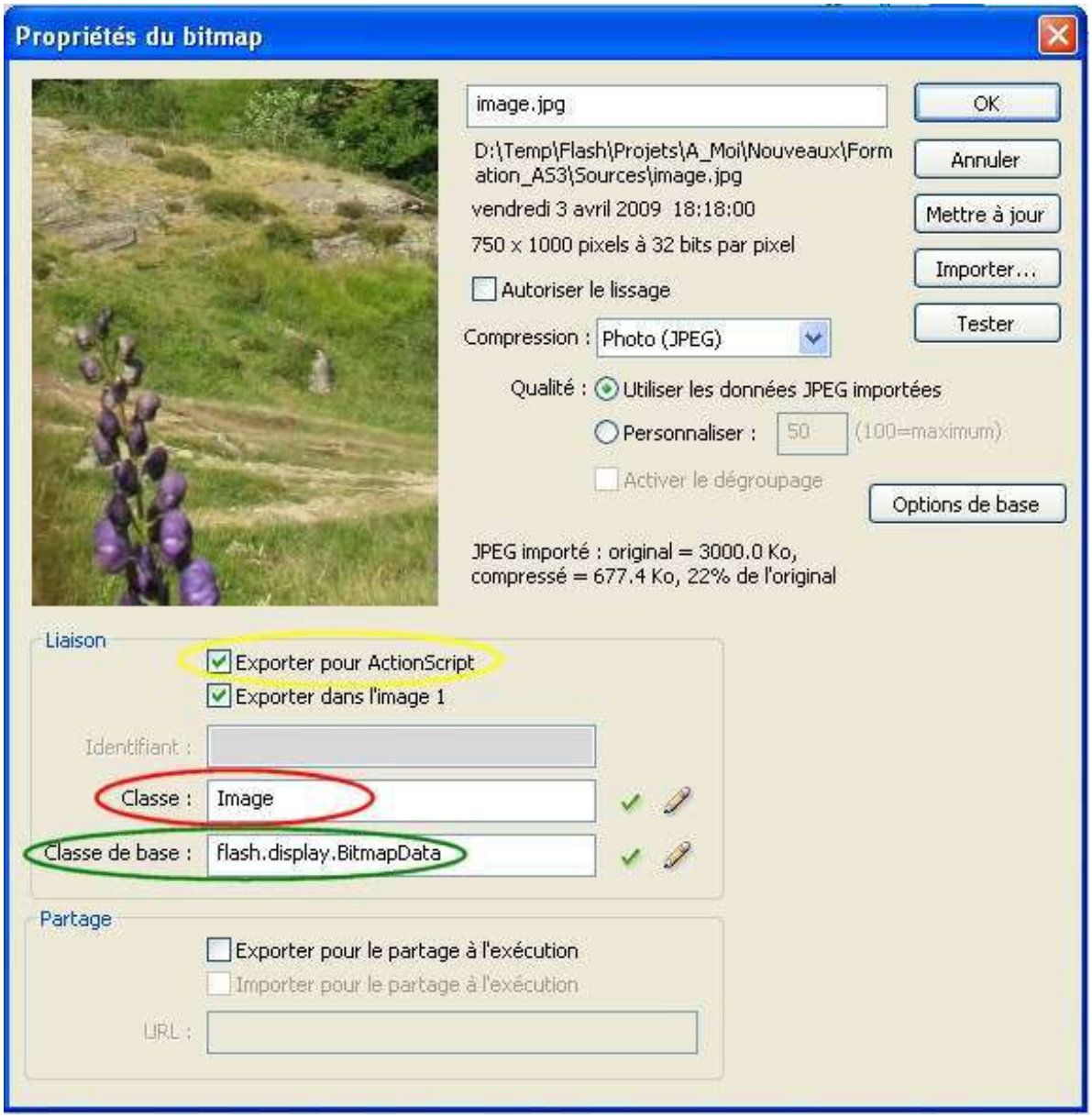
```actionscript
var champTexte:TextField = new TextField();
champTexte.text = "Mon texte";

// Affichage du champ de texte dans un conteneur
var conteneur:Sprite = new Sprite();
conteneur.addChild (champTexte);
this.addChild (conteneur);
// remarque :
// this étant implicite, on aurait pu écrire :
// addChild (conteneur);
```

- Pour importer une image dans la bibliothèque, nous pouvons soit :
  - Directement glisser le fichier de l'explorateur vers la bibliothèque.
  - Faire "Fichier / Importer / Importer dans la bibliothèque...".

7.4.2 Exporter une image pour ActionScript

- Pour pouvoir manipuler cette image par le code d'ActionScript, nous allons devoir modifier les paramètres de celle-ci :
  - Dans la bibliothèque, faire un double-clic sur l'image (ou un clic droit puis propriété).
  - Il faudra alors cocher "**Exporter pour ActionScript**",
  - et donner un **nom de classe** pour identifier la classe qui sera créée pour cet objet.

    Pour respecter la convention de nommage des classes, je fais commencer le nom de la classe par une majuscule.
  - On voit également que la classe de base est la classe **BitmapData**. Cette classe ne fait pas partie des classes d'affichage que nous avons vu dans l'arbre d'héritage il faudra donc passer par une classe qui permet l'affichage de ces données en l'occurrence la classe Bitmap.



7.4.3 Affichage d'une image avec ActionScript

- Comme indiqué dans la documentation relative à la méthode "constructeur", la classe "BitmapData" possède 2 paramètres obligatoires (**width et height**)
- Il va donc falloir renseigner ces paramètres.

  Dans le cas d'une image en bibliothèque, quelque soit les dimensions que l'on indique en paramètres, l'image sera affichée dans ses dimensions d'origine.

  On peut donc indiquer ce que l'on veut comme paramètres car ceux-ci ne seront pas pris en compte. Il est recommandé, bien qu'en théorie ça ne soit pas possible, d'indiquer **(0, 0)** afin de nous indiquer (en voyant le code) que l'image vient de la bibliothèque.

BitmapData() Constructeur

```
public function BitmapData(width:int, height:int, transparent:Boolean = true, fillColor:uint = 0xFFFFFFFF)
```

Version du langage: ActionScript 3.0
: AIR 1.0, Flash Player 9

Crée un objet BitmapData à la largeur et la hauteur spécifiées. Si vous spécifiez une valeur pour le paramètre `fillColor`, chaque pixel du bitmap est défini sur cette couleur.

Par défaut, le bitmap créée est transparente, sauf si vous transmettez la valeur `false` au paramètre `transparent`. Une fois le bitmap opaque créée, vous ne pouvez pas la transformer en bitmap transparente. Chaque pixel d'une image bitmap opaque utilise uniquement 24 bits d'informations du canal de couleur. Si vous réglez le bitmap sur transparent, chaque pixel utilise 32 bits d'informations de canal de couleur, y compris un canal de transparence alpha.

Les largeur et hauteur maximales d'un objet BitmapData sont de 2 880 pixels. Si vous spécifiez une valeur de largeur ou de hauteur supérieure à 2880, la nouvelle occurrence n'est pas créée.

Paramètres

`width``:int` — Largeur de l'image bitmap en pixels.

`height``:int` — La hauteur de l'image bitmap en pixels.

`transparent``:Boolean` (default = `true`) — Spécifie si l'image bitmap prend en charge la transparence par pixel. La valeur par défaut est `true` (transparent). Pour créer une image bitmap entièrement transparente, réglez la valeur du paramètre `transparent` sur `true` et celle du paramètre `fillColor` sur 0x00000000 (ou sur 0). Le réglage de la propriété `transparent` sur `false` peut entraîner une légère amélioration des performances de rendu.

`fillColor``:uint` (default = 0xFFFFFFFF) — Valeur de couleur ARVB 32 bits utilisée pour remplir la zone de l'image bitmap. La valeur par défaut est 0xFFFFFFFF (blanc uni).

Valeur émise

`ArgumentError` — La largeur et/ou la hauteur ne sont pas valides (inférieures ou égales à zéro ou supérieures à 2 880).

```as3
// Création d'une occurrence de mon image
var image1:BitmapData = new Image (0,0);

// Pour pouvoir afficher uneimage sur la scène,
// on doit passer par la classe Bitmap
// et lui indiquer quelle est l'occurrence
// qui contient les données à afficher
var bitmap1:Bitmap = new Bitmap (image1);
                                 new Bitmap([donnéesBitmap:flash.display:BitmapData=null,

// Affichage de l'image
addChild (bitmap1);
```

7.4.4 Affichage d'un clip de la bibliothèque

- Comme pour une image, il va falloir :
  - Exporter le clip pour ActionScript en modifiant ses propriétés :
    - Cocher "**Exporter pour ActionScript**",
    - Donner un **nom de classe** pour identifier la classe qui sera créée pour cet objet.

      Faire commencer le nom de la classe par une majuscule pour respecter la convention de nommage des classes.
    - On voit également que la classe de base est la classe **MovieClip**. Cette classe fait pas partie des classes d'affichage que nous avons vu dans l'arbre d'héritage.


```
// Création d'une occurrence de mon clip
var clip1:MovieClip = new Clip1 ();

// Positionnement du clip sur la scène
clip1.x = 100;
clip1.y = 100;

// Affichage du clip
addChild (clip1);
```

7.5 Manipulation de l'affichage

Nous allons détailler le fonctionnement de la classe "**DisplayObjectContainer**" qui nous permet dans un objet qui hérite de cette même classe d'afficher autant d'enfants que l'on souhaite, de supprimer certains enfants et de réorganiser l'ordre (les index) d'affichage.

- Nous connaissons déjà la méthode "**addChild**".
- Nous avons également une méthode "**addChildAt**" qui nous permet de spécifier un **index** d'affichage. L'index joue le rôle de calque.

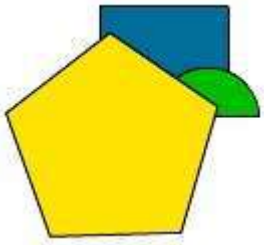
Méthodes publiques

▸ Afficher les méthodes publiques héritées

Méthode
DisplayObjectContainer() L'appel du constructeur DisplayObjectContainer() renvoie une exception ArgumentError.
addChild(child:DisplayObject):DisplayObject Ajoute une occurrence enfant de DisplayObject à cette occurrence de DisplayObjectContainer.
addChildAt(child:DisplayObject, index:int):DisplayObject Ajoute une occurrence enfant de DisplayObject à cette occurrence de DisplayObjectContainer.

Si nous comparons la gestion des index au calque de Flash :

- L'index 0 correspond à l'arrière plan.
- Plus l'index est élevé, plus l'objet est proche du premier plan
- L'index le plus élevé, correspond au premier plan. L'index du premier plan dépendra du nombre d'objets affichés.



Dans le cas de ces 3 objets, le DisplayObjectContainer placera dans le Flash Payer :

Le bleu : clip1 (qui est en arrière plan) sur l'index 0.

Le vert : clip2 (qui est entre les deux) sur l'index 1.

Le jaune : clip3 (qui est au premier plan) sur l'index 2.

- Un index ne peut contenir qu'un seul objet, contrairement à Flash qui nous permet, sur un même claque, de placer plusieurs objets.

- Il existe une propriété ("**numChildren**") qui permet de connaître le nombre d'enfants que possède le conteneur.

  Dans notre exemple le conteneur racine a 3 enfants, donc la propriété "**numChildren**" vaudra 3.

- Il existe une méthode ("**getChildIndex**") qui nous permet de connaître l'index d'affichage pour un objet donné:

  trace (getChildIndex (clip2)); ☛ affiche : 1

- Il existe une méthode qui permet de supprimer l'affichage d'un objet (seulement l'affichage, pas l'objet lui-même) :

  removeChild (clip2); ☛ affiche

- On pourra, plus tard dans le programme, ré-afficher clip2 par :

  addChild (clip2);

  Attention lorsque l'on fait un **addChild** dans un programme, l'objet sera toujours placé au premier plan.

- Nous pouvons modifier l'index d'un objet par la méthode "**setChildIndex**".
- Nous pouvons inverser l'index de 2 objets par les méthodes "**swapChildren**" et "**swapChildrenAt**". La différence entre ces deux méthodes est sur les paramètres.

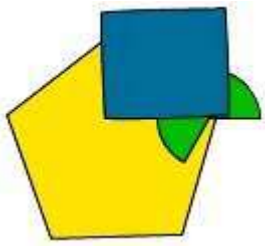
setChildIndex(child:DisplayObject, index:int):void
Modifie la position d'un enfant existant dans le conteneur d'objet d'affichage.

swapChildren(child1:DisplayObject, child2:DisplayObject):void
Intervertit l'ordre z (ordre d'empilement du premier plan vers l'arrière-plan) des deux objets enfants spécifiés.

swapChildrenAt(index1:int, index2:int):void
Intervertit l'ordre z (ordre d'empilement du premier plan vers l'arrière-plan) des objets enfants aux deux positions d'index spécifiées dans la liste d'enfants.

- swapChildren (clip1, clip3);
- swapChildrenAt (0, 2);

☛ afficheront



Nous utiliserons l'une ou l'autre de ces méthodes selon si l'on connait le nom de l'objet ou l'index d'affichage.

- Attention lorsque l'on utilise des méthodes qui possède "**index**" en paramètre, de bien donner une valeur d'index inférieure à "**numChidren**" !

8 Gestion événementielle

8.1 Ajout d'un écouteur d'événement

8.1.1 Syntaxe générique

Pour ajouter un écouteur d'événement, nous allons utiliser la méthode "**addEventListener**" de la classe "**EventDispatcher**" :

EventDispatcher — Propriétés | Méthodes | Evénements

Méthodes publiques

Afficher les méthodes publiques héritées

Méthode	Défini par
EventDispatcher(target:IEventDispatcher = null) Regroupe une occurrence de la classe EventDispatcher.	EventDispatcher
addEventListener(type:String, listener:Function, useCapture:Boolean = false, priority:int = 0, useWeakReference:Boolean = false):void Enregistre un objet écouteur d'événement auprès d'un objet EventDispatcher afin que l'écouteur soit averti d'un événement.	EventDispatcher

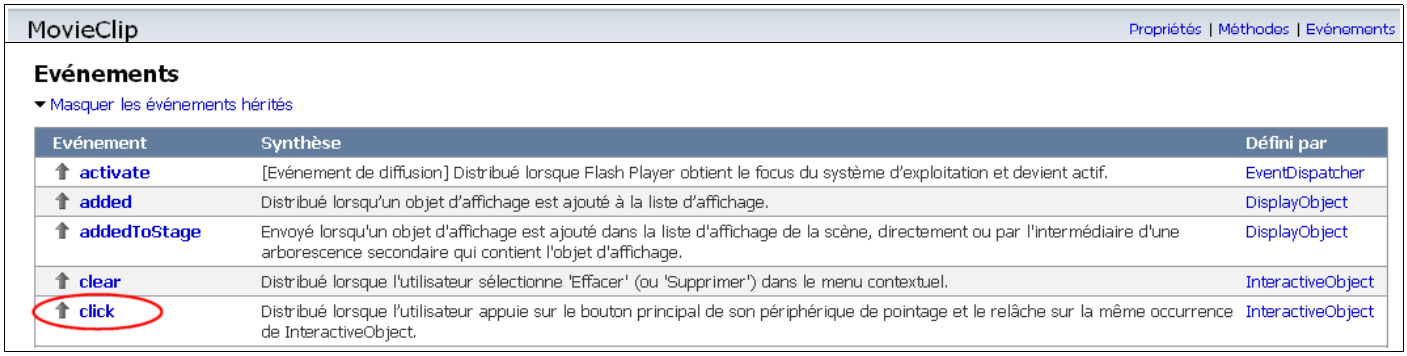
```
// Notre fonction écouteur
function fonctionEcouteur (parametre : ClasseEvenement) : void
{

}

// Ajout de l'écouteur d'événement
objetDistributeur.addEventListener ( identifiantEvenement, fonctionEcouteur );
```

8.1.2 Syntaxe appropriée

Grâce à la documentation, nous allons pouvoir définir tous les paramètres appropriés à l'objet sur lequel nous créons un écouteur.

Exemple : pour gérer un **clic** sur le clip "**clip1**" (qui est de type "**MovieClip**").

La documentation relative à l'événement "**click**" de la classe "**MovieClick**" nous apporte les renseignements nécessaires :

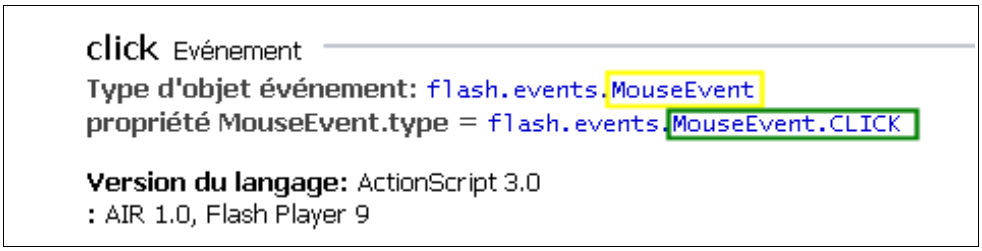


MovieClip — Propriétés | Méthodes | Evénements

Evénements

Masquer les événements hérités

Evénement	Synthèse	Défini par
activate	[Evénement de diffusion] Distribué lorsque Flash Player obtient le focus du système d'exploitation et devient actif.	EventDispatcher
added	Distribué lorsqu'un objet d'affichage est ajouté à la liste d'affichage.	DisplayObject
addedToStage	Envoyé lorsqu'un objet d'affichage est ajouté dans la liste d'affichage de la scène, directement ou par l'intermédiaire d'une arborescence secondaire qui contient l'objet d'affichage.	DisplayObject
clear	Distribué lorsque l'utilisateur sélectionne 'Effacer' (ou 'Supprimer') dans le menu contextuel.	InteractiveObject
click	Distribué lorsque l'utilisateur appuie sur le bouton principal de son périphérique de pointage et le relâche sur la même occurrence de InteractiveObject.	InteractiveObject



click Evénement

Type d'objet événement: flash.events.MouseEvent
propriété MouseEvent.type = flash.events.MouseEvent.CLICK

Version du langage: ActionScript 3.0
: AIR 1.0, Flash Player 9

Nous avons donc :

- ClasseEvenement = **MouseEvent**
- identifiantEvenement = **MouseEvent.CLICK**

Ce qui donne comme code final :

```
// Notre fonction écouteur
function fonctionEcouteur (parametre : MouseEvent) : void
{
    trace ( "Vous avez cliqué sur clip1" );
}

// Ajout de l'écouteur d'événement
clip1.addEventListener ( MouseEvent.CLICK, fonctionEcouteur );
```

Nous pouvons également dire que nous avons **abonné** la fonction "fonctionEcouteur" sur l'événement "click" de l'objet "clip1".

8.2 Suppression d'un écouteur d'événement

Par exemple :

Nous souhaitons que le clip "clip1" ne soit cliquable qu'une seule fois, il faudra supprimer (désabonner) la fonction "fonctionEcouteur" après le premier clic.

La fonction "**removeEventListener**" nous permet de supprimer (désabonner) un écouteur d'événement.

```actionscript
// Notre fonction écouteur
function fonctionEcouteur (parametre : MouseEvent) : void
{
    trace ( "Vous avez cliqué sur clip1" );
    // Suppression de l'écouteur d'événement
    clip1.removeEventListener ( MouseEvent.CLICK, fonctionEcouteur );
}

// Ajout de l'écouteur d'événement
clip1.addEventListener ( MouseEvent.CLICK, fonctionEcouteur );
```

8.3 Propriétés relatives aux événements

Nous pouvons abonner la même fonction à plusieurs événements du même objet.

