Syntaxe essentielle en AS3

Chaines

```
var titre : String  =  «Les fleurs du mal» ;
var auteur_nom : String  =  «Baudelaire » ;
var auteur_prenom : String  =  «Charles   » ;
```

On utilise le signe « + » comme operateur de concatenation (juxtaposition de deux chaines)

```
var chaine_finale: String  =  titre+ « de  »+ auteur_prenom +auteur_nom ;
trace(chaine_finale) ; // renvoie : les fleurs du mal de Charles   Baudelaire
```

(ici l'espace après « Charles » est directement inseré dans la variable ce qui permet d'eviter son ajout entre les deux variables)

Commentaires

Un commentaire se commence avec un double slash (« // ») ;
Un commentaire multiligne débute par /* et se termine par */

Ndr : utilisez les commentaires pour documenter vos classes méthodes et propriétés le mieux possible. Pour des raisons évidentes, modifier une classe 6 mois après sa création ou travailler en équipe sans documentation correcte se révèle vite être une gageure. Utilisez de préférence la notation multiligne directement au dessus de l'objet à documenter cela permet aux commentaires de s'afficher directement dans l'interface de flashDevelop.

Nombres et chiffres

En actionScript les décimales s'écrivent derrière un point 1,27 se note donc 1.27 .
Si l'on rentre la valeur sous forme de chaine elle est automatiquement convertie en valeur numérique.

```
var valeur_1 : Number =  1.5;
var valeur_2 : Number =  "20848.2" ; // transtypage automatique de la chaine vers le type Number

var resultat : Number = valeur_1+valeur_2;
trace (resultat)  // renvoie : 20849.7
```

Si nous savons que jamais la variable ne contiendra de décimale. Nous pouvons lui attribuer un type qui prendra moins de ressources que le type **Number** : nous pouvons la stocker dans un type **int** qui peut recevoir des valeurs environ entre -- 2 200 000 000 et 2 200 000 000

Dans une moindre mesure, si nous savons que la variable ne sera jamais un nombre négatif ni ne contient de décimales nous pouvons la stocker dans un type **uint** (entre 0 et, environ, 4 500 000 000).
C'est par exemple le cas pour un index attaché à un tableau :

```
var tableauvaleur : Array = new Array ("citrouille","avocat", "peche");
var ma_variable : int = 1;
trace ( tableauvaleur[ma_variable] ) // renvoie : avocat
```

int peut être aussi le type de l'identifiant d'une donnée externe chargée dynamiquement.

Ndr concernant la manipulation des nombres : un grand nombre d'opérations mathémathiques, geométriques, trigonometriques et comparatives peuvent être effectuée avec l'objet Math
L'exemple ci-dessus par exemple montre l'utilisation d'une comparaison entre deux nombres renvoyant la plus petite valeur :

```
var minimum : Number = Math.min(valeur_1,valeur_2) ;
Trace(minimum) ; // Renvoie : 1.5
```

On se servira fréquement de cette méthode statique pour determiner si la taille image chargée dynamiquement dépasse de l'emplacement qui lui est alloué.

Crochets

Les crochets dans actionscript3 permettent outre d'initialiser un tableau, de

- créer ou d'appeler une propriété dans un objet.
- insérer une variable dans un élément écrit en syntaxe pointée à la place de la chaine utilisée comme nom d'occurrence .

Exemple :
Le choix du parcours dans l'arborescence peut comporter une variable dans le code, dans l'exemple ci-dessous un xml est structuré pour renvoyer une valeur différente pour un identifiant donné en fonction du nœud parent employé :

```
private var donnees : XML =
<main>
        <select>
                <couleurs contour="0xFFFFFF" fond="0x000000"></couleurs>
                <alphas contour=".8" fond="0" ></alphas>
        </select>
        <no_select>
                <couleurs contour="0xCCFF33" fond="0x234567"></couleurs>
                <alphas contour=".8" fond="0" ></alphas>
        </no_select>
                </main>;
```

