ExtendScript pour Adobe After Effects

Formation de Duduf
http://www.duduf.training

Ce document n'est qu'un résumé, un pense-bête, d'un bout de formation dispensée par Duduf. Il réunit les principaux points théoriques à retenir sans être exhaustif, et de nombreuses remarques, astuces pratiques et exemples concrets n'y sont pas présents.

I – Documentation

Le langage ExtendScript pour les logiciels de la suite Adobe est un dérivé du JavaScript, notamment utilisé pour le web, et utilise la même syntaxe, types, objets, et fonctions de base.

Pour de la documentation sur tous les types, objets et fonctions de base de Javascript, se référer au menu *Aide/Outil de visualisation d'Object Model...* du logiciel *Adobe ExtendScript ToolKit*, ou bien faire une recherche internet pour obtenir de l'aide sur Javascript.

Pour de la documentation sur tous les objets et fonctions ExtendScript communes à tous les logiciels adobe (Interface Graphique Utilisateur, Socket Client/Serveur, XML, XMP...) se référer soit à l'aide dans le logiciel *Adobe ExtendScript Toolkit*, ou pour une documentation plus complète au fichier PDF fourni par Adobe : *JavaScriptToolsGuide_CS4* (ou *CS5*,*CS6*, etc.). Pour être sûr de la rétro-compatibilité des scripts, il faut se référer à la version la plus ancienne où l'on souhaite une compatibilité.

Pour de la documentation sur les objets et fonctions propres à Adobe After Effects (maniement des projets, compos, calques, rendus...) se référer au fichier PDF *aftereffectscs3_scripting_guide* ou bien *After-Effects-CS6-Scripting-Guide* suivant la compatibilité voulue.

II – Syntaxe, JavaScript

La syntaxe est celle de Javascript

- **Symbole de fin d'instruction**

```
;
```

- **Commentaires**

```
//commente tout ce qui est à droite du double slash

/*
commente tout ce qui est compris
entre slash étoile et étoile slash
*/
```

- **Variables**

• Déclaration de variable(s) :

```
var nomDeVariable1, nomDeVariable2 ;
```

• Avec initialisation (exemple avec un nombre) :

```
var nomDeVariable = 0 ;
```

• Suppression de variable(s) :

```
delete variable1,variable2 ;
```

• Types

• Rien
```
null
```

• Tableaux (array)
```
[valeur1,valeur2,valeur3]
```

• Chaînes de caractères (string)
```
«chaine de caractères»
```
ou
```
'chaine de caractères'
```

• Caractère d'échappement
```
\
```
exemple :
```
«l'exemple d'un texte avec des \«guillements\»»
```
ou
```
'l\'exemple d\'un texte avec des \«guillements\»'
```
Il faut donc échapper le backslash lui même si on veut le mettre dans une chaîne
```
«une phrase avec un \\ backslash (et un seul)»
```

nouvelle ligne :
```
«\n»
```
retour :
```
«\r»
```
retour à la ligne sous mac :
```
«\n»
```
retour à la ligne sous windows :
```
«\r\n»
```
tabulation :
```
«\t»
```

- **Conditions**

```
if (condition) {
    instruction1 ;
    instruction2 ;
    } else {
    instruction3 ;
    instruction4 ;
    }
```

- Et :

```
&&
```

- Ou :

```
||
```

- Abrégée :

```
condition ? instructionSiTrue : instructionSiFalse ;
```

- Opérateurs conditionnels :

```
> //strictement supérieur
< //strictement inférieur
<= //inférieur ou égal
>= //supérieur ou égal
== //égal
!= //n'est pas égal
```

- Négation :

```
!
```

• Boucles

• Avec un incrément :

```
for (i = début ; i < fin ; i++) {
    instruction1 ;
    instruction2 ;
    }
```

• Pour parcourir un tableau

```
for (i in tableau) {
    instruction1 ;
    instruction2 ;
    }
```

• Tant que :

```
while (condition) {
    instruction1 ;
    instruction2 ;
    }
```

• Sortir d'une boucle :

```
break ;
```

• Sauter à l'itération suivante :

```
continue ;
```

• Fonctions

• Création d'une fonction :

```
function nomDeLaFonction(argument1,argument2) {
    instruction1 ;
    instruction2 ;
    return variable ;
    }
```

La variable donnée par return est le résultat de la fonction, si il y en a un.
La fonction s'arrête dès qu'elle rencontre un return, mais elle peut en contenir plusieurs (avec des conditions)
Il peut ne rien y avoir après le return, pour ne rien retourner et juste stopper la fonction, ou bien null.

```
return null ;
```