La propriété "**type**" nous permettra alors de faire la distinction entre ces événements :

Event

Propriétés | Méthodes | Evénements

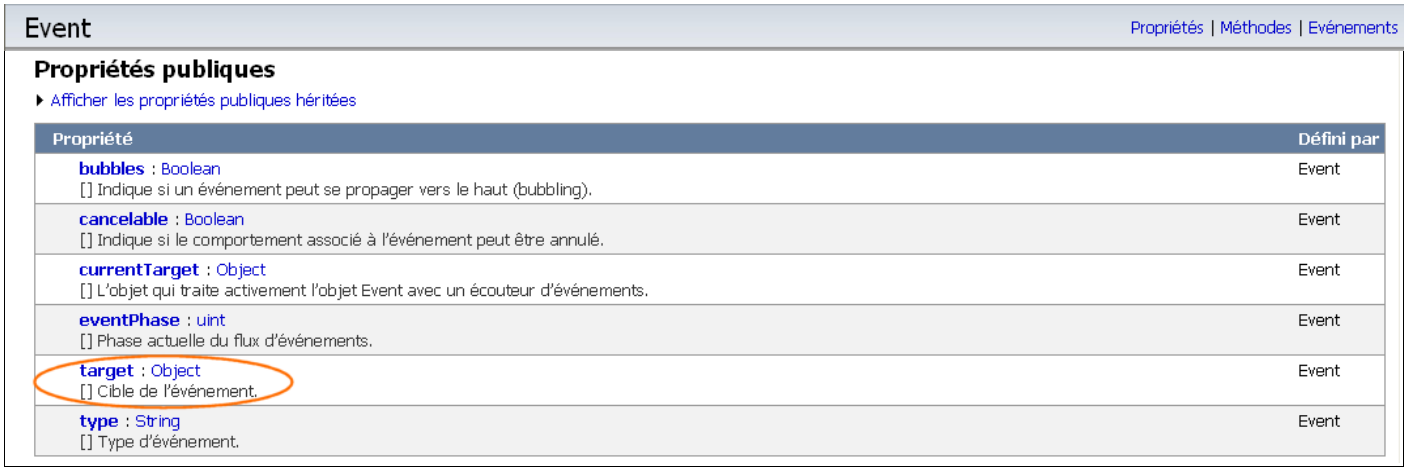
Propriétés publiques

Afficher les propriétés publiques héritées

Propriété	Défini par
bubbles : Boolean [] Indique si un événement peut se propager vers le haut (bubbling).	Event
cancelable : Boolean [] Indique si le comportement associé à l'événement peut être annulé.	Event
currentTarget : Object [] L'objet qui traite activement l'objet Event avec un écouteur d'événements.	Event
eventPhase : uint [] Phase actuelle du flux d'événements.	Event
target : Object [] Cible de l'événement.	Event
type : String [] Type d'événement.	Event

```actionscript
// Notre fonction écouteur
function fonctionEcouteur ( e : MouseEvent ) : void
{
    switch (e.type)
    {
        case MouseEvent.CLICK :
        trace ( "Vous avez cliqué sur clip1" );
        break;

        case MouseEvent.ROLL_OVER :
        trace ( "Vous avez survolé clip1" );
        break;
    }
}

// Ajout des écouteurs d'événement
clip1.addEventListener ( MouseEvent.CLICK, fonctionEcouteur );
clip1.addEventListener ( MouseEvent.ROLL_OVER, fonctionEcouteur );
```

Nous pouvons abonner la même fonction à un ou plusieurs événements de plusieurs objets.
La propriété "**target**" nous permettra alors de faire la distinction entre ces objets :



Event

Propriétés | Méthodes | Evénements

Propriétés publiques

Afficher les propriétés publiques héritées

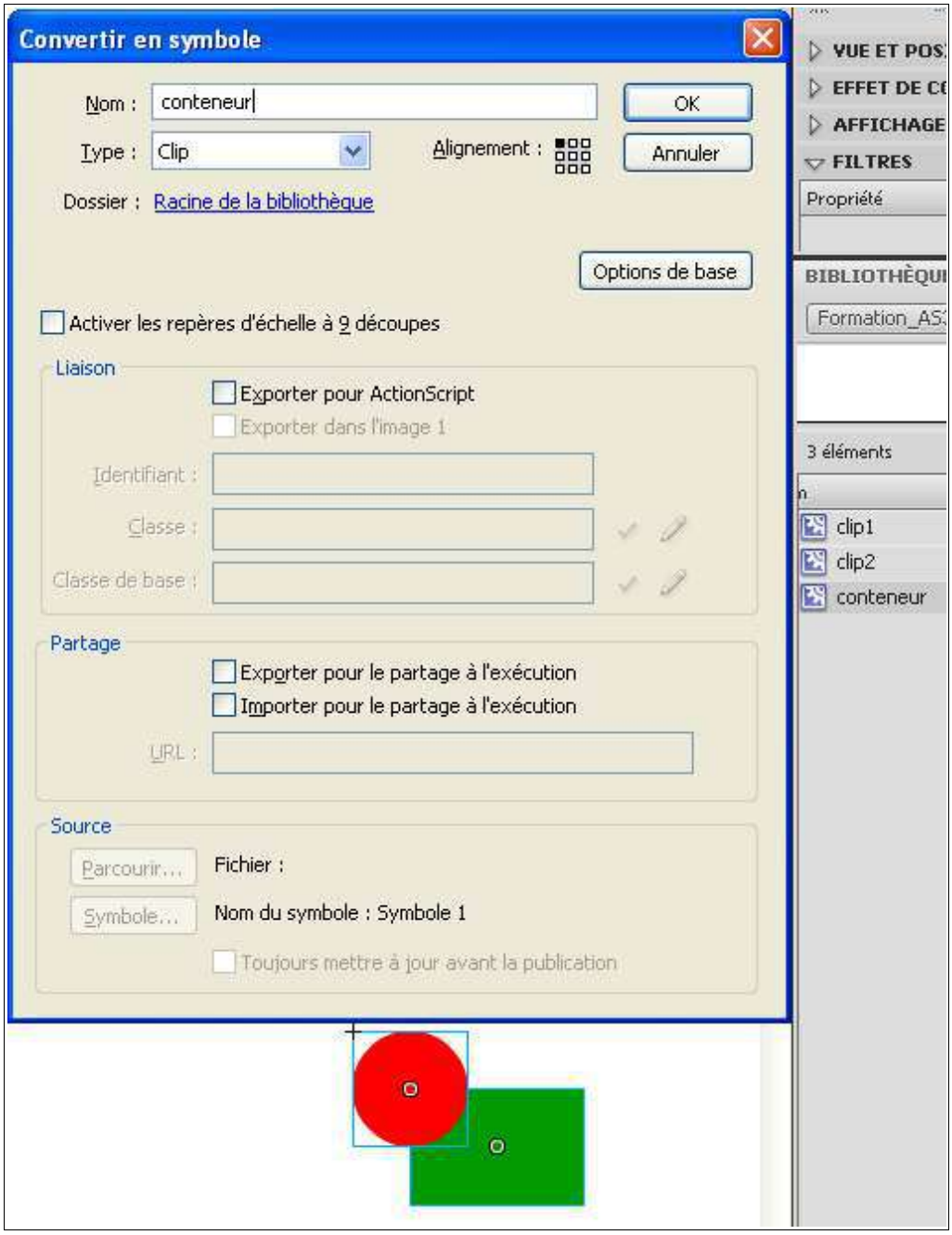
Propriété	Défini par
bubbles : Boolean [] Indique si un événement peut se propager vers le haut (bubbling).	Event
cancelable : Boolean [] Indique si le comportement associé à l'événement peut être annulé.	Event
currentTarget : Object [] L'objet qui traite activement l'objet Event avec un écouteur d'événements.	Event
eventPhase : uint [] Phase actuelle du flux d'événements.	Event
target : Object [] Cible de l'événement.	Event
type : String [] Type d'événement.	Event

```
// Notre fonction écouteur
function fonctionEcouteur ( e : MouseEvent ) : void
{

        if ( e.target == clip1 )
        {
        trace ( "vous avez cliqué sur clip1" );
        }
        else if ( e.target == clip2 )
        {
        trace ( "vous avez cliqué sur clip2" );
        }
}

// Ajout des écouteurs d'événement
clip1.addEventListener ( MouseEvent.CLICK, fonctionEcouteur );
clip2.addEventListener ( MouseEvent.CLICK, fonctionEcouteur );
```

Nous allons également pouvoir utiliser cette propriété associée au principe de la propagation des événements (propagation en bubbling).

Je crée un symbole que j'appelle "**conteneur**" qui regroupe mes deux clips ("clip1" et "clip2").

Je souhaite le rendre plus transparent un clip (clip1 ou clip2) lorsque je clic sur celui-ci.



Je souhaite le rendre plus transparent un clip (clip1 ou clip2) lorsque je clic sur celui-ci.

Bien que la fonction soit abonné sur le conteneur, l'action (grace à la propriété "**target**") se fait bien sur le clip sur lequel l'action a eu lieu.

```
// Notre fonction écouteur
function fonctionEcouteur ( e : MouseEvent ) : void
{
        e.target.alpha = 0.3;
}

// Ajout de l'écouteur d'événement sur le conteneur
conteneur.addEventListener ( MouseEvent.CLICK, fonctionEcouteur );
```

9 Les boucles

9.1 Principe

- Les boucles nous permettent d'effectuer un code un certain nombre de fois pour, par exemple, appliquer un même code à un ensemble ou à une collection d'objets.
- Chaque fois qu'une boucle effectue le code qu'elle contient une fois, nous appelons cela une itération.

9.2 La boucle for

```
for ( var i:int = 0 ; i < 10 ; i++ )
{
    trace (i);
}
```

```
SORTIE
0
1
2
3
4
5
6
7
8
9
```

- La 1ère instruction est exécutée avant la première itération. Elle initialise la variable de boucle (ici à 0).
- La 2ème instruction est une condition, qui va renvoyer true ou false, et qui va indiquer si oui ou non nous commençons la prochaine itération de la boucle.

  Lorsque la condition n'est pas vérifiée (false) la suite du code après la boucle est exécuté.
- La 3ème instruction va s'exécuter à chaque fin d'itération (ici i est incrémenté de 1).

9.3 Les boucles for..each..in et for..in

- Ces boucles permettent de parcourir des objets dynamiques, donc des occurrences de classes dynamiques.
- La boucle **for..each..in** va nous permettre de récupérer les **données** du tableau.
- La boucle **for..in** va nous permettre de récupérer les **index** du tableau qui eux même nous permettent (grâce à la syntaxe à crochets) d'atteindre les données.

```
var liste:Array = new Array( "tata" , "titi" , "toto" );

for each ( var donnee1:String in liste)
{
    trace (donnee1);
}

trace ("_______");

for ( var donnee2:String in liste)
{
    trace (donnee2 , liste[donnee2]);
}
```

```
SORTIE  EDIT
tata
titi
toto

0 tata
1 titi
2 toto
```

9.4 Les boucles while et do..while

- Dans une boucle **while**, une itération est exécutée tant la **condition** renvoie true.

```
while (numChildren > 0)
{
    removeChildAt (0);
}
```

Cet exemple permet de supprimer tous les objets affichés sur la scène (en supprimant celui d'index 0 à chaque fois).

```
do
{
    .. code
}
while (condition)
```

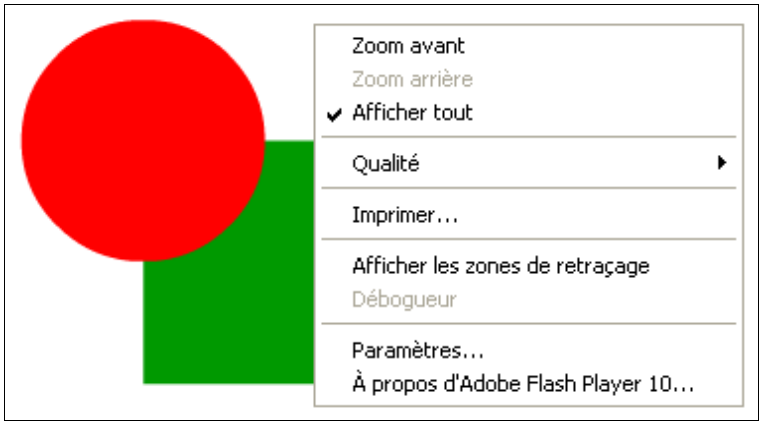
- La boucle **do..while** exécute, quoi qu'il arrive, une première itération. Puis vérifie si la **condition** renvoie true et passe alors à la deuxième itération.

10 L'affichage (2ème partie)

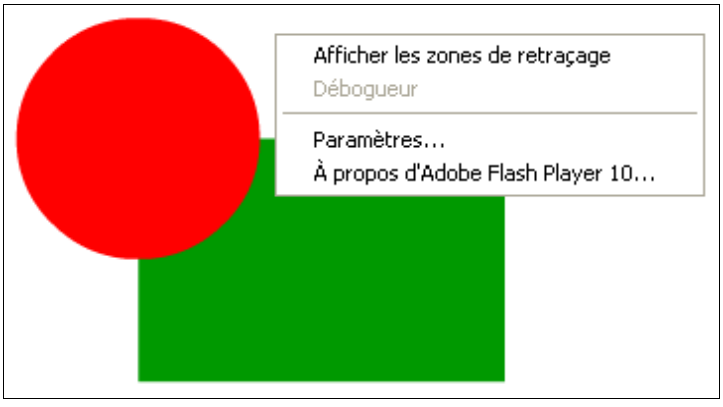
10.1 Menu contextuel sur les objets

10.1.1 Menu contextuel de l'application

Par défaut, le menu contextuel du Flash Player se présente comme suit :



Il est possible de réduire au strict minimum son menu contextuel (comme ci-dessous)



Pour ceci nous utiliserons :

- une occurrence de la classe "**ContextMenu**",
- la méthode "**hideBuiltInItems**" appliquée à notre occurrence,
- la propriété "**contextMenu**" appliquée ici à this (optionnel), c'est à dire au root de notre application.

  Nous retrouverons cette propriété sur tous les objets interactifs (voir arbre d'héritage).

```
var menuGeneral:ContextMenu = new ContextMenu ();
menuGeneral.hideBuiltInItems ();
this.contextMenu = menuGeneral;
```

10.1.2 Menu contextuel d'un objet

Sur le même principe, nous allons appliquer un menu contextuel à un champ texte en utilisant :

- une occurrence de la classe "**ContextMenu**",
- une occurrence de la classe "**ContextMenuItem**" pour créer et paramétrer un élément du menu,
- la propriété "**customItems**" pour ajouter cet élément,
- la propriété "**contextMenu**" appliquée ici à notre champ texte.

ContextMenuItem() Constructeur

```
public function ContextMenuItem(caption:String, separatorBefore:Boolean = false, enabled:Boolean = true, visible:Boolean = true)
```

Version du langage: ActionScript 3.0
: AIR 1.0, Flash Player 9

Crée un objet ContextMenuItem pouvant être ajouté au tableau `ContextMenu.customItems`.

Paramètres

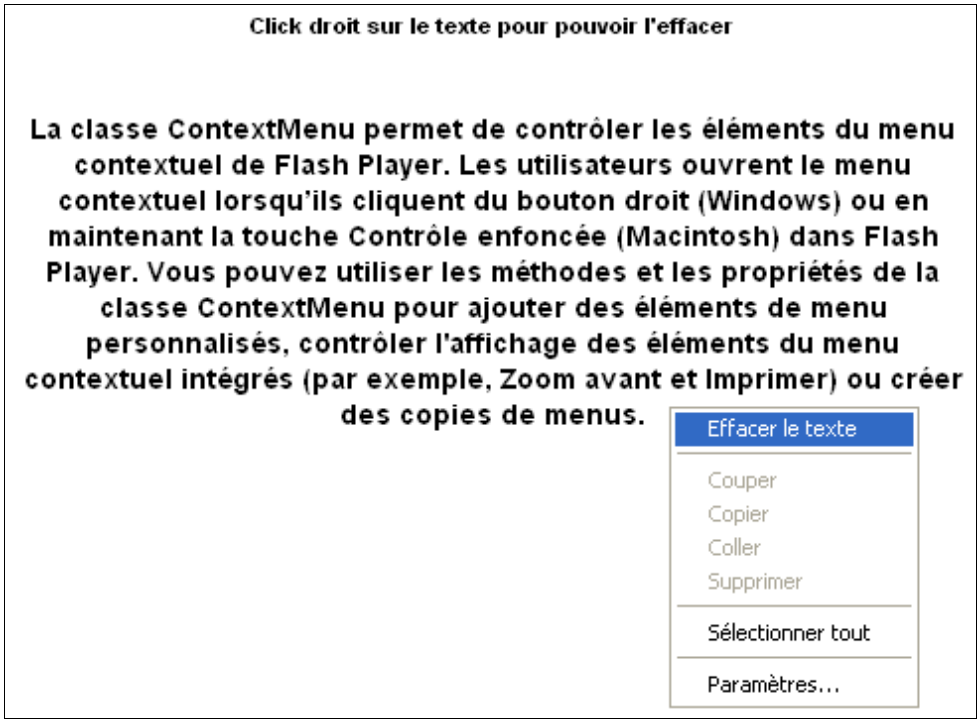
`caption:String` — Spécifie le texte associé à l'élément de menu. Pour plus d'informations sur les restrictions liées à la valeur `caption`, consultez la présentation de la classe ContextMenuItem.

`separatorBefore:Boolean` (default = `false`) — Spécifie si une barre de séparation doit apparaître au-dessus de l'élément dans le menu contextuel. La valeur par défaut est `false`.

`enabled:Boolean` (default = `true`) — Indique si l'élément de menu est activé ou désactivé dans le menu contextuel. La valeur par défaut est `true` (activé). Ce paramètre est facultatif.

`visible:Boolean` (default = `true`) — Indique si l'élément de menu est visible ou invisible. La valeur par défaut est `true` (visible).

```
var menuChamp:ContextMenu = new ContextMenu ();
var itemEffacer:ContextMenuItem = new ContextMenuItem ("Effacer le texte");
menuChamp.customItems = new Array (itemEffacer);
champTexte.contextMenu = menuChamp;
```



Nous allons devoir maintenant donner une action à cet élément du menu en utilisant un écouteur d'événement en appliquant la méthode vue au chapitre 9.1 :

```
itemEffacer.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, ecouteurItemEffacer);

function ecouteurItemEffacer (e:ContextMenuEvent):void
{
    champTexte.text = "";
}
```

Ce qui donne au final :

```
var menuGeneral:ContextMenu = new ContextMenu ();
menuGeneral.hideBuiltInItems();
this.contextMenu = menuGeneral;

var menuChamp:ContextMenu = new ContextMenu ();
var itemEffacer:ContextMenuItem = new ContextMenuItem ("Effacer le texte");
menuChamp.customItems = new Array (itemEffacer);
champTexte.contextMenu = menuChamp;

itemEffacer.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, ecouteurItemEffacer);

function ecouteurItemEffacer (e:ContextMenuEvent):void
{
    champTexte.text = "";
}
```

10.2 Les filtres

Les objets graphiques représentés par la classe "**DisplayObject**" possèdent une propriété "**filters**" de type "Array" ce qui sous-entend que nous pouvons appliquer plusieurs filtres, une liste de filtres, sur un objet graphique.

Nous trouvons dans la documentation, dans le package "flash.filters" l'ensemble des classes liées aux filtres graphiques :

flash.filters

Le package flash.filters contient des classes pour les effets de filtrage de bitmaps. Les filtres permettent d'appliquer des effets visuels riches, tels que les effets de flou, biseau, rayonnement et ombres portées pour afficher des objets.

Classes

Classe	Description
BevelFilter	La classe BevelFilter permet d'appliquer un effet de biseau à des objets d'affichage.
BitmapFilter	La classe BitmapFilter est la classe de base pour tous les effets de filtrage d'image.
BitmapFilterQuality	La classe BitmapFilterQuality contient des valeurs permettant de définir la qualité de rendu d'un objet BitmapFilter.
BitmapFilterType	La classe BitmapFilterType contient des valeurs permettant de définir le type d'un objet BitmapFilter.
BlurFilter	La classe BlurFilter permet d'appliquer un effet visuel de flou aux objets d'affichage.
ColorMatrixFilter	La classe ColorMatrixFilter vous permet d'appliquer une transformation de matrice 4 x 5 aux valeurs de couleur RVBA et alpha de chaque pixel de l'image d'entrée afin d'obtenir un résultat intégrant un nouvel ensemble de valeurs de couleur RVBA et alpha.
ConvolutionFilter	La classe ConvolutionFilter applique un effet de filtre de convolution de matrice.
DisplacementMapFilter	La classe DisplacementMapFilter utilise les valeurs de pixels de l'objet BitmapData spécifié (appelé image de mappage du déplacement) pour déplacer un objet.
DisplacementMapFilterMode	La classe DisplacementMapFilterMode fournit des valeurs à la propriété mode de la classe DisplacementMapFilter.
DropShadowFilter	La classe DropShadowFilter permet d'ajouter un effet d'ombre portée aux objets d'affichage.
GlowFilter	La classe GlowFilter permet d'appliquer un effet de rayonnement aux objets d'affichage.
GradientBevelFilter	La classe GradientBevelFilter permet d'appliquer un effet de biseau en dégradé à des objets d'affichage.
GradientGlowFilter	La classe xxxGradientGlowFilter permet d'appliquer un effet de rayonnement dégradé à des objets d'affichage.
ShaderFilter	La classe ShaderFilter applique un filtre en exécutant un shader sur l'objet filtré.

Par exemple, nous souhaitons appliquer à notre clip une ombre portée et un effet biseau.

```
var ombre:DropShadowFilter = new DropShadowFilter ();
var biseau:BevelFilter = new BevelFilter ();

var listeFiltres:Array = new Array (ombre, biseau);

clip.filters = listeFiltres;
```

Nous avons la possibilité, lorsque nous construisons l'occurrence, de renseigner certains paramètres

```
1  var ombre:DropShadowFilter = new DropShadowFilter ();
new DropShadowFilter([distance:Number=4.0,angle:Number=45,couleur:uint=0,alpha:Number=1.0,flouX:Number=4.0,flouY:Number=4.0,force:I
```

Pour annuler l'effet des filtres, ajouter :

```
clip.filters = null;
```

10.3 Les modes de fusion

Lorsque nous consultons la propriété "**blendMode**" de la classe "**DisplayObject**" nous avons un aperçu des différentes possibilités de fusion :

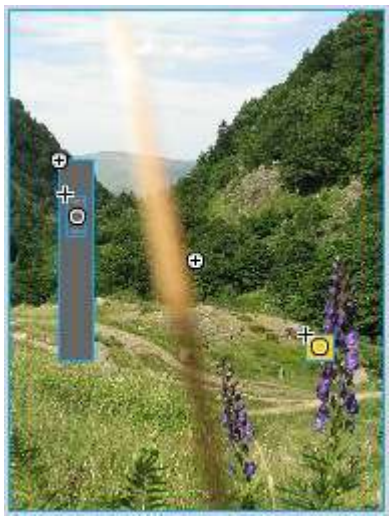
Constante BlendMode	Illustration	Description
BlendMode.NORMAL		L'objet d'affichage apparaît devant l'arrière-plan. Les valeurs de pixels de l'objet d'affichage écrasent celles de l'arrière-plan. Lorsque l'objet d'affichage est transparent, l'arrière-plan est visible.
BlendMode.LAYER		Impose la création d'un groupe de transparences pour l'objet d'affichage. Cela signifie que l'objet d'affichage est précomposé dans un tampon temporaire avant que son traitement ne se poursuive. Cette opération s'exécute automatiquement si l'objet d'affichage est pré-placé en mémoire cache par le biais d'une mise en cache des bitmaps ou s'il correspond à un conteneur d'objet d'affichage qui possède au moins un objet enfant associé à un réglage blendMode autre que BlendMode.NORMAL.
BlendMode.MULTIPLY		Multiplie les valeurs des couleurs élémentaires de l'objet d'affichage par celles de la couleur d'arrière-plan, puis les normalise en les divisant par 0xFF, ce qui donne des couleurs plus sombres. Ce réglage est souvent utilisé pour les effets d'ombre et de profondeur. Par exemple, si une couleur élémentaire (comme le rouge) d'un pixel de l'objet d'affichage et la couleur correspondante du pixel de l'arrière-plan ont toutes les deux une valeur de 0x88, le résultat de la multiplication est 0x4840. La division par 0xFF donne une valeur de 0x48 pour cette couleur élémentaire, qui est plus sombre que celle de l'objet d'affichage ou de l'arrière-plan.
BlendMode.SCREEN		Multiplie le complément (l'inverse) de la couleur de l'objet d'affichage par le complément de la couleur d'arrière-plan, ce qui donne un effet de blanchissement. Ce réglage est couramment utilisé pour la mise en valeur ou pour supprimer les parties noires de l'objet d'affichage.
BlendMode.LIGHTEN		Sélectionne les plus claires des couleurs élémentaires de l'objet d'affichage et la couleur d'arrière-plan (celles qui ont les valeurs les plus élevées). Ce réglage est généralement utilisé pour les superpositions. Par exemple, si l'objet d'affichage possède un pixel dont la valeur RVB est 0xFFCC33, et que le pixel d'arrière-plan possède une valeur RVB réglée sur 0xDDF800, la valeur RVB obtenue pour le pixel affiché est 0xFFF833 (car 0xFF > 0xDD, 0xCC < 0xF8 et 0x33 > 0x00 = 33).
BlendMode.DARKEN		Sélectionne les plus sombres des couleurs élémentaires de l'objet d'affichage et de l'arrière-plan (celles qui ont les valeurs les plus faibles). Ce réglage est généralement utilisé pour les superpositions. Par exemple, si l'objet d'affichage possède un pixel dont la valeur RVB est 0xFFCC33, et que le pixel d'arrière-plan possède une valeur RVB réglée sur 0xDDF800, la valeur RVB obtenue pour le pixel affiché est 0xDDCC00 (car 0xFF > 0xDD, 0xCC < 0xF8 et 0x33 > 0x00 = 33).
BlendMode.DIFFERENCE		Compare les couleurs élémentaires de l'objet d'affichage à celles de son arrière-plan et soustrait la valeur la plus sombre des deux couleurs élémentaires de la plus claire. Ce réglage est habituellement utilisé pour obtenir des couleurs plus vibrantes. Par exemple, si l'objet d'affichage possède un pixel dont la valeur RVB est 0xFFCC33, et le pixel d'arrière-plan possède une valeur RVB réglée sur 0xDDF800, la valeur RVB résultante du pixel affiché est 0x222C33 (parce que 0xFF - 0xDD = 0x22, 0xF8 - 0xCC = 0x2C et 0x33 - 0x00 = 0x33).
BlendMode.ADD		Ajoute les valeurs des couleurs élémentaires de l'objet d'affichage à celles de son arrière-plan, en appliquant un plafond de 0xFF. Ce réglage est habituellement utilisé pour animer un fondu d'éclaircissement entre deux objets. Par exemple, si l'objet d'affichage possède un pixel dont la valeur RVB est 0xAAA633, et que le pixel d'arrière-plan possède une valeur RVB réglée sur 0xDD2200, la valeur RVB résultante du pixel affiché est 0xFFC833 (parce que 0xAA + 0xDD > 0xFF, 0xA6 + 0x22 = 0xC8 et 0x33 + 0x00 = 0x33).
BlendMode.SUBTRACT		Soustrait les valeurs des couleurs élémentaires de l'objet d'affichage de celles de la couleur d'arrière-plan, en appliquant un plancher de 0. Ce réglage est habituellement utilisé pour animer un fondu de plus en plus sombre entre deux objets. Par exemple, si l'objet d'affichage possède un pixel dont la valeur RVB est 0xAA2233, et que le pixel d'arrière-plan a une valeur RVB réglée sur 0xDDA600, alors la valeur RVB obtenue pour le pixel affiché est 0x338400 (car 0xDD - 0xAA = 0x33, 0xA6 - 0x22 = 0x84 et 0x00 - 0x33 < 0x00).
BlendMode.INVERT		Inverse l'arrière-plan.
BlendMode.ALPHA		Applique la valeur alpha de chaque pixel de l'objet d'affichage à l'arrière-plan. Pour ce faire, le réglage blendMode de l'objet d'affichage parent doit être réglé sur BlendMode.LAYER. Par exemple, dans l'illustration, l'objet d'affichage parent, qui est un arrière-plan blanc, a un paramètre blendMode = BlendMode.LAYER.
BlendMode.ERASE		Efface l'arrière-plan sur la base de la valeur alpha de l'objet d'affichage. Pour ce faire, le réglage blendMode de l'objet d'affichage parent doit être défini sur BlendMode.LAYER. Par exemple, dans l'illustration, l'objet d'affichage parent, qui est un arrière-plan blanc, a un paramètre blendMode = BlendMode.LAYER.