Dans le script nous avons initialisé une variable de type String qui indique le statut d'un DisplayObject par rapport aux actions de souris. Nous pouvons accéder aux valeurs stockées dans le xml pour laquelle le nom du nœud correspond a la chaine définissant le statut :

:

```
var statut_objet : String=  « no_select » ; // la variable peut prendre aussi la valeur
« select »
var couleur_selection : Number = donnees [statut_objet].couleurs@fond ;
trace(couleur_selection)//renvoie 0x000000 ;
```

Null

Indicateurs précisant que l'objet n'existe pas. D'autres marqueurs existe tels que : undefined ou NaN (pas un nombre).
Pour supprimer un objet (DisplayObject ou classe non graphique) du script et le rendre éligible à l'effacement définitif, il doit être passé à null et ne doit plus avoir d'écouteurs y faisant référence :

```
Objet_a_supprimer = null ;
```

Objets

Lorsqu'ils sont crées dans une classe, l'ajout d'une paire identifiant / Valeur devra être noté
Comme suit :

```
var nouvelle_valeur : String = « test » ;
var obj_exemple :Object=new Object() ;
 // ( ndr :  Il existe des moyens alternatifs de déclarer un nouvel objet)

obj_exemple[« nouvel_identifiant »] = nouvelle_v aleur ;
```

il pourra être invoqué en écrivant :

```
obj_exemple[« nouvel_identifiant »] ;
```

ou

```
obj_exemple.nouvel_identifiant
```

Tableaux

On crée un tableau soit par la méthode habituelle pour les types composites

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

soit en inscrivant directements ses valeurs entre deux crochets séparés par des parenthèses

```
var tableau_references : Array =  {"valeur_1",  « valeur_2 », « valeur_3 »} ;
```

On accède individuellement à ses valeurs par le numéro d'index du tableau

```
trace( tableau_references[0]  ) ;
// Renvoie : valeur_1
```

Ndr : l'exemple montre que l'index du tableau commence à zéro. Si l'on veut trouver le dernier index du tableau il faudra retrancher 1 à sa longueur :

```
var dernier_index :int=  tableau_references.length-1 ;
var dernier_element :String= tableau_references[dernier_index] ;
```

Opérateurs de comparaison

On les trouvera le plus fréquemment dans les opérateur de condition comme if (si…) :

Les plus fréquent sont :
if(valeur_a **<** valeur_b) : *si (valeur_a est **strictement inférieure** à valeur_b)*
if(valeur_a **<=** valeur_b) : *si (valeur_a est **inférieure ou égale** à valeur_b)*
if(valeur_a **>** valeur_b) : *si (valeur_a est **strictement supérieure** à valeur_b)*
if(valeur_a **>=** valeur_b) : *si (valeur_a est **supérieure ou égale** à valeur_b)*
if(valeur_a **==** valeur_b) : *si (valeur_a **est égale** à valeur_b)*
if(valeur_a **!=** valeur_b) : *si (valeur_a **est différent** de valeur_b)*

if(valeur_a >= valeur_b **||** valeur_c == 0) : *si (valeur_a est supérieure ou égale à valeur_b **OU** si valeur_c est égal à 0) ;*
if(valeur_a >= valeur_b **&&** valeur_c == 0) : *si (valeur_a est supérieure ou égale à valeur_b **ET** si valeur_c est égal à 0)*

Opérateurs conditionnels

Pour établir une condition et le comportement à adopter si cette condition est remplie. Plusieurs if(condition) {comportements} peuvent être imbriqués ou se suivre. Si la condition est remplie plusieurs fois le comportement ad hoc sera executé. Si l'on veut établir une condition excluant les autres on utilisera else if().

```
var chaine :String= "tomate"
if(chaine=="tomate")
{
        chaine="poivron" ;
}
else if(chaine=="poivron") // sans else chaine prendrait pour valeur « courgette »
chaine="courgette" ;
trace(chaine) // renvoie : poivron
```

Operateurs d'itération

(++,--,+=,-=)

++ et **--** : respectivement ajoute ou retranche 1 à la valeur à qui ces opérateurs sont appliqués.

```
var valeur_variable: Number = 12,23 ;
var increment : int = 31 ;
valeur_variable += increment ;
valeur_variable - = increment ;
```
Ajoute ou retranche la valeur de l'incrément à la valeur.