Le return n'est pas obligatoire dans la fonction, la fonction n'a pas forcément de résultat, elle peut juste servir à faire des actions.
Pour avoir plusieurs résultats, il faut utiliser un tableau.

```
return [result1,result2] ;
```

• Usage d'une fonction :

```
var varPourRecupResultat = fonction(argument1,argument2) ;
```

- **Objets/Classes**

```
var objet = {} ;
```

• Contiennent des attributs (propriétés, valeurs) :

```
objet.attribut
```

qui peuvent être modifiables (ou pas)

```
objet.attribut = nouvelleValeur ;
```

• Contiennent des méthodes (fonctions) :

```
objet.méthode(argument1,argument2) ;
```

• Les classes peuvent être héritées : une classe héritée contient tous les attributs et méthodes de la classe parent, plus d'autres à elle

```
var objetParent = {} ;
objetParent.attribut1 = true ;
objetParent.attribut2 = 12 ;
objetParent.methode1 = methode1 ;
var objetHerite = classeParent ;
objetHerite.methode2 = methode2 ; // objetHerite possède les attri-
but1 et attribut2, methode1 et en plus methode2
```

• Opérateur pour vérifier à quelle classe appartient un objet :

```
instanceof
```

exemple :

```
if (objet1 instanceof Classe) { instruction ; }
```

III – Tableaux (Array)

Fonctions couramment utilisées sur les tableaux :

• Créer un tableau vide :

```
var tableau = [] ;
```

• Récupérer à un élément par son index (le premier élément est à l'index 0) :

```
var element = tableau[index] ;
```

• Modifier un élément à son index :

```
tableau[index] = nouvelleValeur ;
```

• Ajouter un objet à la fin :

```
tableau.push(objet) ;
```

ou au début :

```
tableau.unshift(objet) ;
```

• Enlever le dernier élément (en le récupérant dans une variable)

```
var elementEnleve = tableau.pop() ;
```

ou le premier :

```
var elementEnleve = tableau.shift() ;
```

ou plusieurs éléments où on veut, en récupérant les éléments supprimés dans un nouveau tableau (si on ne précise pas quantiteASuppr, tout est supprimé par partir de premierIndex) :

```
var tableauSupprimes = tableau.splice(premierIndex,quantiteASuppr) ;
```

on peut directement remettre d'autres éléments à la place de ceux supprimés :

```
tableau.splice(i,3,nouvelElement1,nouvelElement2,nouvelElement3) ;
```

• Créer un tableau à partir de deux autres (concaténation) :

```
var nouveauTableauConcat = tableau1.concat(tableau2) ;
```

• Récupérer toute une partie d'un tableau (dernierIndex non compris) :

```
var tableauRecup = tableau.slice(premierIndex,dernierIndex) ;
```

• Insérer un élément à un index précis (dans une fonction pratique) :

```
function insert(element,indexInsertion,tableau) {
    //récupère tout ce qui est avant l'insertion
    var tableauGauche = tableau.splice(0,indexInsertion) ;
    //reste ce qui est après
    var tableauDroite = tableau ;
    //ajouter l'élément après tableauGauche, puis ajouter tableau-
Droite
    tableau = tableauGauche.push(element) ;
    tableau = tableau.concat(tableauDroite) ;
    return tableau ;
    }
```

• Transformer un tableau de chaîne de caractères en une seule chaîne de caractères :

```
var chaine = tableau.join(separateur) ;
```

• Trier un tableau dans l'ordre alphabétique/numérique :

```
tableau.sort() ;
```

• Trier un tableau avec une fonction prédéfinie qui compare deux éléments

```
tableau.sort(fonction) ;
```

IV – Chaines de caractères (String)

• Créer une chaine vide :

```
var chaine = «» ;
```

• Assembler deux chaînes :

```
var nouvelleChaine = chaine1 + chaine2 ;
```

• Comparer deux chaines :

```
if (chaine1 == chaine2)
```

• Trouver l'index d'un caractère ou d'une chaine dans une autre chaine (donne -1 si pas trouvé) :

```
var index = chaine1.indexOf(chaine2) ;
```

• Passer une chaine en minuscules :

```
var chaine = chaine.toLowerCase() ;
```

en majuscules :

```
var chaine = chaine.toUpperCase() ;
```

• Regarder si une chaine contient une autre chaine :

```
if (chaine1.indexOf(chaine2) >= 0)
```

sans s'embarrasser de la casse (minuscule/majuscule) :

```
if (chaine1.toLowerCase().indexOf(chaine2.toLowerCase()) >= 0)
```