La plupart vont s'appliquer simplement, mais certains comme "BlendMode.ERASE" vont nécessiter que le BlendMode de l'objet parent soit défini sur "LAYER" :

```
clip1.blendMode = BlendMode.ERASE;
this.blendMode = BlendMode.LAYER;
```

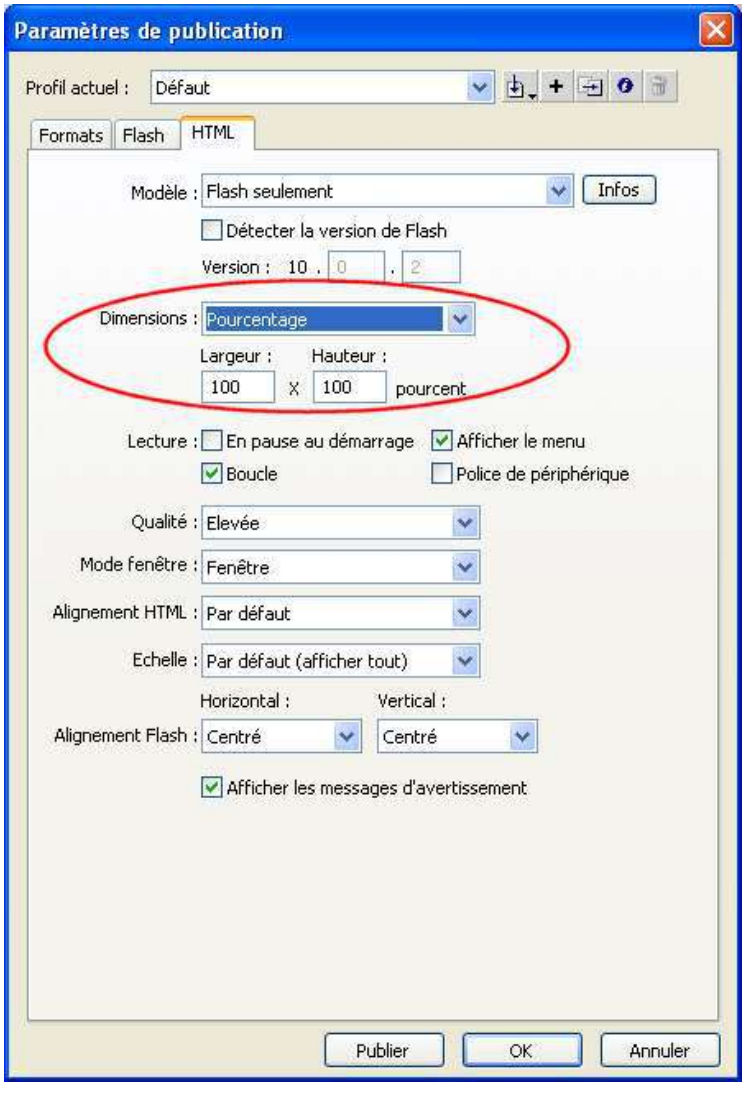
10.4 Repositionnement et redimensionnement des objets

Nous voulons par exemple réaliser un site totalement en Flash qui occupe toute la place disponible dans le navigateur et qui utilise un repositionnement et un redimensionnement intelligent des différents objets de la scène.



Nous avons besoin d'utiliser les fonctionnalités de l'objet "Stage" :

- La première chose à faire est de vérifier que les paramètres de publication sont réglés sur des dimensions en pourcentage réglées sur 100 % (par défaut : Identique à l'animation) :



```actionscript
// Alignement en haut à gauche
stage.align = StageAlign.TOP_LEFT;

// Dimensions et positions fixes
stage.scaleMode = StageScaleMode.NO_SCALE;

// Écouteur d'événement sur le Stage : Redimensionnement
stage.addEventListener (Event.RESIZE, stageResize);

// Fonction Ecouteur : Redimensionnement Stage
function stageResize (e:Event):void
{
    repositionnerElements ();
}

// Fonction : Règles de positionnement des éléments
function repositionnerElements ():void
{
    // Menu toujours sur toute la hauteur
    menu.height = stage.stageHeight;
    // Logo toujours en bas à droite
    logo.x = stage.stageWidth - logo.width;
    logo.y = stage.stageHeight - logo.height;
    // Image toujours centrée
    image.x = stage.stageWidth / 2;
    image.y = stage.stageHeight / 2;
}
```

10.5 Chargement de swf et d'images

Nous allons utiliser la propriété "**contentLoaderInfo**" de la classe "**Loader**" qui renvoie un objet "**LoaderInfo**" correspondant à l'objet en cours de chargement.

Loader — Propriétés | Méthodes | Evénements

Propriétés publiques

Afficher les propriétés publiques héritées

Propriété	Défini par
content : DisplayObject [] Contient l'objet d'affichage racine du fichier SWF ou du fichier d'image (JPG, PNG ou GIF) qui a été chargé à l'aide de la méthode load() ou loadBytes().	Loader
contentLoaderInfo : LoaderInfo [] Renvoie un objet LoaderInfo qui correspond à l'objet en cours de chargement.	Loader

Pour gérer les chargement des swf et des images, nous allons utiliser les **événements** de la classe "**LoaderInfo**" :

LoaderInfo — Propriétés | Méthodes | Evénements

Evénements

Afficher les événements hérités

Evénement	Synthèse	Défini par
complete	Distribué lorsque le chargement de données aboutit.	LoaderInfo
httpStatus	Distribué lorsqu'une requête réseau est envoyée via HTTP et que Flash Player peut détecter le code d'état HTTP.	LoaderInfo
init	Distribué lorsqu'il est possible d'accéder aux propriétés et aux méthodes d'un fichier SWF chargé et de les utiliser.	LoaderInfo
ioError	Distribué lorsqu'il se produit une erreur d'entrée ou de sortie entraînant l'échec d'une opération de chargement.	LoaderInfo
open	Distribué lors du démarrage d'une opération de chargement.	LoaderInfo
progress	Distribué lors de la réception des données au fur et à mesure du téléchargement.	LoaderInfo
unload	Distribué par un objet LoaderInfo lorsqu'un objet chargé est supprimé à l'aide de la méthode unload() de l'objet Loader ou lorsqu'un second chargement est effectué par le même objet Loader et que le contenu d'origine est supprimé avant le début du chargement.	LoaderInfo

Comme à l'accoutumée la documentation nous indique l'identifiant événement pour chaque écouteur :

progress Evénement
Type d'objet événement: flash.events.ProgressEvent
propriété ProgressEvent.type = flash.events.ProgressEvent.PROGRESS

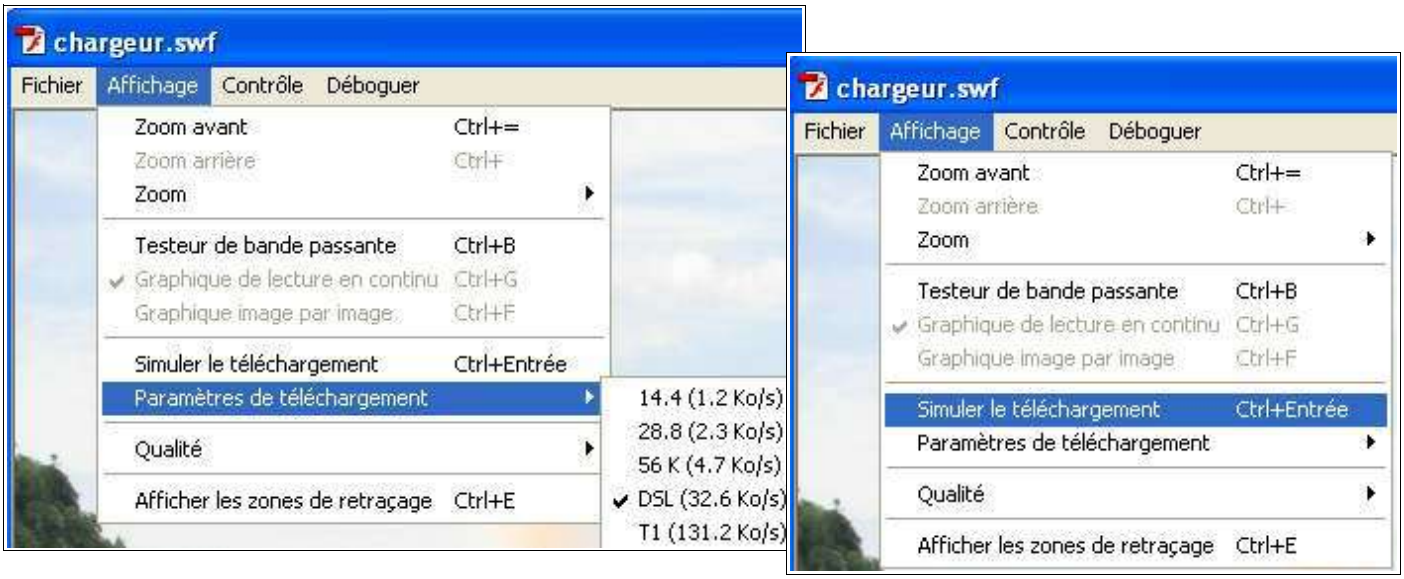
complete Evénement
Type d'objet événement: flash.events.Event
propriété Event.type = flash.events.Event.COMPLETE

ioError Evénement
Type d'objet événement: flash.events.IOErrorEvent
propriété IOErrorEvent.type = flash.events.IOErrorEvent.IO_ERROR

```
// Chargement de image.jpg
var loader:Loader = new Loader ();
loader.load (new URLRequest ("image.jpg"));
addChild (loader);

// Écouteur d'événement : Progression du chargement
loader.contentLoaderInfo.addEventListener (ProgressEvent.PROGRESS, chargementProgression);

// Écouteur d'événement : Chargement terminé
loader.contentLoaderInfo.addEventListener (Event.COMPLETE, chargementTermine);

// Écouteur d'événement : Erreur au chargement
loader.contentLoaderInfo.addEventListener (IOErrorEvent.IO_ERROR, chargementErreur);

// Fonction Écouteur : Progression Chargement
function chargementProgression (e:ProgressEvent):void
{
    barreProgression.scaleX = e.bytesLoaded/e.bytesTotal;
}

// Fonction Écouteur : Chargement terminé
function chargementTermine (e:Event):void
{
    trace( "Dimensions originales du contenu chargé :", loader.contentLoaderInfo.width, loader.contentLoaderInfo.height );
}

// Fonction Écouteur : Erreur Chargement
function chargementErreur (e:IOErrorEvent):void
{
    trace("fichier non trouvé");
}
```

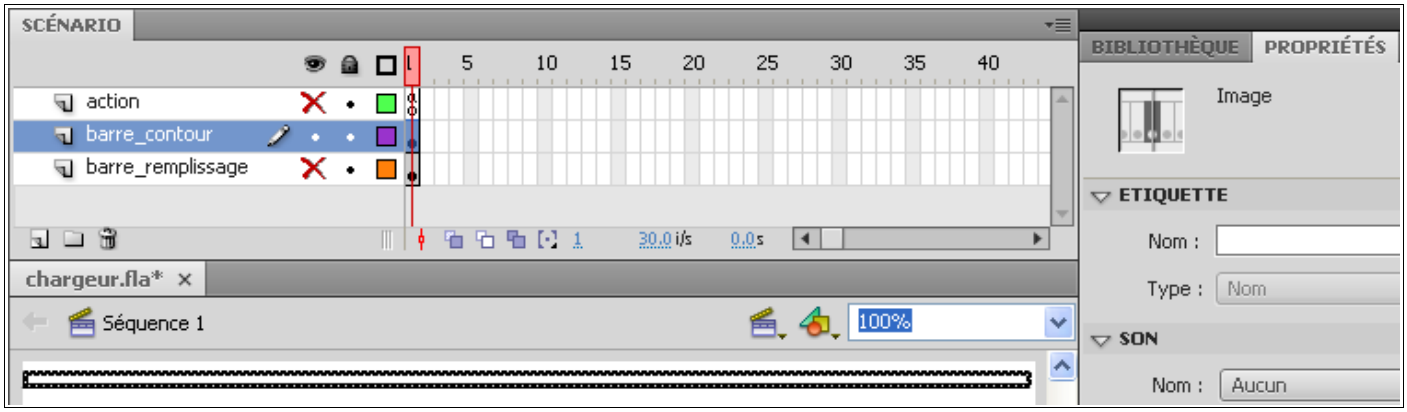
On peut simuler un téléchargement pour voir la progression, dans le Flash Player :

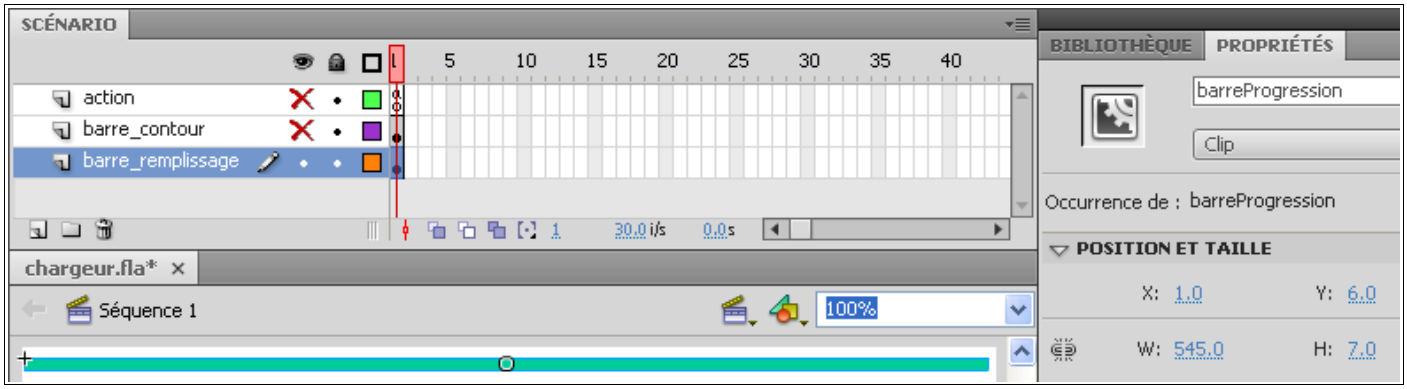
Régler les paramètres de téléchargement, puis cliquer sur "**Simuler le téléchargement**"



Construction de la barre de progression :

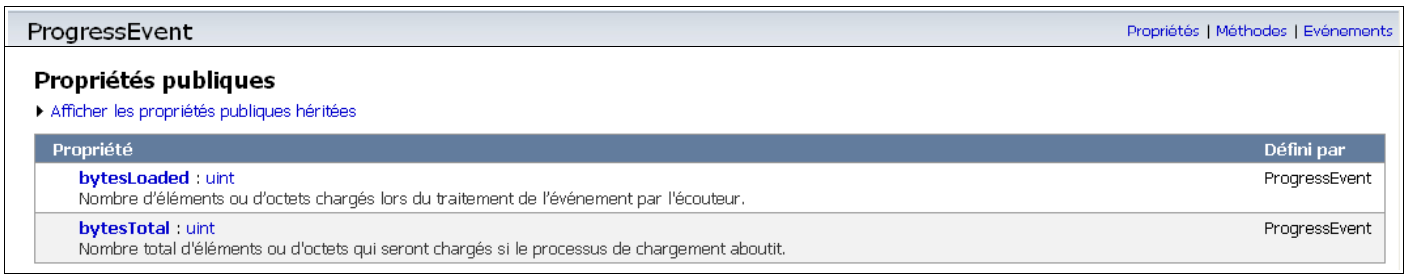
- Nous créons deux calques : barre_contour et barre_remplissage.
- Sur le calque barre_contour , nous dessinons un rectangle (bords noirs, remplissage vert) sur la largeur de la scène.
- Nous convertissons le remplissage en symbole (Clip) que nous appelons "**barreProgression**".
- Nous le coupons et le collons en place sur un calque inférieur pour garder le contour visible (barre_remplissage), nous donnons le même nom d'occurrence que le nom en bibliothèque ("**barreProgression**").





Pour animer la barre de progression :

Nous utilisons les propriétés "**bytesLoaded**" et "**bytesTotal**" de la classe "**ProgressEvent**" pour calculer le rapport de chargement et le transmettre à la propriété "**scaleX**" de la barre de progression :



10.6 Déchargement d'un swf

Lorsque nous déchargeons un swf externe (par exemple par un nouveau chargement) qui lance une musique ou une vidéo, nous nous apercevons que celles-ci continuent de jouer.

Il faut savoir que la méthode "**unload**" du chargeur (effectuée automatiquement lors du chargement du swf suivant) ne fait que le supprimer de l'affichage mais l'objet continue d'exister.

Dans le Flash Player 10, une méthode "**unloadAndStop**" a été rajoutée sur la classe "**Loader**" qui permet de supprimer tous les comportements gênants de ce swf.

Dans le Flash Player 9, cette méthode n'existe pas, il va falloir utiliser un autre moyen. Nous allons le gérer dans le swf externe, c'est lui qui lance la musique donc c'est à lui de l'arrêter :

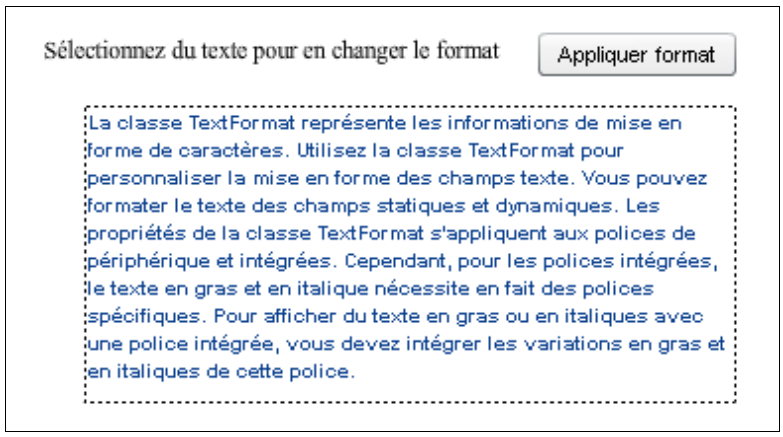
```
var musique:Sound = new Sound();
musique.load(new URLRequest("mp3/musique.mp3"));

var canal:SoundChannel = musique.play();

// Écouteur d'événement : UNLOAD
root.loaderInfo.addEventListener(Event.UNLOAD, swfDecharge);

// Fonction écouteur : Déchargement SWF
function swfDecharge(e:Event):void
{
    root.loaderInfo.removeEventListener(Event.UNLOAD, swfDecharge);

    trace("SWF externe déchargé");

    // Gestion de l'erreur :
    // si le chargement de la musique n'a pas encore commencé ou est déjà terminé
    try
    {
        musique.close(); // Stoppe chargement données
    }
    catch(err:Error){};

    canal.stop(); // Stoppe lecture son
}
```

11 Mise en forme du texte

11.1 Avec la classe TextFormat

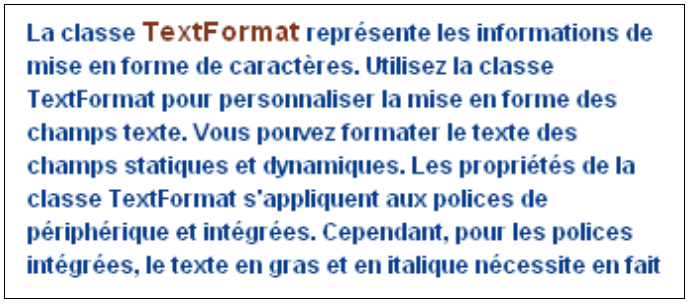
La classe "**TextFormat**" nous permet d"effectuer un formatage sur un objet "**TextField**" que nous allons pouvoir appliquer sur la totalité de TextField ou seulement sur une partie de son contenu.

Elle nous permet d'appliquer un format de texte par défaut (par "**defaultTextFormat**") ainsi qu'un formatage sur une partie du texte (par "**setTextFormat**") délimitée par index de début et un index de fin.