Boucles

Nous n'évoquerons ici que les boucles principalement utilisées.
Une boucle permet d'éxécuter une action autant de fois qu'il lui est indiqué et d'éviter ainsi la répétition d'un même code.

Boucle for()

```
var longueur_tableau : int = tableau.length

for (var :int=0 ;i< longueur_tableau  ;i++)
{
	var obj:DisplayObject=tableau[i];
	if (obj != null)
	{
		Obj.name="objet_"+i ;
		addChild(Obj);
	}
}
```

Boucle for each()

Nouveauté du langage AS3 la boucle for each présente le moyen le plus simple de parcourir un tableau d'éléments.
La boucle parcoure tout le tableau, chaque index correspondant à une variable typée
Si le tableau contient plusieurs types non compatibles on mettra une astérisque à la place du type.

```
for each(var : element :* in tableau)
{
	// si l'élément est un bitmap on le transtype.
	if(element is Bitmap)
	{
		var bmp:Bitmap=element as Bitmap;
		// l'objet bmp contient maintenant toutes les specificité d'un objet bitmap
	}
}
```

Envoyer une requête http :

Flash c'est bien mais comment envoyer le visiteur vers une autre page se son site ou vers un autre site ?
Le getUrl d'as 2 à été remplacé par une fonction globale du package flash.net :

navigateToURL(requête : URLRequest, fenetre :String)

La requête est une instance d'URLRequest(). Cet objet mérite toute votre attention parce que c'est par son entremise que l'on pourra envoyer des variable par GET ou POST. Ajouter des headers. Reportez-vous au langage de référence pour son emploi et

au chapitre 14 de l'ouvrage de thibaut Imbert : « Pratique de l'actionScript 3 ». Pour l'explication détaillée d'URLRequest (envoi de données par GET ou POST).

```
import flash.net.navigateToURL
import flash.net.URLRequest;

//…
//…
bouton.addEventListener(MouseEvent.CLICK,ecouteClic,true);

function ecouteClic(even:MouseEvent):void
{
	var requete:URLRequest = new URLRequest(“http://ressources.ikonika.fr”);
	navigateToURL(requete,"_blank")
}
```

Gestion des erreurs

Try{}
Catch(variable_erreur :Error){}
Finally{}

Certaines erreurs peuvent ne pas apparaitre à la compilation, mais durant l'utilisation de l'application par un utilisateur lambda. Il à dans ce cas la mauvaise surprise de voir s'afficher une fenêtre d'erreur, et de voir s'interrompre le script.
Lors d'un tel cas le lecteur flash émet une erreur héritant de la classe Error.
La séquence *try, catch, finally* permet de détecter, d'intercepter l'erreur émise, et de programmer un fonctionnement alternatif.

On la programme de la manière suivante :

```
Try
{
	Le morceau de code susceptible d'émettre une erreur
}

Catch  ( variable_erreur :Error )
{
	Le comportement alternatif au bloc try en cas d'erreur.
	Note : diverses informations sont visibles en passant par la variable
	Paramétrée dans la fonction.
}
Finally
{
	La phase du script qui s'effectuera erreur ou non
}
```

Note : bien qu'il sagisse bien d'écouter un évènement, nous pouvons remarquer que le bloc try, catch… ne nécessite pas d'installer un écouteur.

la gestion des erreurs comporte aussi des événements répondant au modèle évènementiel d'actionScript 3. De tels évènements sont émis par exemple lors d'une erreur de chargement d'un fichier externe comme une image (…).
Dans ce cas précis il est donc possible de programmer une image ou une animation par defaut qui apparaisse en cas de problème de chargement.