• Récupérer un morceau de la chaine :

```
var chaineRecup = chaine.substr(indexDebut,nombreDeCaracteres) ;
```

ou :

```
var chaineRecup = chaine.substring(indexDebut,indexFin) ;
```

• Transformer une chaine en tableau (chaque élement séparé par le(s) caractère(s) separateur) :

```
var tableau = chaine.split(separateur) ;
```

• Remplacer la premiere occurrence de caractères dans la chaine :

```
chaine.replace(caracteresARemplacer,nouveauxCaracteres) ;
```

ou toutes les occurrences :

```
chaine.split(caracteresARemplacer).join(nouveauxCaracteres) ;
//ou bien :
while(chaine.indexOf(caracteresARemplacer) >=0) {
    chaine.replace(caracteresARemplacer,nouveauxCaracteres) ;
    }
```

V – Boîtes de dialogue de base

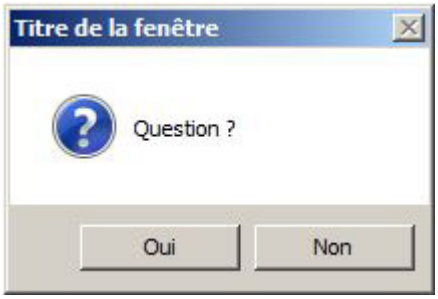
Il y a quelques boîtes de dialogue simples et toutes faites pour communiquer avec l'utilisateur :

• Afficher une alerte :

```
alert(texteDeLAlerte,titreDeLaFenetre,afficherUneIconeErreur) ;
//si  afficherUneIconeErreur == true, affiche une icône rouge, sinon un panneau bleu
```

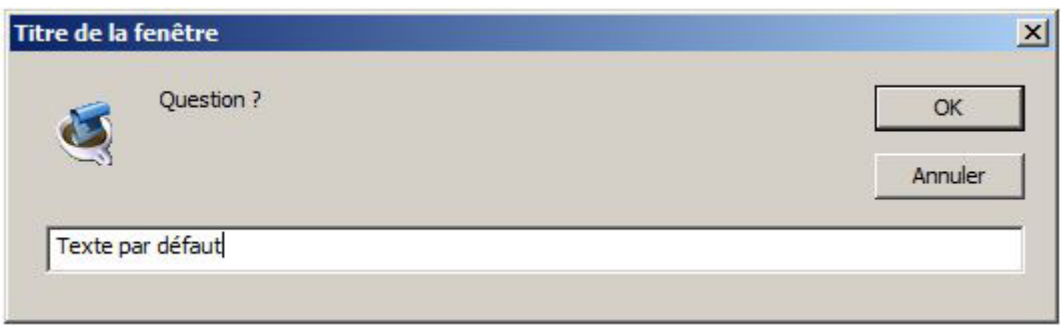
• Demander une confirmation (oui ou non)

```
var reponse = confirm(question,boutonNonParDefaut,titre) ;
if (reponse) ...
//boutonNonParDefaut : bouléen
```



• Demander du texte (affiche une ligne de texte éditable

```
var reponse = prompt(question,texteAfficheParDefaut,titre) ;
```



VI – Accès aux dossiers et aux fichiers (Folder et File)

Les chemins sont des chaînes de caractères, soit en notation fsName qui dépend du système,

```
«C:\\un dossier\\un sous dossier\\un fichier.pdf» //Win
«/Documents/un fichier.pdf» //Mac/Unix/Linux
```

soit en URI (fonctionne quel que soit le système d'exploitation)

```
«/C/un dossier/un sous dossier/un fichier.pdf»
```

Extendscript tolère aussi un mélange des deux

```
«C:/un dossier/un sous dossier/un fichier.pdf»
«C:\\un dossier/un sous dossier/un fichier.pdf» //extendscript est
très tolérant...
```

• Les dossiers (Folder)

• La récupération et le maniement des dossiers se fait en créant (instançant) un objet Javascript représentant le dossierà manier

```
var dossier = new Folder(cheminDuDossier) ;
```

Un objet Folder ne représente pas nécessairement un fichier qui existe réellement, le chemin n'est pas forcément valide.