Nous avons sur la scène, un champ de texte nommé "**champ**" ainsi qu'un composant bouton nommé "**boutonAppliquer**". Nous allons appliquer un formatage sur l'ensemble du champ puis permettre un formatage plus spécifique sur la sélection faite par l'utilisateur après avoir cliquer sur le bouton.

```
// Format global
var formatGlobal:TextFormat = new TextFormat ("_sans", 12);

// Formatage global de champ
champ.setTextFormat (formatGlobal);

// Format sélection
var formatSelection:TextFormat = new TextFormat ("_sans", 14, 0x80371A);

// Écouteur d'événement : Clic sur le bouton boutonAppliquer
boutonAppliquer.addEventListener (MouseEvent.CLICK, appliquerFormatSurSelection);

// Fonction écouteur : Formatage de la sélection
function appliquerFormatSurSelection (e:MouseEvent):void
{
    champ.setTextFormat (formatSelection, champ.selectionBeginIndex, champ.selectionEndIndex);
}
```



11.2 En utilisant HTML et CSS dans des fichiers externes

Nous allons utiliser des fichiers externes, un qui contient le texte formaté HTML et un autre qui contient notre feuille de style :

texte.txt

```
<h3>Utilisation avancée du texte avec <span class="clef">Flash CS3</span></h3>
<p>
    Cet exemple <a href="http://google.fr" target="_blank">illustre</a>
    l'utilisation d'une <span class="clef">feuille de styles</span> et d'un texte externe.
    <br />
</p>
```

styles.css

```
h3 {
    font-size:18px;
    font-family:_serif;
    color:#000000;
}

p {
    font-size:12px;
    font-family:_sans;
    color:#0;
}

.clef {
    font-weight:bold;
}

a {
    color:#0000ff;
    text-decoration:underline;
    font-weight:bold;
}

a:hover {
    text-decoration:none;
}
```

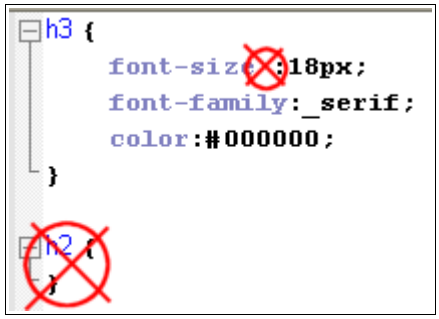
Utilisation avancée du texte avec Flash CS3

Cet exemple **illustre** l'utilisation d'une **feuille de styles** et d'un texte externe.

```actionscript
// Occurence champ de TextField
var champ:TextField = new TextField ();

// formatage de champ
// position
champ.x = champ. y = 100;
// taille
champ.width = champ.height = 400;
// supression des RC non gérés par HTML
champ.condenseWhite = true;
// multiligne
champ.multiline = true;
// retour automatique
champ.wordWrap = true;

// Suppression Pb si CSS chargé après HTML
champ.styleSheet = new StyleSheet ();

// Affichage de champ
addChild (champ);

// Chargement du fichier texte
var chargeurTxt:URLLoader = new URLLoader (new URLRequest ("txt/texte.txt"));

// Écouteur d'événement : Fichier txt chargé
chargeurTxt.addEventListener (Event.COMPLETE, texteCharge);

// Fonction écouteur : remplissage texte
function texteCharge (e:Event):void
{
    champ.htmlText =  chargeurTxt.data;
}

// Chargement du fichier CSS
var chargeurCss:URLLoader = new URLLoader (new URLRequest ("css/styles.css"));

// Écouteur d'événement : Fichier CSS chargé
chargeurCss.addEventListener (Event.COMPLETE, cssCharge);

// Fonction écouteur : mise en forme par CSS
function cssCharge (e:Event):void
{
    // Occurence styles de StyleSheet
    var styles:StyleSheet = new StyleSheet ();
    // Application des styles
    styles.parseCSS(chargeurCss.data);
    champ.styleSheet = styles;
}
```

Attention: Flash est assez sensible à la syntaxe des CSS :

- Pas d'espace entre le nom et les :.
- Pas de styles vides.

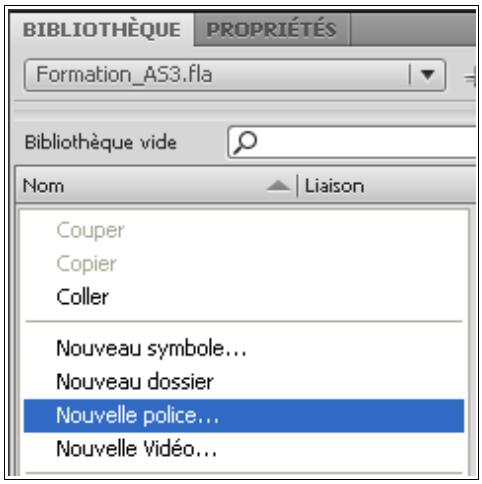


11.3 Intégration de polices de caractères

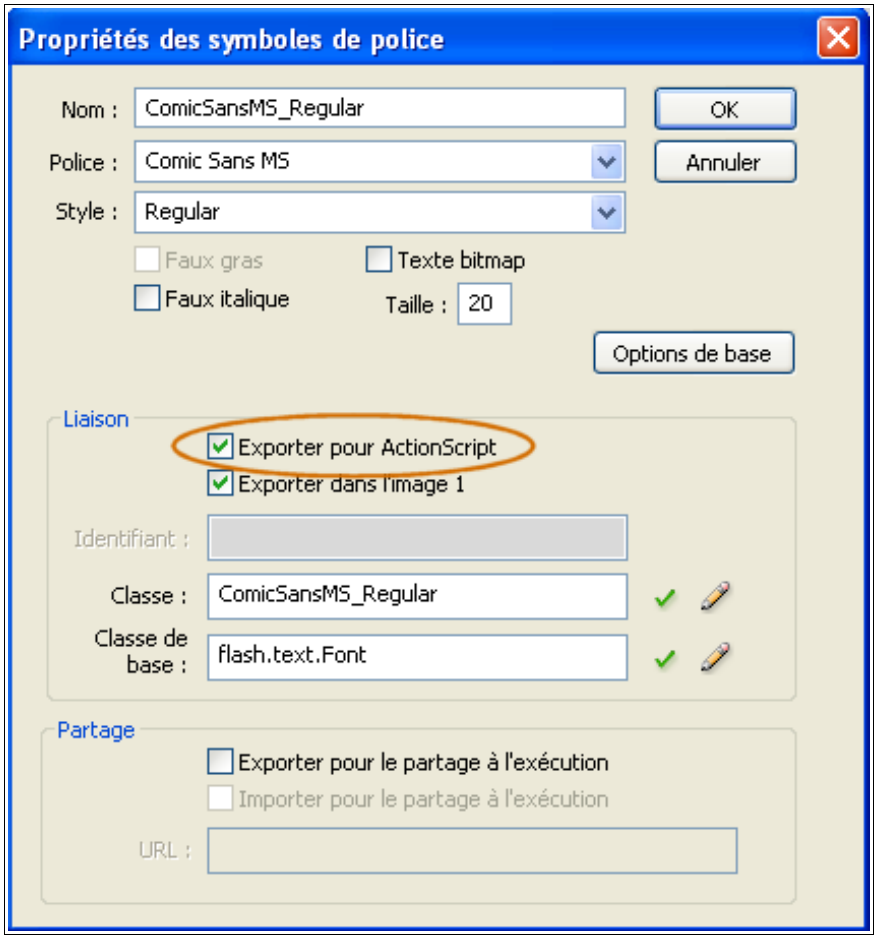
L'intégration de polices dans un swf nous permet de nous assurer que l'utilisateur visualisera nos textes avec la police que nous avons choisie dans Flash, même s'il ne possède pas cette police sur son système.

L'intégration de police alourdira le swf d'un poids qui dépendra de la police et des styles que nous aurons intégrés.

– Nous allons commencer par créer en bibliothèque une police par un clic droit sur la bibliothèque :



– Nous choisissons la police et le style, nous lui donnons un nom et nous cochons "**Exporter pour ActionScript**"



- Nous devons définir un format en lui indiquant le **nom réel de la police**.

- Nous devons l'appliquer comme format par défaut.

- Nous devons indiquer que l'on utilise des polices intégrées.

```
var format:TextFormat = new TextFormat ("Comic Sans MS");

var champ:TextField = new TextField ();
champ.defaultTextFormat = format;
champ.x = champ. y = 100;
champ.width = 400;
champ.height = 300;
champ.condenseWhite = true;
champ.multiline = true;
champ.wordWrap = true;
champ.styleSheet = new StyleSheet ();
champ.text = "Intégration de polices";
champ.embedFonts = true;
champ.antiAliasType = AntiAliasType.ADVANCED;

addChild (champ);
```

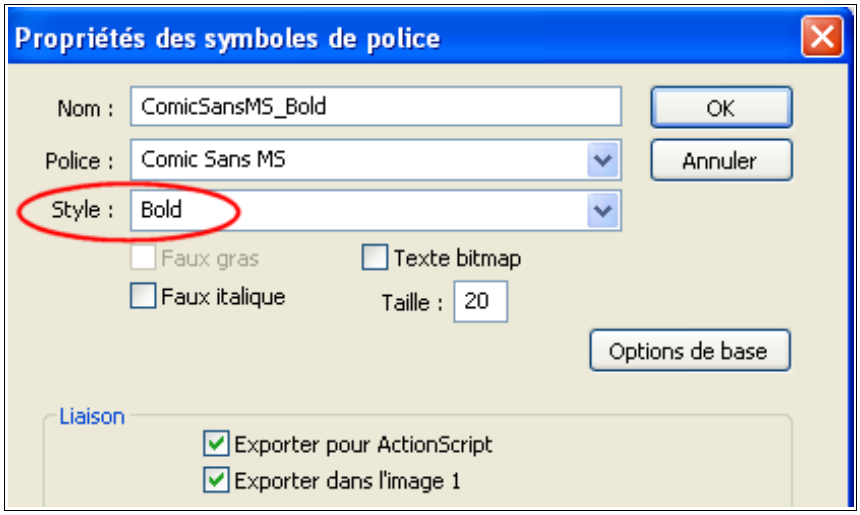
- Nous avons la possibilité de lisser les polices, pour les rendre plus lisibles, en utilisant la propriété "**ADVANCED**" de l'**anti aliasing**.

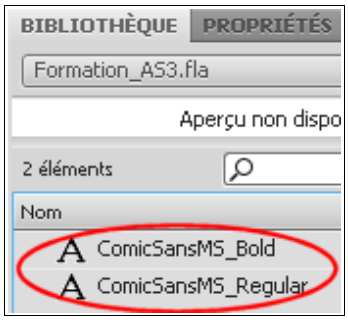
Si nous souhaitons utiliser un **format HTML** (pour, par exemple mettre **en gras** le mot polices) :

- Nous devons le spécifier dans le code par la propriété **htmlText**.
- Nous devons insérer les balises HTML dans le texte.

```
var format:TextFormat = new TextFormat ("Comic Sans MS");

var champ:TextField = new TextField ();
champ.defaultTextFormat = format;
champ.x = champ. y = 100;
champ.width = 400;
champ.height = 300;
champ.condenseWhite = true;
champ.multiline = true;
champ.wordWrap = true;
champ.styleSheet = new StyleSheet ();
champ.htmlText = "Intégration de <b>polices</b>";
champ.embedFonts = true;
champ.antiAliasType = AntiAliasType.ADVANCED;

addChild (champ);
```

– Nous allons devoir intégrer une deuxième fois notre police en lui indiquant le style qui convient.





Si nous utilisons les polices intégrées et une feuille de styles externe, nous devons indiquer dans **font-family** uniquement le nom réel de la police :

```
var format:TextFormat = new TextFormat ("Comic Sans MS");

var champ:TextField = new TextField ();
champ.defaultTextFormat = format;
champ.x = champ. y = 100;
champ.width = 400;
champ.height = 300;
champ.condenseWhite = true;
champ.multiline = true;
champ.wordWrap = true;
champ.styleSheet = new StyleSheet ();
champ.htmlText = "<h3>Intégration de <b>polices</b></h3>";
champ.embedFonts = true;
champ.antiAliasType = AntiAliasType.ADVANCED;

addChild (champ);

var chargeurCss:URLLoader = new URLLoader (new URLRequest ("css/styles.css"));
chargeurCss.addEventListener (Event.COMPLETE, cssCharge);
function cssCharge (e:Event):void
{
    var styles:StyleSheet = new StyleSheet ();
    styles.parseCSS(chargeurCss.data);

    champ.styleSheet = styles;
}
```

```
h3 {
    font-size:18px;
    font-family:Comic Sans MS;
    color:#000000;
}
```

12 Le son

12.1 Les classes Sound et SoundChannel

Ces classes nous permettent de manipuler du son en AS3.

La classe "**Sound**" nous permet principalement de :

- lancer le chargement d'un son (méthode "**load**"),
- d'en lancer la lecture (méthode "**play**")
- de fermer le flux de chargement (méthode "**close**").

Sound — Propriétés | Méthodes | Evénements

Propriétés publiques

Afficher les propriétés publiques héritées

Propriété	Défini par
bytesLoaded : uint [] Renvoie le nombre d'octets actuellement disponibles dans cet objet Sound.	Sound
bytesTotal : int [] Renvoie le nombre total d'octets que contient l'objet Sound.	Sound
id3 : ID3Info [] Donne accès aux métadonnées faisant partie d'un fichier MP3.	Sound
isBuffering : Boolean [] Renvoie l'état de mise en mémoire tampon des fichiers MP3 externes.	Sound
length : Number [] Durée du son actuel, en millisecondes.	Sound
url : String [] URL à partir de laquelle le son a été chargé.	Sound

Méthodes publiques

Afficher les méthodes publiques héritées

Méthode	Défini par
Sound(stream:URLRequest = null, context:SoundLoaderContext = null) Crée un objet Sound.	Sound
close():void Ferme le flux, ce qui entraîne l'arrêt du téléchargement des données.	Sound
extract(target:ByteArray, length:Number, startPosition:Number = -1):Number Extrait les données audio brutes d'un objet Sound.	Sound
load(stream:URLRequest, context:SoundLoaderContext = null):void Lance le chargement d'un fichier MP3 externe à partir de l'URL spécifiée.	Sound
play(startTime:Number = 0, loops:int = 0, sndTransform:SoundTransform = null):SoundChannel Crée un objet SoundChannel pour lire le son.	Sound

Evénements

Masquer les événements hérités

Evénement	Synthèse	Défini par
activate	[Evénement de diffusion] Distribué lorsque Flash Player obtient le focus du système d'exploitation et devient actif.	EventDispatcher
complete	Distribué lorsque le chargement de données aboutit.	Sound
deactivate	[Evénement de diffusion] Distribué lorsque Flash Player perd le focus du système d'exploitation et devient inactif.	EventDispatcher
id3	Distribué par un objet Sound lorsque des données ID3 sont disponibles pour un son MP3.	Sound
ioError	Distribué lorsqu'il se produit une erreur d'entrée/sortie entraînant l'échec d'un chargement.	Sound
open	Distribué lors du démarrage d'une opération de chargement.	Sound
progress	Distribué lors de la réception de données, au fur et à mesure d'un chargement.	Sound
sampleData	Distribué lorsque le lecteur demande de nouvelles données audio.	Sound

La classe "**SoundChannel**" va essentiellement s'occuper de la lecture de son.

12.2 Lecture d'un son (intégré ou externe)

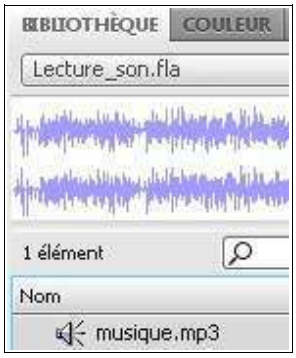
Il existe deux méthodes pour la lecture d'un son :

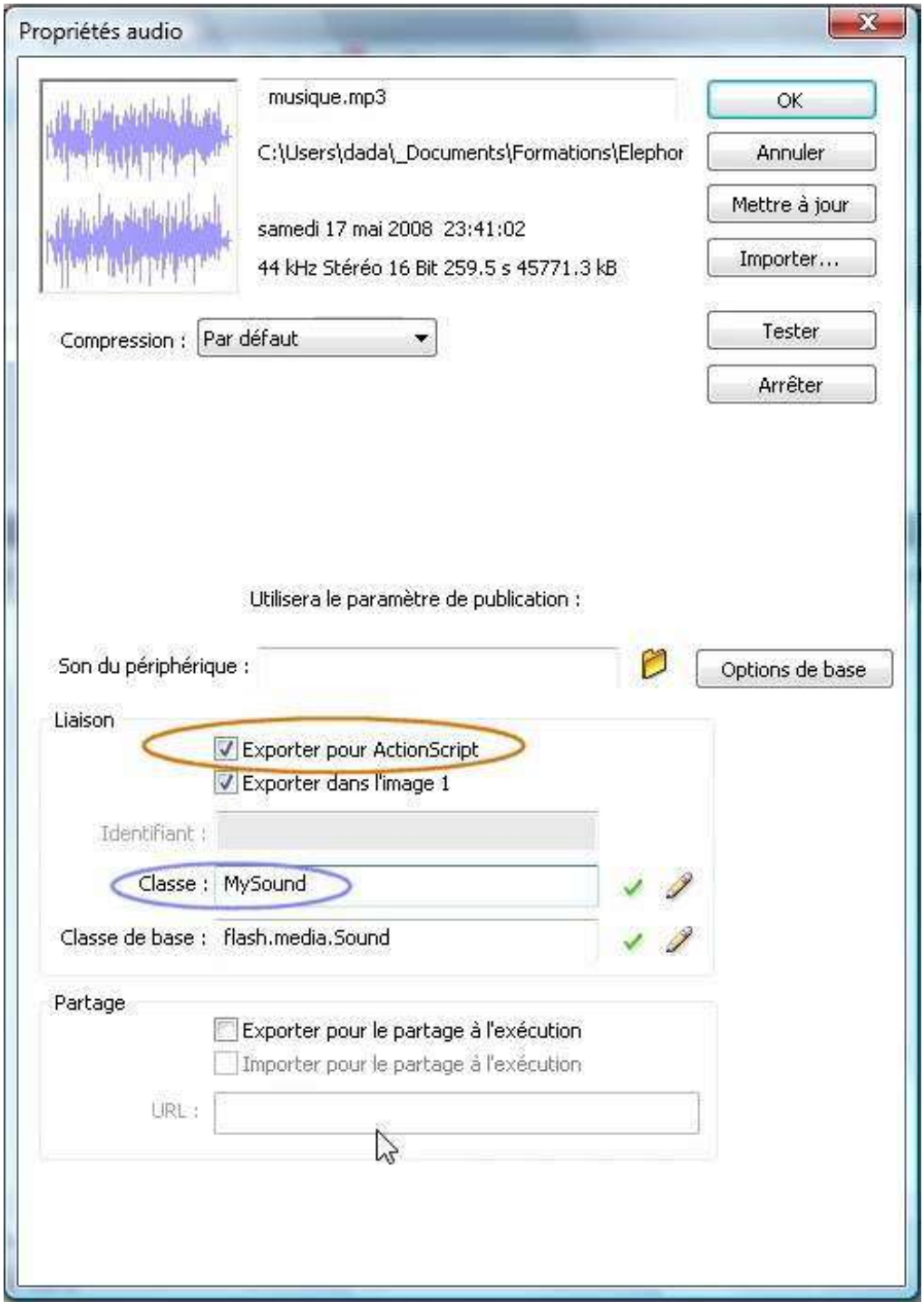
- soit par un son en bibliothèque, réservé aux sons légers (événements).
- soit par un son externe, réservé aux sons longs (musiques).

12.2.1 Son en bibliothèque

Modifier les propriétés du son en bibliothèque :

- Cocher "**Exporter pour ActionScript**".
- Lui donner un nom de classe.





```
var mySound:Sound = new MySound ();
mySound.play ();
```

12.2.2 Son externe

```
// Occurence musique de la classe Sound
var musique:Sound = new Sound ();

// Chargement du fichier
musique.load (new URLRequest("mp3/musique.mp3"));

// Lecture de musique
musique.play();
```

Nous pouvons lancer la musique même si le chargement n'est pas terminé.

Il s'agit d'un chargement progressif, la lecture commencera dès que suffisamment de données sont disponibles. Le chargement continuera en parallèle de la lecture de la musique et l'utilisateur ne s'en rendra pas compte à moi que sa connexion Internet soit plus faible que le débit de la musique.

La classe "**Sound**" ne permet pas de charger plusieurs sons d'affilé. Si nous souhaitions en charger un second, nous devrions créer une nouvelle occurrence de la classe "**Sound**".

12.3 Modification du volume et du panoramique

Lorsque nous souhaitons modifier le volume ou le panoramique (balance) d'un son, nous disposons sur la classe "**SoundChannel**" d'une propriété "**soundTransform**".

SoundChannel — Propriétés | Méthodes | Evénements

soundTransform propriété

soundTransform:SoundTransform []

Version du langage: ActionScript 3.0
: AIR 1.0, Flash Player 9

Objet SoundTransform affecté au canal audio. Un objet SoundTransform comprend les propriétés de réglage du volume, du déplacement panoramique, ainsi que des haut-parleurs gauche et droit.

Cette propriété attend une occurrence de la classe "**SoundTransform**" qui contient des propriétés nous permettant de manipuler volume et panoramique.

SoundTransform — Propriétés | Méthodes | Evénements

Propriétés publiques

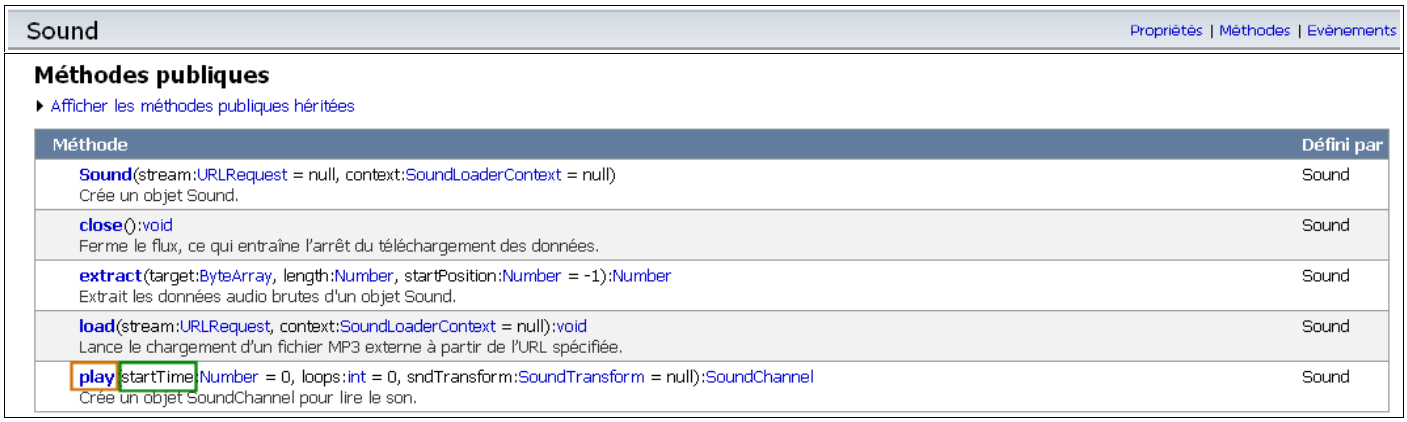
Afficher les propriétés publiques héritées

Propriété	Défini par
leftToLeft : Number Valeur, comprise entre 0 (aucun) et 1 (maximum), indiquant la quantité d'entrée gauche à émettre dans le haut-parleur gauche.	SoundTransform
leftToRight : Number Valeur, comprise entre 0 (aucun) et 1 (maximum), indiquant la quantité d'entrée gauche à émettre dans le haut-parleur droit.	SoundTransform
pan : Number Balance horizontale du son, comprise entre -1 (balance à gauche) et 1 (balance à droite).	SoundTransform
rightToLeft : Number Valeur, comprise entre 0 (aucun) et 1 (maximum), indiquant la quantité d'entrée droite à émettre dans le haut-parleur gauche.	SoundTransform
rightToRight : Number Valeur, comprise entre 0 (aucun) et 1 (maximum), indiquant la quantité d'entrée droite à émettre dans le haut-parleur droit.	SoundTransform
volume : Number Volume, compris entre 0 (muet) et 1 (volume maximal).	SoundTransform

```actionscript
// Occurence musique de la classe Sound
var musique:Sound = new Sound ();

// Chargement du fichier
musique.load (new URLRequest("mp3/musique.mp3"));

// Lecture de musique et récupération du canal audio dans canal
var canal:SoundChannel = musique.play ();

// Occurence transformationAudio de SoundTransform
var transformationAudio:SoundTransform = new SoundTransform ();

// Paramètres transformation audio
// volume à 50 %
transformationAudio.volume = 0.5;
// son tout à gauche
transformationAudio.pan = -1;

// Application de la transformation audio à canal
canal.soundTransform = transformationAudio;
```

12.4 Déplacement dans un son

Lorsque nous souhaitons nous déplacer dans un son, nous disposons via la méthode "**play**" d'un paramètre "**startTime**". Son unité est en milliseconde, et il nous permet de nous déplacer dans la lecture d'un son.



Sound — Propriétés | Méthodes | Evénements

Méthodes publiques

▸ Afficher les méthodes publiques héritées

Méthode	Défini par
Sound(stream:URLRequest = null, context:SoundLoaderContext = null) Crée un objet Sound.	Sound
close():void Ferme le flux, ce qui entraîne l'arrêt du téléchargement des données.	Sound
extract(target:ByteArray, length:Number, startPosition:Number = -1):Number Extrait les données audio brutes d'un objet Sound.	Sound
load(stream:URLRequest, context:SoundLoaderContext = null):void Lance le chargement d'un fichier MP3 externe à partir de l'URL spécifiée.	Sound
play(startTime:Number = 0, loops:int = 0, sndTransform:SoundTransform = null):SoundChannel Crée un objet SoundChannel pour lire le son.	Sound

Mais à chaque fois que nous lançons la méthode "**play**" sur un son, il commence une nouvelle lecture sur un nouveau canal, mais rien ne stoppe la lecture précédente dans le précédent canal.

Si par exemple nous lançons une musique et que nous souhaitons, via un bouton, nous déplacer sur la 1ère minute du son :

```
// Occurence musique de la classe Sound
var musique:Sound = new Sound ();

// Chargement du fichier
musique.load (new URLRequest("mp3/musique.mp3"));

// Lecture de musique et récupération du canal audio dans canal
var canal:SoundChannel = lireSon (musique);

// Occurence transformationAudio de SoundTransform
var transformationAudio:SoundTransform = new SoundTransform ();

// Paramètres transformation audio
// volume à 50 %
transformationAudio.volume = 0.5;
// son tout à gauche
transformationAudio.pan = -1;

// Application de la transformation audio à canal
canal.soundTransform = transformationAudio;

// Écouteur d'événement : Click sur le clip
clip.addEventListener (MouseEvent.CLICK, lireSon1Minute);

// Fonction : Lecture de son pendant à partir de la 1ère minute
function lireSon1Minute (e:MouseEvent):void
{
    //musique.play (60000);
    lireSon (musique, 60000, 0, canal);
}

// Fonction : Lecture de son
function lireSon (pSon:Sound, pTempsDebutMs:Number = 0, pBoucle:int = 0,
                  pAncienCanal:SoundChannel = null):SoundChannel
{
    var transformAudio:SoundTransform;
    // Stoppe la lecture de l'ancien canal s'il existe
    if (pAncienCanal != null)
    {
        pAncienCanal.stop ();
        transformAudio = pAncienCanal.soundTransform;
    }
    // Transmettre l'objet SoundTransform au nouveau canal
    return pSon.play (pTempsDebutMs, pBoucle, transformAudio);
}
```

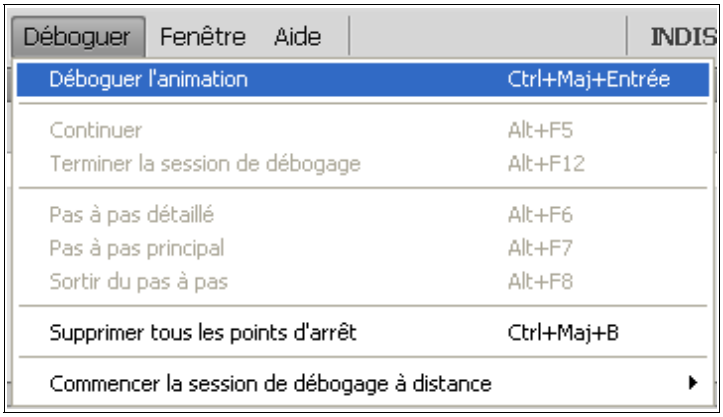
13 Le débogueur de Flash

Le débogueur de Flash nous permet d'exécuter un code pas à pas en maitrisant l'avancement instruction par instruction.

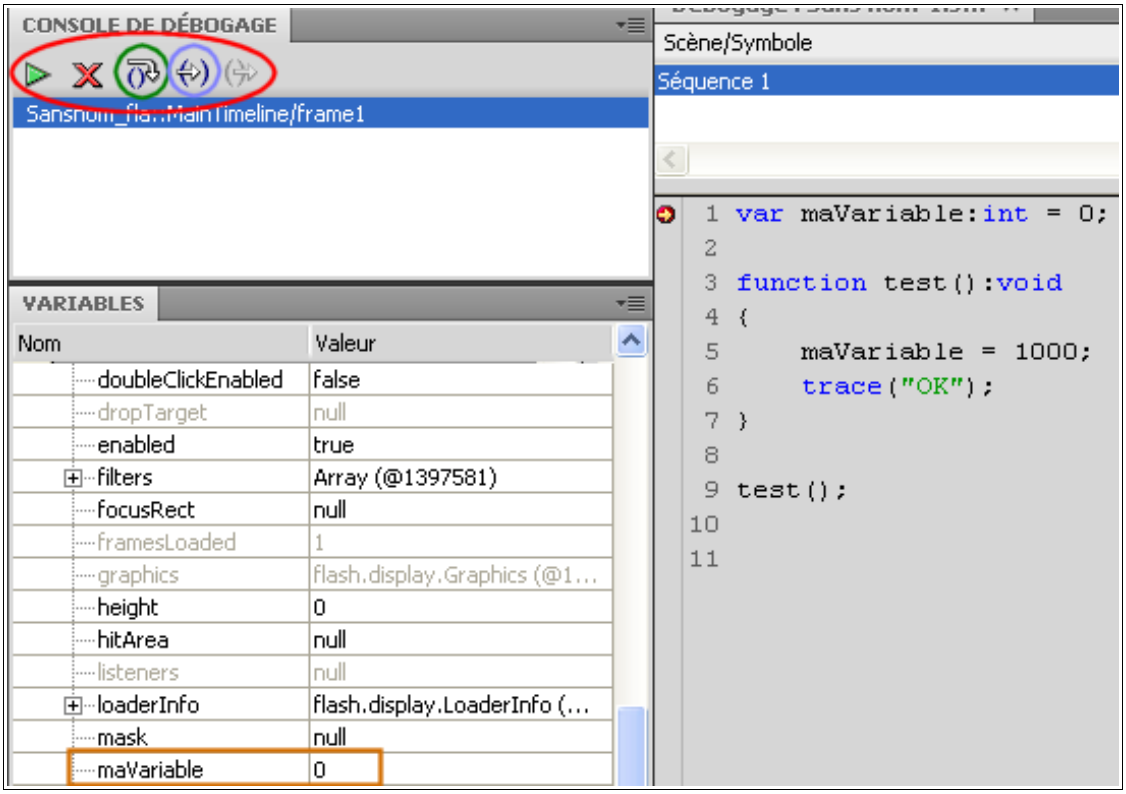
Pour utiliser le débogueur, nous devons déjà ajouter un point d'arrêt (par un clic devant le numéro de ligne).

```
1  var maVariable:int = 0;
2
3  function test():void
4  {
5      maVariable = 1000;
6      trace("OK");
7  }
8
9  test();
```

Nous utilisons ensuite le raccourci clavier "**Ctrl+Shift+Entrée**" ou la menu "**Déboguer / Déboguer l'animation**".



Après un temps d'initialisation, nous avons à l'affichage les différentes propriétés initialisées sur this.



On retrouve : **maVariable** qui vaut **0**.

Nous avons en haut à gauche un contrôleur, par lequel nous pouvons faire du pas à pas principal ou du pas à pas détaillé.

Commençons par le pas à pas principal :

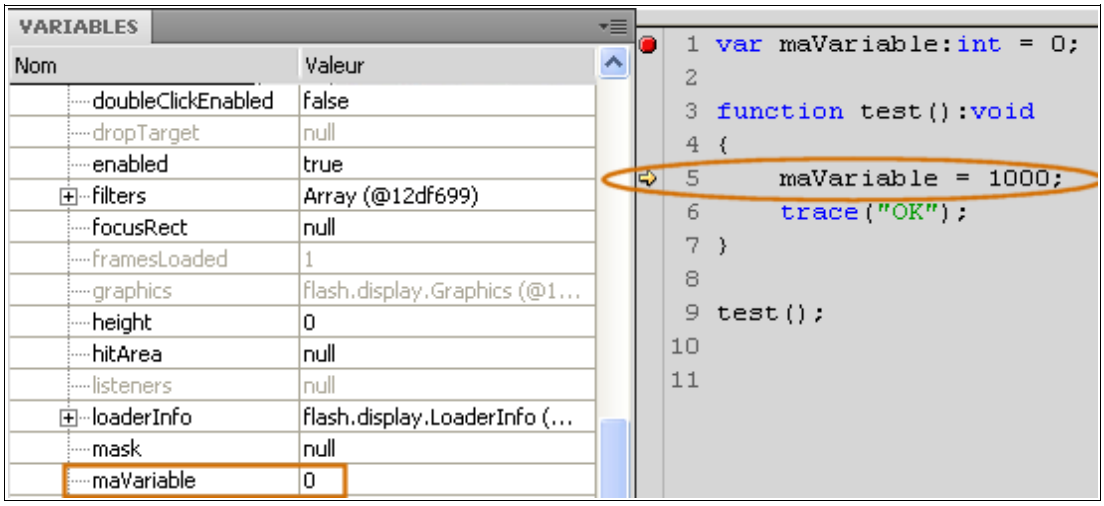
```
1 var maVariable:int = 0;
2
3 function test():void
4 {
5     maVariable = 1000;
6     trace("OK");
7 }
8
9 test();
```

La petite flèche qui était, avant que je clique, sur la première ligne m'indique que la fonction "test" va s'exécuter au prochain pas.

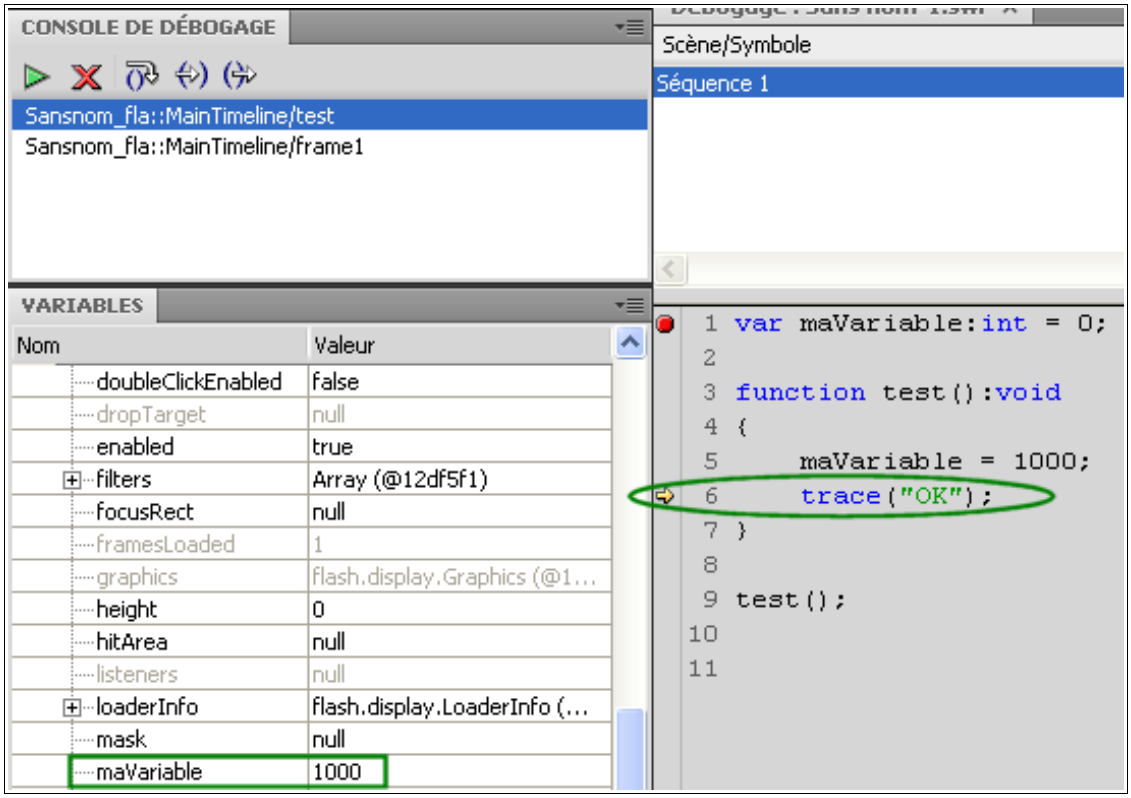
- Si je clique de nouveau sur "**pas à pas principal**" je vais effectuer la fonction d'un bloc.

- Si je souhaite rentrer en pas à pas dans ma fonction, je vais cliquer sur "**pas à pas détaillé**"

Nous arrivons sur la ligne :**maVariable = 1000** pour l'instant elle vaut encore **0**.



Nous avons d'un pas, **maVariable** vaut maintenant **1000** et l'instruction contenant est prête à être exécutée.



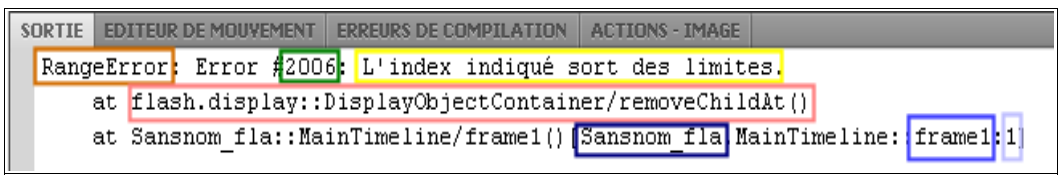
Pour quitter le pas à pas, cliquer sur "**sortir du pas à pas**".

Pour quitter le débogueur, cliquer sur la **croix rouge**.

14 Gestion des erreurs à l'exécution

14.1 Identification des erreurs

```
removeChildAt (0);

trace ("OK");
```

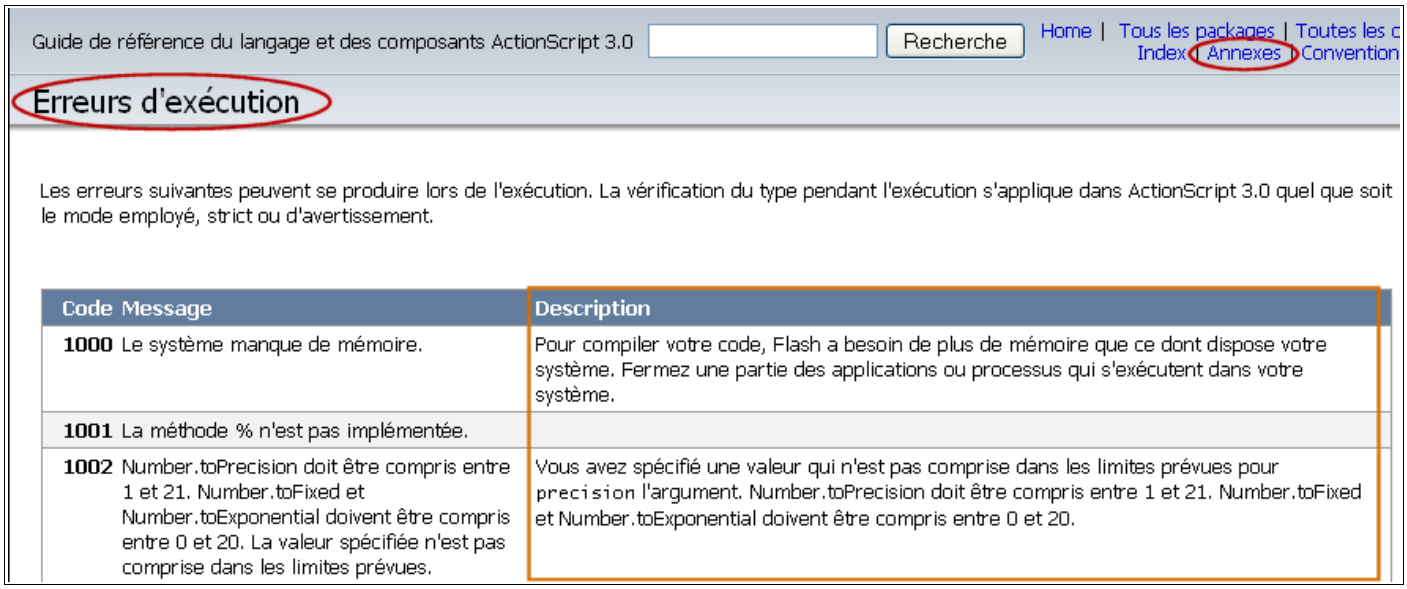


Le message d'erreur nous donne :

- Le nom de la classe d'erreur.
- Un code d'erreur.
- Une description de l'erreur.
- L'objet et la fonction qui a déclenché l'erreur.
- La position de l'erreur (fichier, image, ligne), à condition d'avoir coché "**Autoriser le débogage**" comme indiqué au paragraphe 1.3 "Paramètres de publication".

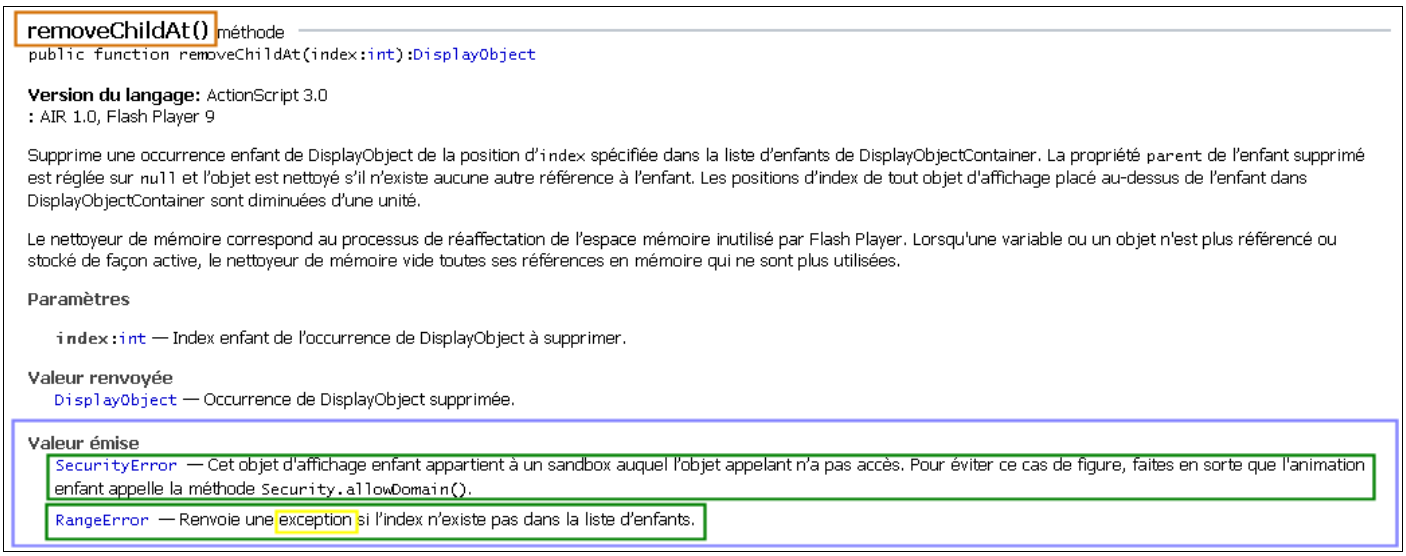
La documentation nous apporte, dans certains cas, des renseignements supplémentaires sur l'erreur dans la colonne "**Description**".

Les codes d'erreur sont accessibles par "**Annexes / Erreurs d'exécution**" :



14.2 Prévoir les risques d'erreur grâce à la documentation

La partie "Valeur émise" de la documentation de la méthode (ici, "**removeChildAt**"), nous indique les risques d'erreur (appelé également exception) lorsque nous utilisons cette méthode :



Nous allons pouvoir faire en sorte que notre code réagisse correctement même en cas d'erreur.

14.3 Gérer les erreurs : utilisation de blocs try / catch

Sachant que cette méthode peut déclencher une erreur (s'il n'y a pas d'enfant sur l'index indiqué), comment aurai-je pu faire pour l'intercepter et pour faire en sorte que mon code s'exécute de manière normale malgré cette erreur ?

Avant de répondre à cette question, il faut savoir que lorsque une erreur se déclenche dans un bloc de code, le code cesse son exécution. Dans notre cas, nous ne voyons pas le trace OK.