• Vérifier si le dossier existe :

```
if (dossier.exists)
```

• Créer un dossier « réellement » :

```
var nouveauDossier = new Folder(cheminDuNouveauDossier) ;
if (!nouveauDossier.exists) nouveauDossier.create() ;
```

• Afficher une boite de dialogue pour sélectionner un dossier et le récupérer celui sélectionné (null si annulé) :

```
var dossier = Folder.selectDialog(question) ;
```

• Le dossier des documents de l'utilisateur :

```
var mesDocuments = Folder.myDocuments ;
```

• Le bureau

```
var bureau = Folder.desktop ;
```

• Les données d'application de l'utilisateur (dossier caché)

```
var appData = Folder.userData ;
```

• Le dossier de démarrage de l'application (dossier d'installation de After Effects)

```
var demarrage = Folder.startUp ;
```

• La corbeille

```
var corbeille = Folder.trash ;
```

• Dossier parent :

```
var parent = dossier.parent ;
```

• Chemin du dossier (String) :

```
var chemin = dossier.absoluteURI ; //URI
var cheminFS = dossier.fsName ; //fsName
```

• Supprimer un dossier (ne le met pas à la corbeille, le dossier doit être vide)

```
dossier.remove() ;
```

• Récupérer les élements (fichiers et dossiers) contenus dans un dossier (optionnellement filtrer les noms par une chaine de caractères) :

```
var tableauDesElements = dossier.getFiles(filtre) ;
```

• Ouvrir le dossier avec l'explorateur windows ou le finder Mac

```
dossier.execute() ;
```

• Les fichiers (File)

• La récupération et le maniement des fichiers se fait en créant (instançant) un objet Javascript représentant le fichier à manier

```
var fichier = new File(chemin) ;
```

Un objet File ne représente pas nécessairement un fichier qui existe réellement, le chemin n'est pas forcément valide.

• Vérifier si le fichier existe :

```
if (fichier.exists)
```

• Créer un fichier « réellement » :

```
var nouveauFichier = new Folder(cheminDuNouveauFichier) ;
if (!nouveauFichier .exists) nouveauFichier.create() ;
```

• Nom du fichier (String)

```
var nom = fichier.name ;
```

• Lancer le fichier (même effet qu'un double clic dessus)

```
fichier.execute() ;
```

• Supprimer un fichier (sans le mettre dans la corbeille) :

```
fichier.remove() ;
```

• Copier un fichier :

```
fichier.copy(nouveauCheminEtNomDuFichier) ;
```

• Mettre un fichier à la corbeille

```
if ( fichier.copy(Folder.trash.absoluteURI + «/» + fichier.name) )
fichier.remove() ;
```

• Renommer un fichier

```
fichier.rename(nouveauNom) ;
```

• Pour lire ou écrire dans un fichier, il faut d'abord prévenir le système d'exploitation qu'on l'ouvre

```
fichier.open(«r») ; //en lecture seule
fichier.open(«w») ; //en écriture uniquement, le contenu sera remplacé si on modifie le fichier après
ficier.open(«e») ; //en lecture/écriture, permet de modifier le contenu du fichier
fichier.open(«a») ; //pour seulement ajouter du contenu à la fin du fichier
```

• Et il faut le fermer une fois les opérations dessus terminées

```
fichier.close() ;
```

• Pour récupérer le contenu d'un fichier et le mettre dans une chaîne de caractères

```
var contenu = fichier.read() ;
```

• Pour récupérer un seul caractère :

```
var caractere = fichier.readch() ;
```

• Pour récupérer une seule ligne de texte :

```
var ligne = fichier.readln() ;
```

• Chaque utilisation de readch ou readln lit le caractère/la ligne suivante, pour savoir si on a atteint la fin du fichier :

```
if (fichier.eof) //end of file
```

• Fonction pour se déplacer dans un fichier, et lire/écrire à l'endroit qu'on veut

```
fichier.seek(index) ;
```

• Fonction pour faire un tableau des lignes contenues dans le fichier :

```
function getLines(fichier) {
    fichier.open(«r») ;
    var lignes = [] ;
    while (!fichier.eof) {
        lignes.push(fichier.readln()) ;
        }
    fichier.close() ;
    return lignes ;
    }
```

• Ecrire dans un fichier (si ouvert en type « w » remplace entièrement l'ancien contenu, si en « e », remplace à l'endroit où on est) :

```
fichier.write(chaineAEcrire) ;
```