14.3.1 Gérer une erreur

Nous avons vu dans la documentation, que la méthode "**removeChildAt**" peut renvoyer une erreur de type "**RangeError**", nous allons donc gérer cette erreur en utilisant un bloc "**try / catch**" :

```
try {
    removeChildAt(0);
}
catch (err:RangeError)
{
    trace ("Erreur gérée");
}
trace ("OK");
```

Le code s'exécute alors normalement :

```
SORTIE | EDITEUR DE M
Erreur gérée
OK
```

14.3.2 Gérer plusieurs erreurs

addChildAt() méthode
public function addChildAt(child:DisplayObject, index:int):DisplayObject

Valeur émise
RangeError — Renvoie une exception si la position d'index n'existe pas dans la liste d'enfants.
ArgumentError — Renvoie une exception si l'enfant et le parent sont identiques. Renvoie également une exception si l'appelant est un enfant (ou petit-enfant, etc.) de l'enfant qui est en cours d'ajout.

Cette méthode peut renvoyer 2 exceptions "**RangeError**" et "**ArgumentError**".

Nous allons créer 2 blocs catch pour gérer ces 2 types d'erreur :

```
addChildAt(clip, 10);

trace ("OK");
```

```
RangeError: Error #2006: L'index indiqué sort des limites.
    at flash.display::DisplayObjectContainer/addChildAt()
    at Sansnom_fla::MainTimeline/frame1()[Sansnom_fla.MainTimeline::frame1:3]
```

```
clip.addChildAt(clip, 0);

trace ("OK");
```

```
ArgumentError: Error #2024: Impossible d'ajouter un objet en tant que son propre enfant.
    at flash.display::DisplayObjectContainer/addChildAt()
    at Sansnom_fla::MainTimeline/frame1()[Sansnom_fla.MainTimeline::frame1:3]
```

```
try {
    addChildAt(clip, 10);
    //clip.addChildAt(clip, 0);
}
catch (err:RangeError)
{
    trace ("Erreur RangeError gérée");
}
catch (err:ArgumentError)
{
    trace ("Erreur ArgumentError gérée");
}
trace ("OK");
```

Si nous ne souhaitons ne pas faire de distinction entre les deux erreurs mais que nous souhaitons les traiter de la même manière, nous utiliserons alors qu'un seul bloc catch en utilisant la classe de base "**Error**":

```
try {
    addChildAt(clip, 10);
    //clip.addChildAt(clip, 0);
}
catch (err:Error)
{
    trace ("Erreur gérée");
}
trace ("OK");
```

Admettons que le code que nous mettons dans le bloc catch déclenche lui aussi une erreur.

Que se passe t-il alors ?

La première erreur a bien été gérée, la deuxième (celle du bloc catch) n'a pas été gérée. De plus le trace OK ne s'est pas exécuté.

Si nous souhaitons qu'un code s'exécute dans tous les cas, même si une erreur a été déclenchée dans le bloc catch, nous pouvons terminer l'ensemble des blocs par un bloc "**finaly**".

Le code dans "**finally**" s'exécutera quoi qu'il arrive, mais l'erreur n'est pas gérée et nous aurions du ajouter (imbriquer) un autre bloc "try / catch".

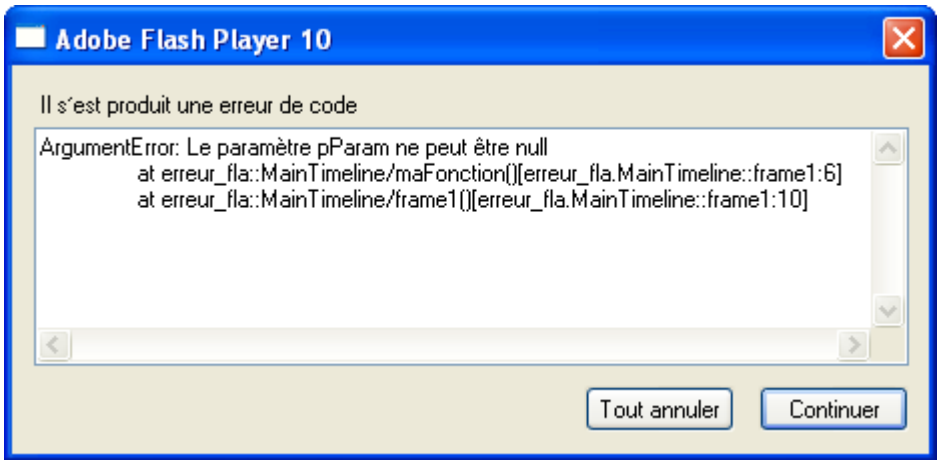
```
try {
    addChildAt(clip, 10);
}
catch (err:Error)
{
    trace ("Erreur gérée");
    clip.addChildAt(clip, 0);
}
finally
{
    trace ("OK");
}
```

Il est préférable, de traiter les erreurs avec une condition "**if**" au lieu d'utiliser les blocs "**try / catch**". De plus cette méthode est plus rapide (en performance).

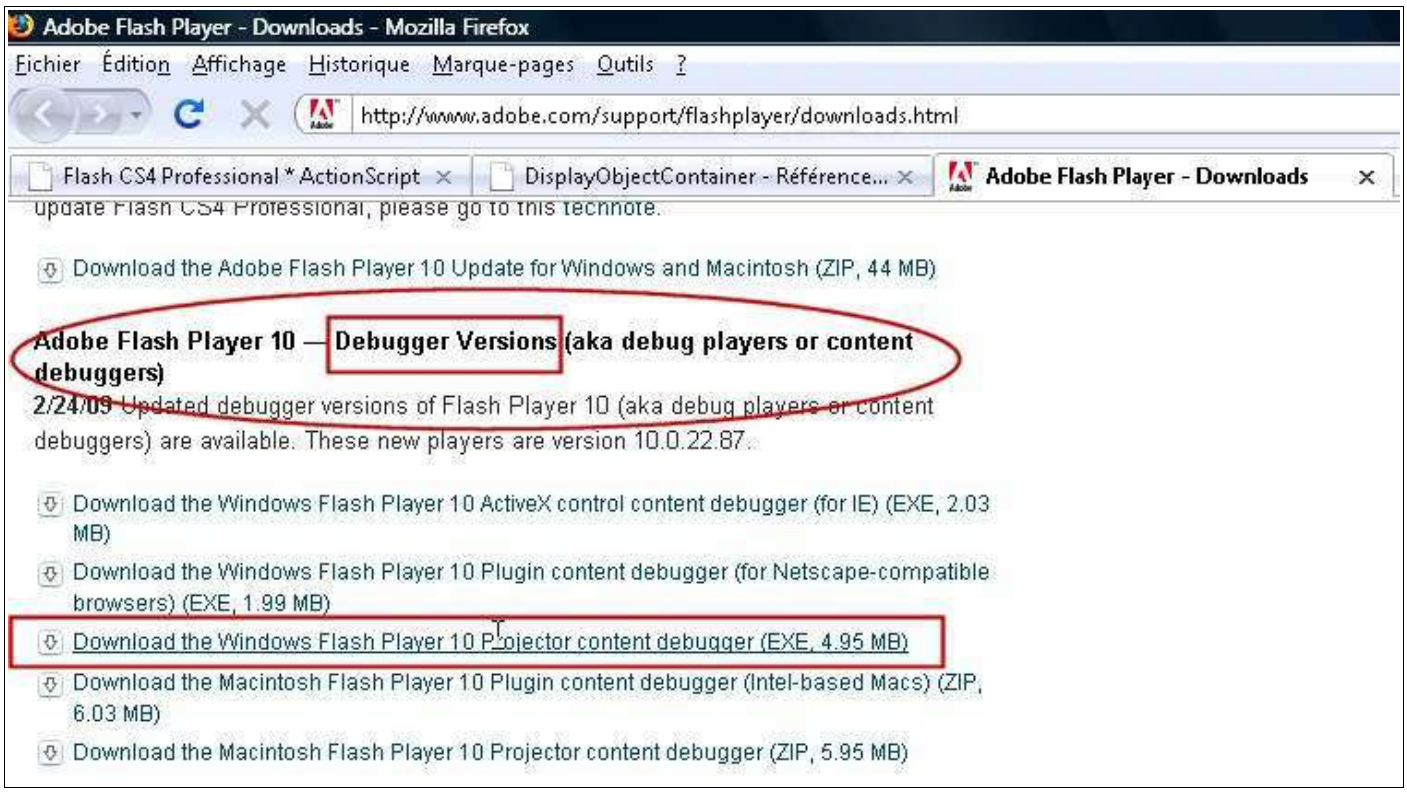
14.4 Lancer des erreurs

Il est possible de générer nous même des erreurs selon certaines condition dans notre code qui en justifierais l'emploi.

Si nous testons ce code dans un navigateur et que nous disposons du "**Flash Player Debugger**", nous obtiendrons ce popup d'erreur :



Nous trouvons les **versions "Debugger"** des Flash Player sur Internet, sur le site Adobe :



15 Gestion de la mémoire

15.1 Garbage Collector

Le "**Garbage Collector**" est le système chargé de gérer les ressources mémoire dans le Flash Player.

Son rôle est de supprimer les objets dont nous ne nous servons plus dans l'application.

- Lorsque nous créons des objets, nous les référençons dans des "**variables**".
- Le "**Garbage Collector**" part du principe que si un objet en mémoire n'est plus référencé dans l'application, c'est que celle-ci n'en a plus besoin. Il supprime alors définitivement en mémoire cet objet.
- Si je souhaite qu'un objet soit supprimé de la mémoire, je dois supprimer la référence de la variable vers cet objet (en affectant la valeur "**null**" à la variable).

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

sprite.addEventListener(Event.ENTER_FRAME, entree)
function entree (e:Event):void
{
    // affichage à chaque frame
    trace("Nouvelle Frame");
}

    // suppression de la référence vers l'objet
    sprite = null;
```

Attention :

- Il faut savoir que le "**Garbage Collector**" vérifie nos références à intervalles plus ou moins réguliers.
- Dans une application qui utilise très peu de mémoire, il ne passera pas forcément pour nettoyer la mémoire ou il passera un peu plus tard (lorsque nous créerons de nouveaux objets) !

Nous avons sur la classe "**System**" une méthode qui se nomme "**gc**", qui permet de forcer le "**Garbage Collector**" à vérifier nos références. Elle n'est utilisable que par le **Flash Payer Debug** (donc seulement en phase de test et non en production) !

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

sprite.addEventListener(Event.ENTER_FRAME, entree)
function entree (e:Event):void
{
    // affichage à chaque frame
    trace("Nouvelle Frame");
}

    // suppression de la référence vers l'objet
    sprite = null;

    // Permet de forcer le Garbage Collector.
    // Attention, utilisable uniquement avec le Flash Player Debug
    // donc seulement en test mais pas en production !!!
    System.gc();
```

15.2 Utilisation mémoire

Si nous souhaitons suivre l'utilisation mémoire de Flash Player, nous pouvons utiliser la propriété "**totalMemory**" de la classe "**System**". Elle renvoie l'utilisation mémoire du Flash Player en octets.

```
// Utilisation mémoire en octets
trace (System.totalMemory + " octets");

// Utilisation mémoire en Méga octets à deux chiffres après la virgule
trace ((System.totalMemory / 1048576).toFixed(2)+ " méga-octets");
```

Attention : lorsque nous utilisons ce code dans un navigateur à onglets, il peut arriver que "**totalMemory**" renvoi le nombre d'octets utilisés par l'ensemble des "Flash Player" ouverts dans les différents onglets.

16 Exemples d'applications

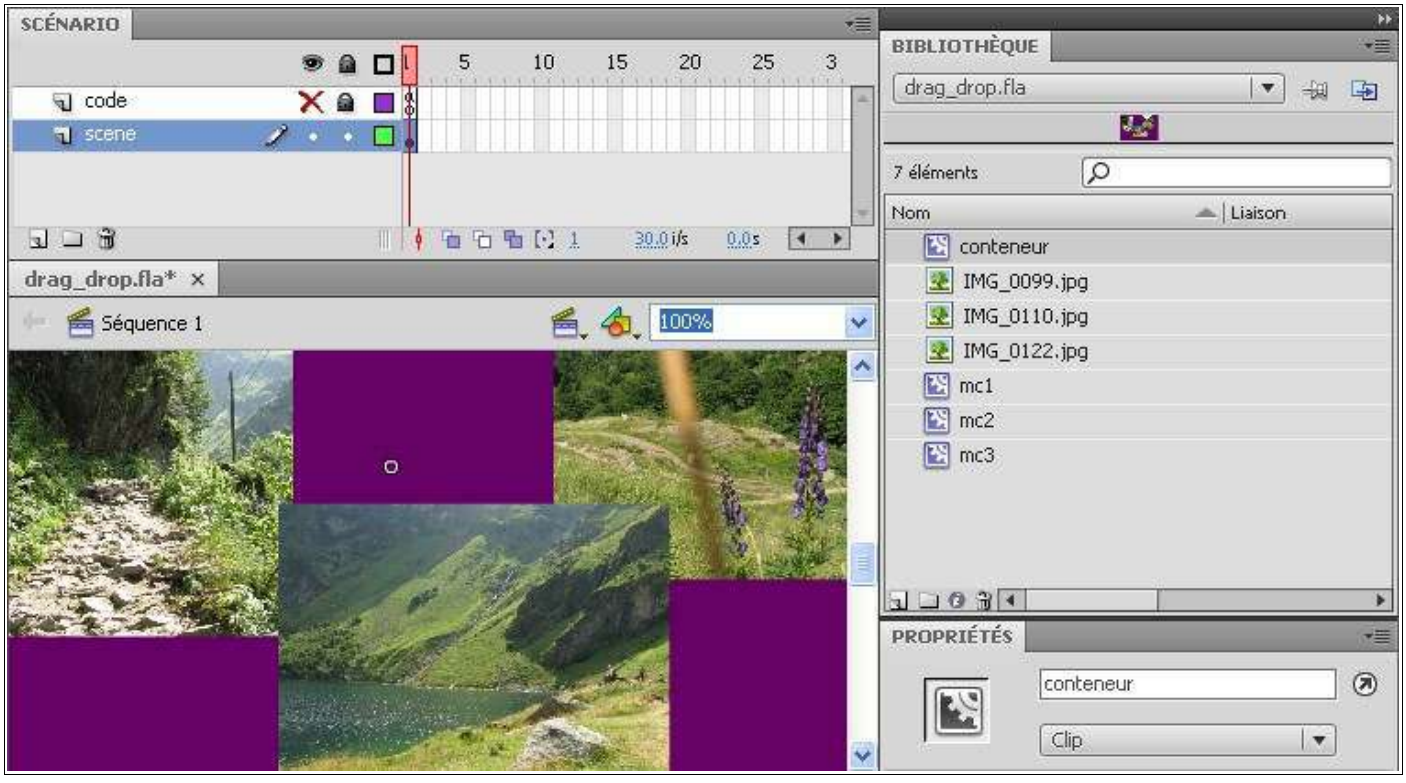
16.1 Drag and Drop

Nous avons sur la scène :

- Un clip conteneur dans lequel se trouvent
- 3 clips contenant chacun une image.
- 1 fond vectoriel.

Déroulement :

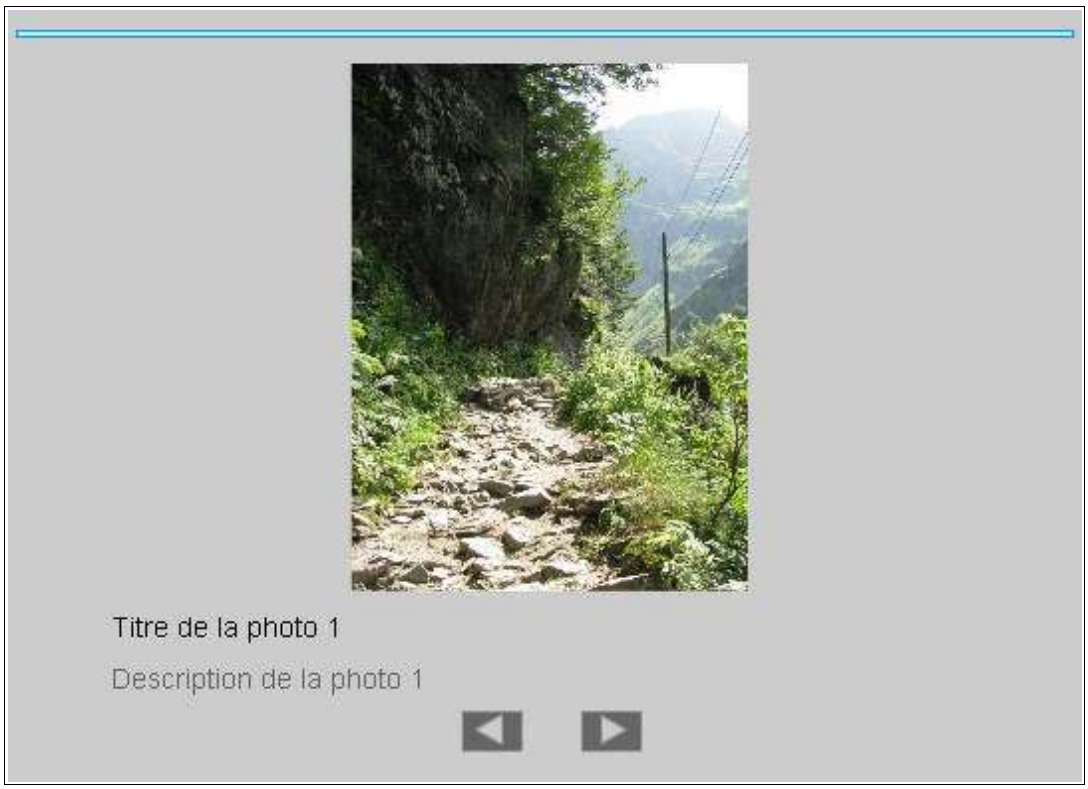
- Le curseur devient une main sur un élément déplaçable.
- Les 3 images sont déplaçables avec la souris et passent alors au premier plan.
- Le fond n'est pas déplaçable.



16.1.1 Événements souris, startDrag et stopDrag, index de profondeur

```
// Rendre le conteneur non interactif donc le fond non déplaçable
conteneur.mouseEnabled = false;

// Curseur en forme de main sur les objets interactifs
conteneur.buttonMode = true;

// Grâce à la propagation en bubbling,
// le conteneur verra passer les événements de ses enfants interactifs.

// Écouteur d'événement : Appui sur le bouton principal de la souris
conteneur.addEventListener ( MouseEvent.MOUSE_DOWN, conteneurDown );

// Écouteur d'événement : Relachement du bouton principal de la souris
conteneur.addEventListener ( MouseEvent.MOUSE_UP, conteneurUp );

// Fonction écouteur appui : conteneurDown
function conteneurDown (e:MouseEvent):void
{
    // Casting de clipTarget (e.target) en tant que MovieClip
    var clipTarget:MovieClip = e.target as MovieClip;

    // Début du Drag de l'objet cliqué
    clipTarget.startDrag ();
    if (e.target != e.currentTarget) // si (clipTarget != conteneur)
    {
        // Mettre l'objet cliqué au premier plan
        conteneur.setChildIndex (clipTarget, conteneur.numChildren - 1);
    }
}

// Fonction écouteur relachement : conteneurUp
function conteneurUp (e:MouseEvent):void
{
    var clipTarget:MovieClip = e.target as MovieClip;

    // Fin du Drag de l'objet cliqué
    clipTarget.stopDrag();
}
```

16.2 Diaporama



Nous avons sur la scène :

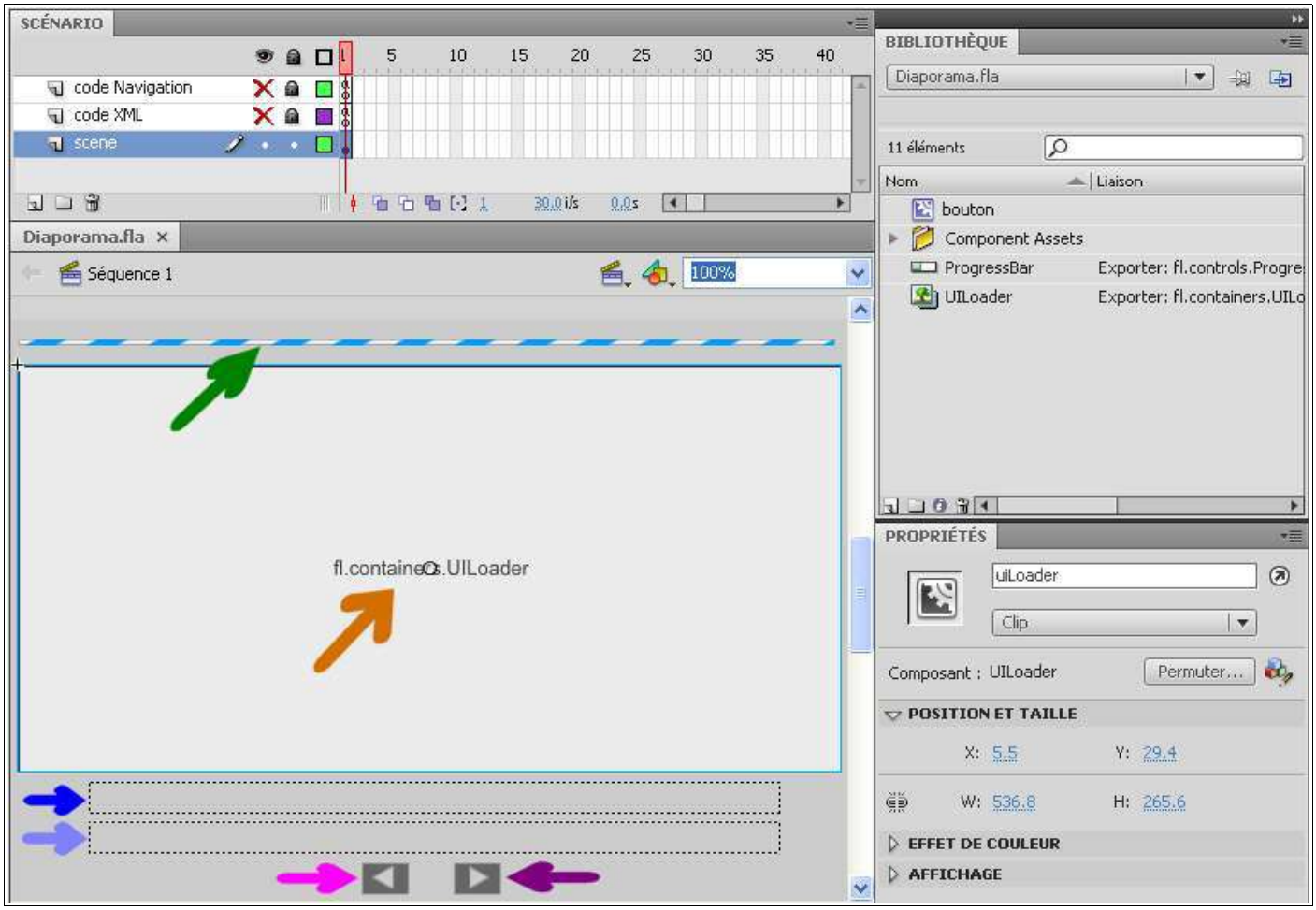
- Affichage de l'image, au centre.
- Une barre de progression du chargement de l'image, en haut.
- Le titre et la description de l'image, sous l'image.
- 2 boutons qui permettent d'avancer ou de reculer dans la série d'images en tournant en boucle.

Organisation :

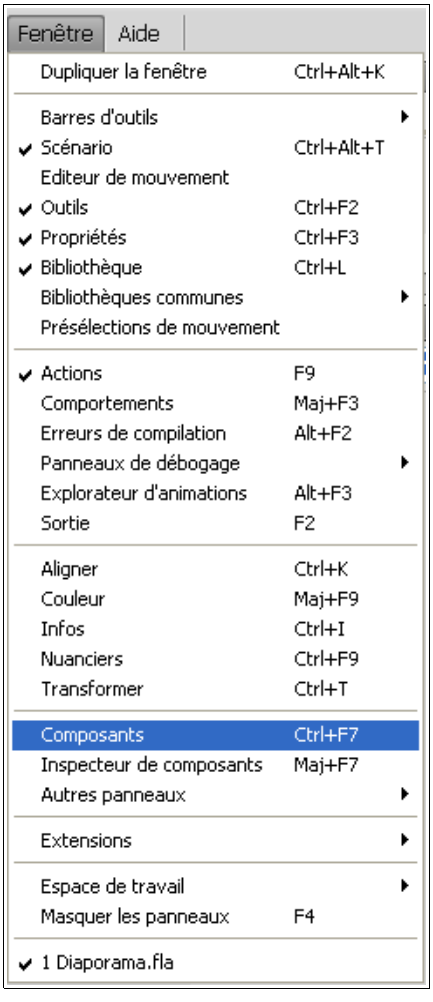
- Les images sont placées dans un dossier "**img**".
- L'application est renseignée par le fichier "**photos.xml**" placé dans le dossier "**xml**".

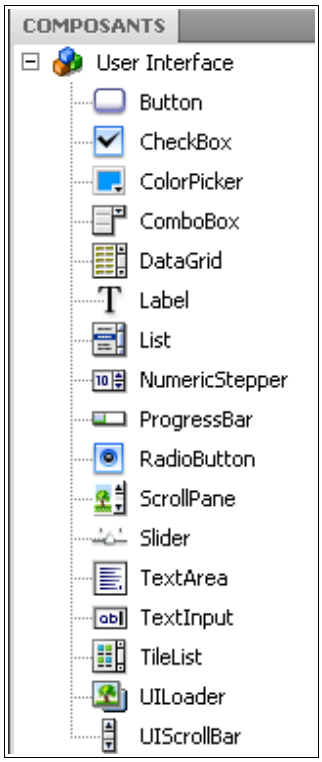
Nous partons d'un document fla qui contient des éléments posés sur la scène :

- 2 champs de texte.
- 2 boutons.
- 2 composants :
  - Un UILoader qui permet de charger des images ou des swf, de les centrer et de les redimensionner automatiquement en fonction du rapport largeur sur hauteur.
  - Un ProgressBar, nous lui indiquerons qu'il doit s'appliquer au chargement des images de l'UILoader.



Nous pouvons accéder aux composants à partir du menu Fenêtre / Composants :





Fichier **photos.xml** :

photos.xml

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

<photos>

    <photo>
        <url>img/photo1.jpg</url>
        <titre>Titre de la photo 1</titre>
        <desc>Description de la photo 1</desc>
    </photo>

    <photo>
        <url>img/photo2.jpg</url>
        <titre>Titre de la photo 2</titre>
        <desc>Description de la photo 2</desc>
    </photo>

    <photo>
        <url>img/photo3.jpg</url>
        <titre>Titre de la photo 3</titre>
        <desc>Description de la photo 3</desc>
    </photo>

</photos>
```

16.2.1 Code Navigation : Object, Array, opérateurs d'incrémentation

```
// progressBar affecté à l'uiLoader
progressBar.source = uiLoader;
// Déclaration du tableau des images
var listeImages:Array = [];
// Initialisation de l'index à 0
var indexImage:int = 0;

// Écouteur d'événement : Click sur le bouton précédent
btPrecedent.addEventListener (MouseEvent.CLICK, precedentClick);
// Fonction écouteur : précedent
function precedentClick (e:MouseEvent):void
{
    if (--indexImage < 0)
    // si index après décrémentation < 0
    {
        indexImage = listeImages.length - 1;
    }
    afficherImage ();
}

// Écouteur d'événement : Click sur le bouton suivant
btSuivant.addEventListener (MouseEvent.CLICK, suivantClick);
// Fonction écouteur : suivant
function suivantClick (e:MouseEvent):void
{
    if (++indexImage > listeImages.length - 1)
    // si index après incrémentation > index maxi
    {
        indexImage = 0;
    }
    afficherImage ();
}

// Fonction : affichage des images
function afficherImage ():void
{
    // Récupération de l'objet correspondant
    // à l'index indexImage dans objImage
    var objImage:Object = listeImages [indexImage];

    // On va utiliser les propriétés que l'on a créees
    // sur l'objet listeImages dans code XML

    // On utilise la propriété titre de objImage
    // pour remplir le champ texte champTitre
    champTitre.text = objImage.titre;

    // On utilise la propriété desc de objImage
    // pour remplir le champ texte champDesc
    champDesc.text = objImage.desc;

    // On utilise la propriété url de objImage
    // pour renseigner uiLoader et charger l'image
    uiLoader.load ( new URLRequest(objImage.url) );
}
```

Différents opérateurs d'incrémentation (+) ou de décrémentation (-) :

- maVariable = maVariable + 10 ☛ incrémentation de 10
- maVariable += 10 ☛ incrémentation de 10
- maVariable ++ ☛ incrémentation de 1
- if (++maVariable > 0) ☛ incrémentation de 1 puis test
- if (maVariable++ > 0) ☛ test puis incrémentation de 1

16.2.2 Code XML : Chargement et traitement du fichier, création liste et objet

```
// Occurence  chargeurXML de URLLoader
var chargeurXML:URLLoader = new URLLoader (new URLRequest("xml/photos.xml"));

// Écouteur d'événement : chargement terminé
chargeurXML.addEventListener (Event.COMPLETE, xmlCharge);

// Fonction écouteur : traitement du fichier XML
function xmlCharge (e:Event):void
{
    // Occurence xml de la classe XML remplie avec chargeurXML
    var xml:XML = new XML (chargeurXML.data);

    // Création de xmlListPhoto avec la liste des noeuds photo
    var xmlListPhoto:XMLList = xml.photo;

    // Création de objPhoto qui servira dans la boucle suivante
    var objPhoto:Object;

    // Pour parcourir la liste xmlListPhoto
    for each (var xmlPhoto:XML in xmlListPhoto)
    {
        // Occurence objPhoto de la classe dynamique Object
        objPhoto = new Object ();

        // La classe étant dynamique, on peut créer des propriétés :

        // Création et remplissage de url avec l'élément url du XML
        objPhoto.url = String (xmlPhoto.url);

        // Création et remplissage de titre avec l'élément titre du XML
        objPhoto.titre = String (xmlPhoto.titre);

        // Création et remplissage de desc avec l'élément desc du XML
        objPhoto.desc = String (xmlPhoto.desc);

        // Remplissage du tableau listeImages (crée dans code Navigation)
        // avec l'objet qui vient d'être crée
        listeImages.push (objPhoto);
    }

// Affichage de la 1ère image
afficherImage ();
}
```

Différents traitements d'un fichier XML :

Soit le fichier XML :

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

<photos>

    <photo id="1">
        <url>img/photo1.jpg</url>
        <titre>Titre de la photo 1</titre>
        <desc>Description de la photo 1</desc>
    </photo>

    <photo id="2">
        <url>img/photo2.jpg</url>
        <titre>Titre de la photo 2</titre>
        <desc>Description de la photo 2</desc>
    </photo>

    <photo id="3">
        <url>img/photo3.jpg</url>
        <titre>Titre de la photo 3</titre>
        <desc>Description de la photo 3</desc>
    </photo>

</photos>
```

- Filtre du fichier par le nom d'un élément (ex: "**photo**") :

```actionscript
// Occurence  chargeurXML de URLLoader
var chargeurXML:URLLoader = new URLLoader (new URLRequest("xml/photos.xml"));

// Écouteur d'événement : chargement terminé
chargeurXML.addEventListener (Event.COMPLETE, xmlCharge);

// Fonction écouteur : traitement du fichier XML
function xmlCharge (e:Event):void
{
    // Occurence xml de la classe XML remplie avec chargeurXML
    var xml:XML = new XML (chargeurXML.data);

    trace (xml.photo);
}
```

```xml
<photo id="1">
  <url>img/photo1.jpg</url>
  <titre>Titre de la photo 1</titre>
  <desc>Description de la photo 1</desc>
</photo>
<photo id="2">
  <url>img/photo2.jpg</url>
  <titre>Titre de la photo 2</titre>
  <desc>Description de la photo 2</desc>
</photo>
<photo id="3">
  <url>img/photo3.jpg</url>
  <titre>Titre de la photo 3</titre>
  <desc>Description de la photo 3</desc>
</photo>
```

– Recherche sur la valeur d'un élément (url == …)

Pour appliquer une condition, il faut utiliser les parenthèses :

```
trace (xml.photo.(url == "img/photo1.jpg"));
```

```
<photo id="1">
  <url>img/photo1.jpg</url>
  <titre>Titre de la photo 1</titre>
  <desc>Description de la photo 1</desc>
</photo>
```

– Recherche sur l'attribut d'un élément (@id="1")

Pour appliquer une condition sur l'attribut, il faut utiliser faire précéder le nom de l'attribut de @ :

```
trace (xml.photo.(@id == "1"));
```

```
<photo id="1">
  <url>img/photo1.jpg</url>
  <titre>Titre de la photo 1</titre>
  <desc>Description de la photo 1</desc>
</photo>
```

– Filtre sur l'élément d'un élément (titre) :

```
trace (xml.photo.(@id == "1").titre);
```

```
Titre de la photo 1
```

– Code HTML dans le fichier XML :

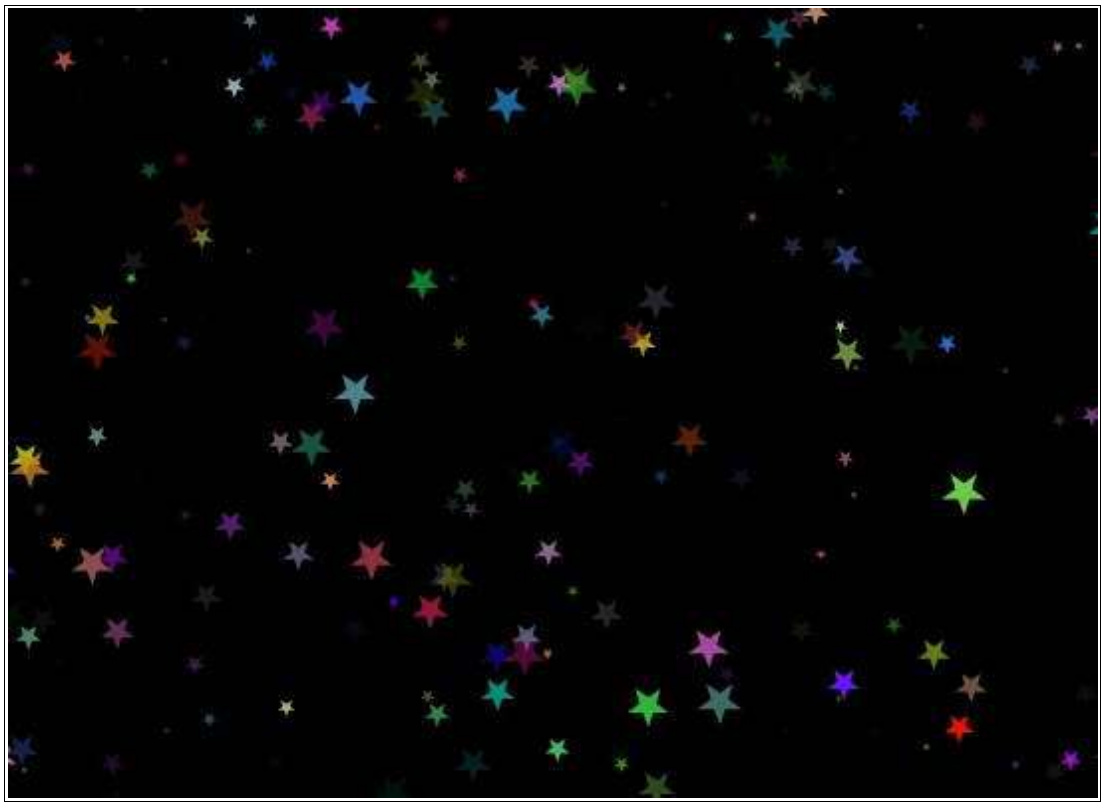
Si nous souhaitons utiliser du code HTML dans les éléments, nous devons encadrer le contenu de l'élément par une balise "**CDATA**" dont la syntaxe est : ![CDATA[…]]

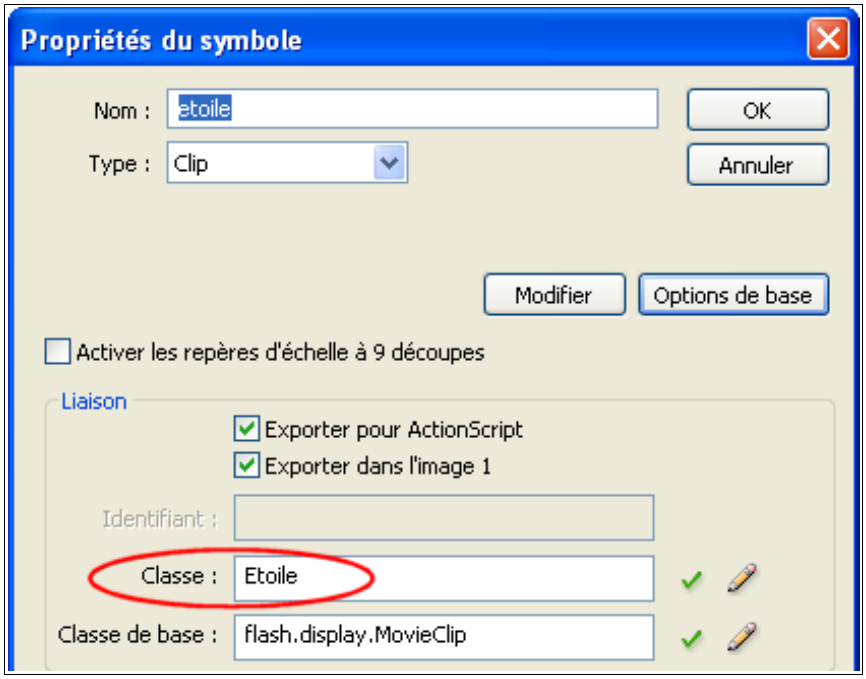
```
<photo id="1">
    <url>img/photo1.jpg</url>
    <titre><![CDATA[Titre de la <b>photo 1</b>]]></titre>
    <desc>Description de la photo 1</desc>
</photo>
```

16.3 Générateur de particules

16.3.1 Événement ENTER_FRAME, boucle d'animation, collection d'objets

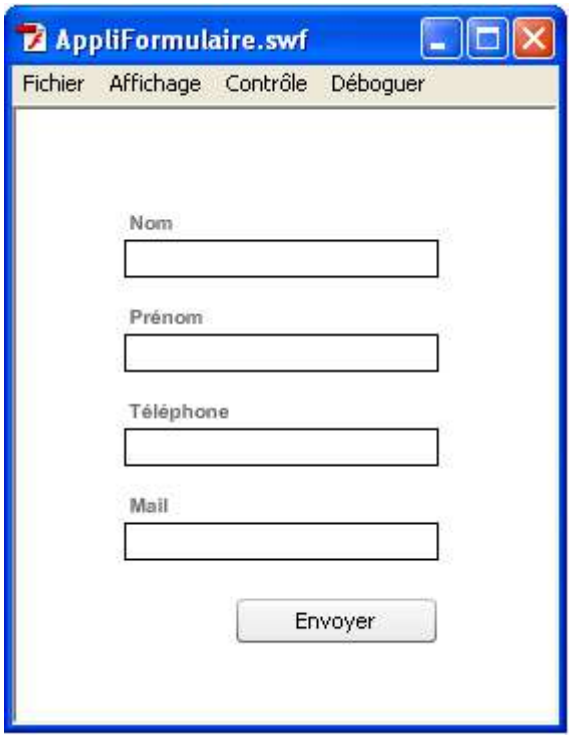
16.3.2 comportement aléatoire, classe dynamique, mémoire cache





```
// Dimensions et positions fixes des objets
stage.scaleMode = StageScaleMode.NO_SCALE;
// Alignement en haut à gauche
stage.align = StageAlign.TOP_LEFT;
// Déclaration variable etoile en MovieClip
var etoile:MovieClip;
// Tableau de référencement de chaque étoile
var listeEtoiles:Array = new Array ();
// Génération de 200 étoiles
for (var i:int = 0; i<200; i++)
{
    // occurence de la classe Etoile (symbole clip etoile)
    etoile = new Etoile ();
    // Comportement aléatoire avec Math.random()
    // Position
    etoile.x = Math.random () * stage.stageWidth;
    etoile.y = Math.random () * stage.stageHeight;
    // Echelle (taille de l'étoile) entre 0.1 et 1 (jamais à 0)
    etoile.scaleX = etoile.scaleY = 0.1 + Math.random() * 0.9;
    // Couleur
    // On utilise la Classe ColorTransform
    var ct:ColorTransform = new ColorTransform ();
    ct.color = 0xffffff * Math.random();
    etoile.transform.colorTransform = ct;
    // Transparence
    etoile.alpha = Math.random();
    // Vitesse
    // On utilise le caractère dynamique de la Classe MovieClip pour créer une propriété
    // Plus l'étoile est grande, plus elle se déplace rapidement
    etoile.vitesseX = 0.5 + 2*Math.random() + 4*etoile.scaleX;
    // Le Flash Player crée une représentation bitmap de l'objet
    // et le place en mémoire cache pour diminuer l'utilisation processeur
    etoile.cacheAsBitmap = true;
    // Remplissage du tableau de référencement
    listeEtoiles.push (etoile);
}
// Tri du tableau pour avoir les étoiles rangées par taille
listeEtoiles.sortOn ("scaleX", Array.NUMERIC);
    // Les étoiles les plus petites seront à l'arrière, les plus grosses à l'avant
    for each (etoile in listeEtoiles)
    {
    // Affichage de chaque étoile
    addChild (etoile);
    }
// Écouteur d'événement : à chaque image (cadence animation, par ex. 30 images/seconde)
addEventListener (Event.ENTER_FRAME, chaqueImage);
// Fonction Écouteur : chaqueImage
function chaqueImage (e:Event):void
{
    for each (etoile in listeEtoiles)
    {
        etoile.x += etoile.vitesseX; // On utilise la propriété vitesse que l'on a créée
        // Gestion des limites de la scène (largeur scène + 1/2 largeur étoile)
        if (etoile.x > stage.stageWidth + etoile.width / 2)
        {
            etoile.x = -etoile.width / 2;
            etoile.y = Math.random () * stage.stageHeight;
        }
    }
}
```

16.4 Formulaire



Attention :

Si nous voulons tester (dans Flash) une application qui utilise des entrées clavier, nous devons cocher "**désactiver les raccourcis clavier**" dans le Flash Player intégré à Flash. Sinon les raccourcis clavier vont être dirigés vers le logiciel Flash plutôt que vers le Flash Player et nous risquerons alors de modifier l'application (ajout image clé, …).

16.4.1 Gestion des ordres de tabulation

```
// Ordres de tabulation
champNom.tabIndex = 1;
champPrenom.tabIndex = 10;
champTelephone.tabIndex = 20;
champMail.tabIndex = 30;

// Suppression du bouton
// de la liste des tabulations
btEnvoyer.tabEnabled = false;
```

16.4.2 Validation des entrées

```
// Restrictions sur le champ Téléphone :
// on ne peut y saisir que des chiffres
champTelephone.restrict = "0-9";
// il est limité à 10 caractères
champTelephone.maxChars = 10;

// Validation des champs
function validerFormulaire ():String
{
    // Pas de message affiché si pas d'erreur de saisie
    var messageRetour:String = "";

    // Schéma de l'expression régulière pour champ Mail
    var regExpMail:RegExp = /[\w.-]+@[\w.-]+\.[a-zA-Z]{2,5}$/;

    if (champNom.text == "")
    // Si champ Nom non rempli (vide)
    {
        messageRetour = "Merci de renseigner votre nom.";
        stage.focus = champNom;
    }
    else if (regExpMail.exec (champMail.text) == null)
    // Si champ Mail mal rempli
    // (le texte ne correspond pas à l'expression régulière)
    {
        messageRetour = "L'adresse mail n'est pas valide.";
        stage.focus = champMail;
    }

    return messageRetour;
}
```

16.4.3 Événements focus

Lorsqu'un champ a le focus, les entrées du clavier sont dirigées sur lui.