• Afficher une boîte de dialogue d'ouverture de fichier, et récupérer le fichier choisi par l'utilisateur (null si aucun ou annulation)

```
var fichier = File.openDialog(question,filtre,multiselect) ; //filtre
sous la forme : «Tous :*,*,ExtendScript :*,jsx;*,jsxinc»
//multiselect : bouléen
```

• Afficher une boîte de dialogue pour sauvegarder un fichier (demander un chemin + un nom) :

```
var fichier = File.saveDialog(question,filtre) ;
```

• Fonction « enregistrer sous » :

```
function saveAs(texteAEnregistrer) {
    var fichierSauve = File.saveDialog(«Enregistrer sous...») ;
    if (fichierSauve == null) return ; //si annulé on arrete
    var save = true ;

    if (fichierSauve.exists) save = confirm(«Le fichier « +
    fichierSauve.name + « existe,\nvoulez vous le remplacer ?»,true) ;

    if (save) {
        fichierSauve.open(«w») ;
        fichierSauve.write(texteAEnregistrer) ;
        fichierSauve.close() ;
        }
    }
```

VII – Accès à un serveur (Socket)

C'est un objet Socket qui gère l'échange avec un serveur, quel qu'il soit (http, ftp, sql...)

• Créer l'objet socket :

```
var connexion = new Socket;
```

• Il faut commencer par connecter le socket au serveur :

```
var connexionOk = connexion.open(adresseServeur) ;
```

AdresseServeur est sous la forme : IP:Port ou Adresse:Port
Exemple : « 192.168.1.57:80 » ou bien « www.google.com:80 »
Le port 80 est le port standard en http (sur le web), 21 pour le ftp.
ConnexionOk est un bouléen qui vaut true si le socket a bien réussi à se connecter (pas de coupure réseau, le serveur est joignable...)

• On peut ensuite envoyer une requète au serveur (en vérifiant d'abord si la connexion est bien faite) :

```
var requeteOk = false ;
if (connexionOk) requeteOk = connexion.writeln(requete) ;
```

exemple de requete http pour obtenir le contenu d'une page :

```
«GET http://www.google.ch»
```

• Une fois la requète effectuée, on peut lire la réponse du serveur :

```
var reponse = «» ;
if requeteOk reponse = connexion.read(nombreDeCaracteresALire) ;
```

• Il ne reste plus qu'à gérer la chaîne de caractères qui correspond à la réponse, et ne pas oublier de fermer la connexion :

```
connexion.close() ;
```

VIII – Interface Utilisateur (ScriptUI)

La plupart du temps, un script avec une interface utilisateur sur After Effects utilise de base : soit un « *Panel* » : le fichier du script est placé dans le dossier *Scripts/ScriptUI Panels* et lancé depuis le menu « *fenêtres* » de After Effects (supérieur à CS3), soit une « *Window* » quand le fichier est lancé depuis le menu « *fichier/scripts* » de After Effects (supérieur à 6.5).

C'est donc soit dans ce *Panel* soit dans la *Window* qu'on va créer l'interface (boutons, cases à cocher, champs de texte, etc)

• Script simple en Window

• Il faut créer la fenètre de base de l'interface graphique :

```
var fenetre = new Window(type,bounds,titre,creationPorperties) ;
```

Il y a trois types :
«window» : fenetre standard d'application, considérée comme une application autonome (donc listée dans la barre des tâches windows).
«palette» : fenetre plus simple, sans les boutons réduire et agrandir dans la barre de titre, et « enfant » d'une application (donc pas listée par la barre des tâches windows)
«dialog» : fenètre modale : son affichage empèche l'utilisation de l'application parent, jusqu'à ce qu'elle ait été fermée.
Bounds : ce sont les coordonnées de la fenêtre, données par rapport à l'écran, en tableau :
[coinSupGaucheX,coinSupGaucheY,coinInfDroitX,coinInfDroitY]
On peut remplacer bounds par undefined si on veut que la fenètre soit placée automatiquement.
- CreationProperties : utile surtout pour rendre la fenètre « resizeable » :

```
{ resizeable:true}
```

• Il faudra, une fois l'interface créée et le script prèt à être exécuté, afficher la fenètre :

```
fenetre.show() ;
```

• Si on a besoin de la fermer :

```
fenetre.hide() ;
```

• Script en Panel

Le panel ne doit pas être créé dans le script, il est créé par after au moment où l'utilisateur lance le script depuis le menu « fenêtres ». Par conséquent, il faut juste récupérer le Panel créé dans une variable pour pouvoir l'utiliser par la suite ; ce Panel, c'est le script lui même, donc :

```
var fenetre = this ;
```

Il n'y a pas besoin du .show() après, puisque c'est after qui affiche le panel automatiquement au lancement du script.