Nous voulons que lorsqu'un champ prend le focus, tout son contenu soit sélectionné.

```
// Focus sur le champ Nom au lancement de l'application
stage.focus = champNom;

function champFocusIn (e:FocusEvent):void
{
    // Casting de champTarget en TextField (object par défaut)
    var champTarget:TextField = e.target as TextField;
    // Sélection à partir du caractère 0 sur toute la longueur
    champTarget.setSelection(0, champTarget.length);
}

// Écouteur d'événement : prise de focus, sur chacun des champs
champNom.addEventListener (FocusEvent.FOCUS_IN, champFocusIn);
champPrenom.addEventListener (FocusEvent.FOCUS_IN, champFocusIn);
champTelephone.addEventListener (FocusEvent.FOCUS_IN, champFocusIn);
champMail.addEventListener (FocusEvent.FOCUS_IN, champFocusIn);
```

16.4.4 Événements clavier

Nous voulons que la touche "Entrée" valide un champ texte lorsque celui-ci a le focus.

```
function clavierTouchePressee (e:KeyboardEvent):void
{
    if (e.keyCode == Keyboard.ENTER)
    // si touche "Entrée" pressés
    {
    envoyerFormulaire ();
    }
}
// Écouteur d'événement : touche pressée, sur chacun des champs
champNom.addEventListener (KeyboardEvent.KEY_DOWN, clavierTouchePressee);
champPrenom.addEventListener (KeyboardEvent.KEY_DOWN, clavierTouchePressee);
champTelephone.addEventListener (KeyboardEvent.KEY_DOWN, clavierTouchePressee);
champMail.addEventListener (KeyboardEvent.KEY_DOWN, clavierTouchePressee);
```

16.4.5 Envoyer des variables vers une URL

Pour envoyer les valeurs saisies dans le formulaire vers un script serveur (Php, …), pour par exemple enregistrer ces valeurs dans une base de données.

```
var chargeur:URLLoader = new URLLoader ();

// Écouteur d'événement : Click sur le bouton btEnvoyer
btEnvoyer.addEventListener(MouseEvent.CLICK, envoyerClick);
// Écouteur d'événement : URLLoader Chargement complet
chargeur.addEventListener(Event.COMPLETE, chargeurComplet);
// Écouteur d'événement : URLLoader Chargement erreur
chargeur.addEventListener(IOErrorEvent.IO_ERROR, chargeurErreur);

function envoyerClick (e:MouseEvent):void
{
    envoyerFormulaire ();
}

function chargeurComplet (e:Event):void
{
    champMessage.text = "Envoi réussi.";

    // On pourrait récupérer la variable resultat (par ex)
    // retournée par le script à la fin de son traitement :
    // trace( chargeur.data.resultat);
}

function chargeurErreur (e:Event):void
{
    champMessage.text = "Erreur d'envoi du formulaire.";
}

function envoyerFormulaire ():void
{
    var messageUtilisateur:String = validerFormulaire ();
    if (messageUtilisateur != "")
    {
        champMessage.text = messageUtilisateur;
    }
    else
    {
        champMessage.text = "Envoi en cours...";
        // Adresse du script serveur réceptionnant les données.
        var urlScriptServeur:String = "scripts/form.php";
        // Création de la requete avec les variables
        var requete:URLRequest = new URLRequest (urlScriptServeur);
        requete.method = URLRequestMethod.POST;
        var vars:URLVariables = new URLVariables ();
        vars.nom = champNom.text;
        vars.prenom = champPrenom.text;
        vars.telephone = champTelephone.text;
        vars.mail = champMail.text;
        requete.data = vars;
        chargeur.load (requete);
    }
}
```

16.5 Jeu de grattage

Lorsque nous avons gratté une partie suffisante de l'image, la totalité de l'image se dévoile et nous pouvons afficher un message à l'utilisateur.

16.5.1 Classe BitmapData

Nous allons utiliser la classe BitmapData en tant que masque et dessiner le motif dans l'objet BitmapData. C'est donc l'objet Bitmap qui va servir de masque.

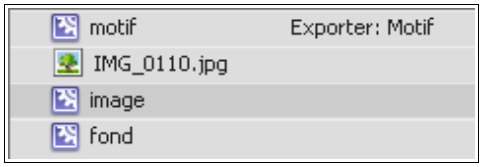
Un BitmapData n'est qu'une grille de pixels, lorsque nous dessinons dans un bitmapData, nous ne créons pas un nouvel objet mais nous changeons simplement la couleur des pixels de ce BitmapData.

Cette classe permet de faire du traitement d'image. Ce n'est pas un objet graphique car c'est la classe Bitmap qui est l'objet.

Dans cette classe BitmapData, nous utiliserons la méthode draw avec la matrice de transformation et la méthode threshold.

Nous avons sur la scène un clip image et un clip fond. Au fur et à mesure que nous allons gratter, nous allons révéler l'image qui remplacera petit à petit le fond. En ActionScript, lorsque nous utilisons la propriété masque d'un DisplayObject, le contenu du masque ne va pas masquer l'objet qui se trouve dessous mais va le révéler.

Nous allons donc avoir une première couche qui est le fond, une seconde qui est l'image et nous allons par dessus créer un DisplayOject qui servira de masque. Il sera vide au départ, il ne révélera aucune partie de l'image et le fond sera apparent, au fur et à mesure que le grattage évoluera nous dessinerons dans le masque qui révélera petit à petit l'image.



16.5.2 Événements souris, affichage et masquage curseur

Nous allons utiliser 3 événements souris :

- MOUSE_DOWN lorsque le grattage commence,
- il enchaine sur MOUSE_MOVE qui va être déclenché à chaque fois que l'on bougera la souris,
- qui se terminera par MOUSE_UP au moment où l'on relachera le bouton de la souris.

16.5.3 Matrice de transformation

Nous allons utiliser le paramètre "**matrix**" de la méthode **draw** de la classe BitmapData.

Il s'agit d'une matrice de transformation qui contient l'ensemble des transformations géométriques (translation, rotation, inclinaison, échelle) applicables sur un objet graphique.

Nous allons utiliser la translation (déplacement).

Nous allons utiliser les propriétés **tx** et **ty** de cette matrice pour déplacer le motif.

draw() méthode

```
public function draw(source:IBitmapDrawable, matrix:Matrix = null, colorTransform:ColorTransform = null, blendMode:String = null,
clipRect:Rectangle = null, smoothing:Boolean = false):void
```

Version du langage: ActionScript 3.0
: AIR 1.0, Flash Player 9

Dessine l'objet d'affichage `source` sur l'image bitmap avec la fonctionnalité de rendu vectoriel de Flash Player. Vous pouvez spécifier les paramètres `matrix`, `colorTransform`, `blendMode`, ainsi qu'un paramètre de destination `clipRect` pour contrôler l'exécution du rendu. Vous pouvez éventuellement indiquer si le bitmap doit être lissé lorsqu'il est redimensionné (cette opération ne fonctionne que si l'objet source est un objet BitmapData).

Cette méthode correspond directement au mode de traçage des objets à l'aide de la fonctionnalité de rendu vectoriel standard dans l'interface de l'outil de programmation.

L'objet d'affichage source n'utilise pas les transformations appliquées pour cet appel. Il est traité de la manière dont il apparaît dans la bibliothèque ou dans le fichier, sans transformation de matrice, de couleurs et sans mode de fondu. Pour dessiner un objet d'affichage, tel qu'un clip, en utilisant ses propres propriétés de transformation, vous pouvez copier sa propriété `transform` dans la propriété `transform` de l'objet Bitmap qui utilise l'objet BitmapData.

Remarque : l'objet `source` et (dans le cas d'un objet Sprite ou MovieClip) tous ses objets enfants doivent provenir du même domaine que l'appelant ou résider dans un fichier SWF accessible à l'appelant par le biais de la méthode `Security.allowDomain()`. Si ces conditions ne sont pas remplies, la méthode `draw()` ne dessine rien.

Cette méthode est prise en charge sur RTMP dans Flash Player 9.0.115.0 et versions ultérieures. Vous pouvez contrôler l'accès aux flux sur un serveur FMS (Flash Media Server) dans un script coté serveur. Pour plus de détails, consultez les propriétés `Client.audioSampleAccess` et `Client.videoSampleAccess` dans le *Guide de référence ActionScript d'Adobe Flash Media Server côté serveur*.

Paramètres

`source`:`IBitmapDrawable` — Objet d'affichage ou objet BitmapData à dessiner sur l'objet BitmapData. (Les classes DisplayObject et BitmapData mettent en œuvre l'interface IBitmapDrawable.)

`matrix`:`Matrix` (default = `null`) — Objet Matrix utilisé pour redimensionner, faire pivoter ou traduire les coordonnées du bitmap. Si vous ne souhaitez pas appliquer une matrice de transformation à l'image, réglez ce paramètre sur une matrice d'identité, créée à l'aide du constructeur `new Matrix()` par défaut, ou transmettez une valeur `null`.

`colorTransform`:`ColorTransform` (default = `null`) — Objet ColorTransform utilisé pour définir les valeurs de couleur du bitmap. Si aucun objet n'est fourni, les couleurs de l'image bitmap ne sont pas transformées. Si ce paramètre doit être transmis, alors que vous ne souhaitez pas transformer l'image, réglez-le sur un objet ColorTransform créé à l'aide du constructeur `new ColorTransform()` par défaut.

`blendMode`:`String` (default = `null`) — Chaîne extraite de la classe flash.display.BlendMode, qui spécifie le mode de fondu à appliquer au bitmap générée par l'opération.

`clipRect`:`Rectangle` (default = `null`) — Objet Rectangle qui définit la zone de l'image source à dessiner. Si cette valeur n'est pas fournie, aucun découpage n'est effectué et l'objet source est dessiné dans sa totalité.

`smoothing`:`Boolean` (default = `false`) — Une valeur booléenne qui détermine si l'objet BitmapData doit être lissé lors d'une mise à l'échelle ou d'une rotation demandée par le paramètre `matrix`. Le paramètre `smoothing` s'applique uniquement lorsque le paramètre `source` est un objet BitmapData. Lorsque le paramètre `smoothing` est défini sur `false`, l'image BitmapData pivotée ou mise à l'échelle peut sembler pixélisée ou irrégulière. Par exemple, les deux images suivantes utilisent le même objet BitmapData pour le paramètre `source`, mais le paramètre `smoothing` est défini sur `true` à gauche et sur `false` à droite :

Matrix

Package flash.geom
Classe public class Matrix
Héritage Matrix → Object

Version du langage: ActionScript 3.0
: AIR 1.0, Flash Player 9

La classe Matrix représente une matrice de transformation qui détermine le mappage des points d'un espace de coordonnées à l'autre. Pour appliquer diverses transformations graphiques à un objet d'affichage, vous pouvez définir les propriétés d'un objet Matrix, puis appliquer cet objet à la propriété `matrix` d'un objet Transform que vous appliquez ensuite comme propriété `transform` de l'objet d'affichage. Ces fonctions de transformation incluent la translation (repositionnement de *x* et *y*), la rotation, le redimensionnement et l'inclinaison.

Associés, ces types de transformations sont connus sous le nom de *transformations affines*. Les transformations affines préservent la rectitude des lignes au cours de la transformation, de sorte que les lignes parallèles restent parallèles.

Pour appliquer une matrice de transformation à un objet d'affichage, vous créez un objet Transform, réglez sa propriété `matrix` sur la matrice de transformation, puis réglez la propriété `transform` de l'objet d'affichage sur l'objet Transform. Les objets Matrix peuvent également être utilisés comme paramètres de certaines méthodes, indiquées ci-dessous :

- la méthode `draw()` d'un objet BitmapData ;
- les méthodes `beginBitmapFill()`, `beginGradientFill()` ou `lineGradientStyle()` d'un objet Graphics.

Un objet de matrice de transformation est considéré comme étant une matrice 3 x 3 comprenant le contenu suivant :

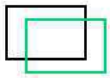
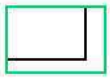
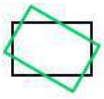
$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ u & v & w \end{bmatrix}$$

Dans le cas des matrices de transformation classiques, les propriétés `u`, `v` et `w` sont dotées de fonctionnalités supplémentaires. La classe Matrix fonctionne uniquement dans un espace bidimensionnel ; ainsi, elle suppose toujours que les valeurs des propriétés `u` et `v` sont 0,0, et que la valeur de la propriété `w` est 1,0. Les valeurs réelles de la matrice sont les suivantes :

$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Vous pouvez obtenir et définir les valeurs des six autres propriétés d'un objet Matrix : `a`, `b`, `c`, `d`, `tx` et `ty`.

La classe Matrix prend en charge les quatre principaux types de transformations : translation, redimensionnement, rotation et inclinaison. Vous pouvez définir trois de ces transformations à l'aide de méthodes spécialisées, tel que décrit dans le tableau ci-dessous.

Transformation	Méthode	Valeurs de matrice	Résultat affiché	Description
Translation (déplacement)	`translate(tx, ty)`	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Déplace les pixels `tx` de l'image vers la droite, et les pixels `ty` vers le bas.
Redimensionnement	`scale(sx, sy)`	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$		Redimensionne l'image en multipliant l'emplacement de chaque pixel par `sx` sur l'axe *x*, et par `sy` sur l'axe *y*.
Rotation	`rotate(q)`	$\begin{bmatrix} \cos(q) & \sin(q) & 0 \\ -\sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$		Fait pivoter l'image selon un angle `q`, mesuré en radians.
Inclinaison ou cisaillement	Aucun ; il est nécessaire de définir les propriétés `b` et `c`.	$\begin{bmatrix} 0 & \tan(skew_y) & 0 \\ \tan(skew_x) & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		Fait glisser l'image progressivement dans une direction parallèle à l'axe *x* ou *y*. La propriété `b` de l'objet Matrix représente la tangente de l'angle d'inclinaison sur l'axe *y*, sa propriété `c` la tangente de l'angle d'inclinaison sur l'axe *x*.

Chaque fonction de transformation modifie les propriétés de matrice actuelles, ce qui vous permet d'associer plusieurs transformations. Pour ce faire, il vous suffit d'appeler plusieurs fonctions de transformation avant d'appliquer la matrice à son objet d'affichage cible (à l'aide de la propriété `transform` de celui-ci).

16.5.4 Comparaison de la valeurs des pixels

Nous allons utiliser la méthode "**threshold**" de la classe BitmapData.

Elle renvoi le nombre de pixels répondant à notre méthode de comparaison par rapport au seuil.

threshold() méthode

```
public function threshold(sourceBitmapData:BitmapData, sourceRect:Rectangle, destPoint:Point, operation:String, threshold:uint, color:uint = 0, mask:uint = 0xFFFFFFFF, copySource:Boolean = false):uint
```

Version du langage: ActionScript 3.0
: AIR 1.0, Flash Player 9

Teste les valeurs de pixels d'une image selon un seuil spécifié et définit les pixels qui réussissent le test sur de nouvelles valeurs de couleur. L'utilisation de la méthode `threshold()` permet d'isoler et de remplacer les gammes de couleurs d'une image et d'effectuer d'autres opérations logiques sur les pixels de l'image.

La logique du test de la méthode `threshold()` est définie comme suit :

1. Si `((pixelValue & mask) operation (threshold & mask))`, définissez le pixel sur `color`.
2. Dans le cas contraire, si `copySource == true`, réglez le pixel sur la valeur de pixel correspondante dans `sourceBitmap`.

Le paramètre `operation` spécifie l'opérateur de comparaison à utiliser pour le test de seuil. Par exemple, si vous utilisez « == » en tant que paramètre `operation`, vous pouvez isoler une valeur de couleur spécifique dans une image. Ou si vous utilisez {`operation: "<", mask: 0xFF000000, threshold: 0x7F000000, color: 0x00000000`}, vous pouvez définir tous les pixels de destination comme étant entièrement transparents lorsque la valeur alpha du pixel de l'image source est inférieure à 0x7F. Vous pouvez utiliser cette technique pour les transitions animées et d'autres effets.

Paramètres

`sourceBitmapData:BitmapData` — L'image bitmap d'entrée à utiliser. L'image source peut être un autre objet BitmapData ou faire référence à l'occurrence de BitmapData actuelle.

`sourceRect:Rectangle` — Rectangle qui définit la zone de l'image source à utiliser en tant qu'entrée.

`destPoint:Point` — Point de l'image de destination (l'occurrence de BitmapData actuelle) correspondant au coin supérieur gauche du rectangle source.

`operation:String` — L'un des opérateurs de comparaison suivants, transmis en tant que chaîne : "<", "<=", ">", ">=", "==", "!="

`threshold:uint` — Valeur par rapport à laquelle chaque pixel est testé afin de déterminer s'il est inférieur ou égal au seuil ou s'il le dépasse.

`color:uint` (default = 0) — Valeur de couleur sur laquelle un pixel est réglé si le test de seuil aboutit. La valeur par défaut est 0x00000000.

`mask:uint` (default = 0xFFFFFFFF) — Masque à utiliser pour isoler un composant de couleur.

`copySource:Boolean` (default = `false`) — Si la valeur est `true`, les valeurs de pixels de l'image source sont copiées vers la destination lorsque le test de seuil échoue. Si la valeur est `false`, l'image source n'est pas copiée lorsque le test de seuil échoue.

Valeur renvoyée

`uint` — Nombre de pixels modifiés.

16.5.5 Classe Timer

Nous allons utiliser la classe Timer qui permet d'exécuter une fonction à intervalles réguliers, un certains nombre de fois.

Timer

Propriétés | Méthodes | Evénements

Package flash.utils
Classe public class Timer
Héritage Timer → EventDispatcher → Object

Version du langage: ActionScript 3.0
: AIR 1.0, Flash Player 9

La classe Timer est l'interface des horloges Flash Player. La création d'objets Timer vous permet d'exécuter le code avec une chronologie spécifiée. Utilisez la méthode `start()` pour démarrer une horloge. Ajoutez un écouteur à l'événement `timer` pour définir le code à exécuter à intervalles définis.

Vous pouvez créer des objets Timer pour les exécuter une seule fois ou à intervalles spécifiés, afin d'exécuter du code conformément au planning défini. Selon la cadence du fichier SWF ou l'environnement de Flash Player (mémoire disponible et autres facteurs), Flash Player risque de distribuer les événements à intervalles légèrement décalés. Par exemple, si un fichier SWF doit être lu à 10 images/seconde [i/s], soit une fréquence de 100 millisecondes, mais que votre horloge est réglée de sorte à déclencher un événement à 80 millisecondes, Flash Player déclenche l'événement à une fréquence proche de 100 millisecondes. Les scripts qui consomment beaucoup de mémoire risquent également de décaler les événements.

Consulter les exemples

Voir aussi
Utilisation des dates et heures
Contrôle des intervalles temporels

Propriétés publiques

Afficher les propriétés publiques héritées

Propriété	Défini par
currentCount : int [] Nombre total de déclenchements de l'horloge depuis son démarrage.	Timer
delay : Number Délai, en millisecondes, entre les événements d'horloge.	Timer
repeatCount : int Nombre total de répétitions définies de l'horloge.	Timer
running : Boolean [] Etat actuel de l'horloge : true si l'horloge est en cours d'exécution, false dans le cas contraire.	Timer

Méthodes publiques

Afficher les méthodes publiques héritées

Méthode	Défini par
Timer(delay:Number, repeatCount:int = 0) Construit un nouvel objet Timer en tenant compte du délai et du nombre de répétitions spécifié.	Timer
reset():void Arrête l'horloge, le cas échéant, et redéfinit la propriété currentCount sur 0, tout comme le bouton de remise à zéro d'un chronomètre.	Timer
start():void Démarre l'horloge, le cas échéant.	Timer
stop():void Arrête l'horloge.	Timer

Evénements

Afficher les événements hérités

Evénement	Synthèse	Défini par
timer	Distribué lorsqu'un objet Timer atteint un intervalle spécifié conformément à la propriété Timer.delay.	Timer
timerComplete	Distribué lorsque le traitement du nombre de requêtes défini par Timer.repeatCount est terminé.	Timer

Nota :

Nous pouvons modifier la difficulté du jeu soit par :

- la modification du rapport de grattage,
- la modification de l'aspect du clip motif.

```
// Création de masque occurrence de l'objet BitmapData
// qui prend les dimensions de l'image, transparent, remplissage noir transparent
var masque:Bitmap = new Bitmap( new BitmapData(image.width, image.height, true, 0) );
addChild(masque);

// Application du masque
image.mask = masque;
// les objets masqués et le masque doivent avoir la propriété "cacheAsBitmap" à true
image.cacheAsBitmap = true;
masque.cacheAsBitmap = true;

// Occurrence motif du symbole en bibliothèque (Classe : Motif)
var motif:DisplayObject = new Motif ();

// Occurence matriceMotif de la classe Matrix
var matriceMotif:Matrix = new Matrix ();

// Occurrence timerValidation du timer (toutes les secondes, répétition infinie)
var timerValidation:Timer = new Timer (1000);

// Écouteur d'événement : click bouton souris
addEventListener(MouseEvent.MOUSE_DOWN, mouseEventsHandler);

// Écouteur d'événement : à chaque exécution du timer (toutes les secondes)
timerValidation.addEventListener(TimerEvent.TIMER, timerValidationHandler);

function dessinerMotif ():void
// Dessiner le motif
{
    // modification des propriétés tx et ty de la matrice
    // en utilisant la position souris
    matriceMotif.tx = mouseX - motif.width / 2;
    matriceMotif.ty = mouseY - motif.height / 2;
    // capture de l'aspect du motif pour le dessiner dans le BitmapData
    masque.bitmapData.draw (motif, matriceMotif);
}

function validerFinJeu (pBD:BitmapData, pRapportGrattage:Number):Boolean
{
    // fin si : nb pixels grattés >= nb total pixels * taux grattage ok
    // pRapportGrattage : taux grattage ok
    var totalPixels:uint = pBD.width * pBD.height;
    var pixelsGrattes:uint = totalPixels - pBD.threshold(pBD, pBD.rect, new Point(), "==", 0);
    return pixelsGrattes / totalPixels >= pRapportGrattage;
}
```

```
// Centralisation des événements souris
function mouseEventsHandler (e:MouseEvent):void
{
    switch (e.type)
    {
        case MouseEvent.MOUSE_DOWN :
            // Début du Timer
            timerValidation.start ();
            // Masquage curseur souris
            Mouse.hide ();
            // Écouteur d'événement sur l'image : mouvement souris
            addEventListener (MouseEvent.MOUSE_MOVE, mouseEventsHandler);
            // Écouteur d'événement sur la scène : relachement bouton souris
            stage.addEventListener (MouseEvent.MOUSE_UP, mouseEventsHandler);
        // break;
        // suppression du break pour que ça continue sur le cas MOUSE_MOVE
        // pour que le grattage commence au 1er click avant que la souris bouge

        case MouseEvent.MOUSE_MOVE :
            dessinerMotif ();
        break;

        case MouseEvent.MOUSE_UP :
            // Fin du Timer
            timerValidation.stop ();
            Mouse.show ();
            // Fin d'écoute du mouvement et du click souris
            removeEventListener (MouseEvent.MOUSE_MOVE, mouseEventsHandler);
            stage.removeEventListener (MouseEvent.MOUSE_UP, mouseEventsHandler);
        break;
    }
}

// Gestion timer
function timerValidationHandler (e:TimerEvent):void
{
    if (validerFinJeu (masque.bitmapData, 0.6))
    // si fin du jeu (taux grattage) : fonction validerFinJeu renvoie true
    {
        trace("Bravo !");
        // Fin d'exécution du Timer
        timerValidation.stop ();
        // Ré-affichage du curseur souris
        Mouse.show ();
        // Affichage image en totalité
        image.mask = null; // suppression du masque
        removeChild (masque);// suppression affichage masque
        // Fin d'écoute des événements souris
        removeEventListener (MouseEvent.MOUSE_MOVE, mouseEventsHandler);
        stage.removeEventListener (MouseEvent.MOUSE_UP, mouseEventsHandler);
        removeEventListener(MouseEvent.MOUSE_DOWN, mouseEventsHandler);
    }
}
```