• Script compatible à la fois en Panel (dossier ScriptUI Panels) ou utilisable en Window (ailleurs)

• Comme lorsque le script est lancé depuis le menu « fenêtres » le panel est créé automatiquement, il suffit de vérifier si le Panel existe pour savoir si on doit créer une fenêtre ou pas :

```
var fenetre ;
if (this instanceof Panel) fenetre = this ;
else fenetre = new Window(type,bounds,titre,creationPorperties) ;
```

• Plus tard, il faudra afficher la fenêtre si on n'est pas en panel :

```
if (fenetre instanceof Window) fenetre.show() ;
```

• L'idéal étant d'inclure tout le script dans une fonction, il faut donc passer this comme argument à la fonction :

```
function leScript(obj) {
    //les fonctions nécessaires au script
    if (obj instanceof Panel) fenetre = obj ;
    else fenetre = new Window(type,bounds,titre,creationPorperties) ;
    //construction de l'interface...
    if (fenetre instanceof Window) fenetre.show() ;
}
leScript(this) ; //on lance la fonction principale, en passant this en argument pour les vérifications à la création de l'interface
```

• Ajout des éléments d'interface

Il y a deux types d'éléments : les conteneurs, qui servent à organiser l'interface (dont Panel et Window font partie), et les contrôleurs, qui sont les éléments d'intéraction (boutons, textes, etc.)
Les trois conteneurs existant sont :
«window» : conteneur de base, de trois types possibles : window, palette ou dialog (cf partie « Script simple en Window »)
«panel» : si comme conteneur de base, c'est un panneau ancrable dans After Effects, si c'est enfant d'une window ou d'un autre conteneur, c'est un groupe d'élements entouré d'un cadre avec optionnellement un titre.
«group» : c'est un groupe d'éléments, invisible, qui sert juste au layout de l'interface (et à grouper les radio-buttons).
• Les élements enfants des conteneurs (d'autres conteneurs ou des contrôleurs) sont alignés soit horizontalement, soit verticalement. On peut le décider :

```
conteneur.orientation = «row» ;
conteneur.orientation = «column» ;
```

• Changer les marges autour du conteneur :

```
conteneur.margins = valeurEnPixels ;
```

• Changer l'espace entre ses enfants :

```
conteneur.spacing = valeurEnPixels ;
```

• Changer l'alignement du conteneur par rapport à son parent :

```
conteneur.alignment = [horizontal,vertical] ;
```

et l'alignement des enfants de ce conteneur :

```
conteneur.alignChildren = [horizontal,vertical] ;
```

• Les alignements horizontaux possibles sont :

```
«fill» : prend toute la place en largeur, «left»,»center» ou «right»
```

Les alignements verticaux possibles sont :

```
«fill»,»top»,»center» ou «bottom»
```

• Pour ajouter des enfants dans le conteneur :

```
conteneur.add(type,bounds,texte,propriétés) ;
```

Quelques types de contrôleurs pouvant être ajoutés :

«statictext» : un simple texte, non modifiable et non sélectionnable par l'utilisateur.
«edittext» : un champ de texte éditable, qui peut avoir en propriété : {multiline:true}
«button» : un bouton simple
«iconbutton» : un bouton, avec une icône (image) en plus dessus
«checkbox»: une case à cocher
«radiobutton» : une case à cocher ronde, dont une seule peut être sélectionnée au sein d'un groupe
«slider» : un curseur horizontal
«progressbar» : une barre de progression

• Tous ces contrôleurs possèdent différents attributs tels que text, value, enabled, etc.

• On peut détecter l'intéraction avec l'utilsateur grâce à des évènements, auxquels on assigne une fonction :
En définissant la fonction directement sur l'évènement :

```
controleur.onClick = function () { faireQuelqueChose ; }
```

ou en la définissant avant pour être plus clair :

```
function aFaire() {
    faireQuelqueChose ;
}
controleur.onClick = aFaire ;
```

Quelques évènements courant (les événements varient en fonction des contrôleurs) :
`onClick` : un clic sur le contrôleur (boutons...)
`onChange` : la valeur a été changée (edittext, slider...)
`onChanging` : la valeur est en train d'être changée (edittext, slider...)
`onResize` : la taille du conteneur a été changée (window, panel...)
`onResizing` : la taille du conteneur est en train d'être changée (window, panel...)