CEPTIA
CEntre Permanent des Technologies Industrielles Avancées

JAVASCRIPT

Génaël VALET

Ce cours Javascript décrit les principales fonctionnalités du langage. Il n'est nullement exhaustif et doit être utilisé pour découvrir le langage. Des ouvrages de référence seront indispensables pour aller plus loin dans la programmation Javascript

CENTRE DE FORMATION INDIVIDUALISE DU GITA
Lycée Technique Régional Diderot
61, rue David d'Angers – 75019 PARIS
tél. : 01.40.40.36.27/28— Fax : 01.40.40.36.30
Email : greta.gita@wanadoo.fr
N° SIRET 197 507 122 00046 – code APE 804

INTRODUCTION

Le Javascript est un langage interprété et orienté objet qui s'exécute côté client ou côté serveur. Il est particulièrement adapté au World Wide Web puisqu'il fonctionne avec le H.T.M.L (Hyper Text Markup Langage). Ce langage prend une place prépondérante dans l'environnement Internet. C'est grâce à lui que les pages Web s'enrichit avec un contenu interactif et animé. Les possibilités de H.T.M.L étant limitées, Javascript s'impose comme un complément indispensable.

Javascript est-il normalisé ?

Oui et non. En fait, Javascript est né chez la société Netscape et portait le nom de Live Script. Devant le succès et le phénomène de mode du langage JAVA, Live Script est devenu Javascript. Lorsque Microsoft s'est rendu compte de l'impact de ce nouveau langage, il a crée Jscript. Les deux éditeurs ont implémentés des fonctionnalités propres et cela pose certains problèmes de compatibilité entre les versions de navigateurs internet.

Heureusement, l'organisme de normalisation ECMA a crée ECMA Script. Ce standard permet de fixer certaines fonctionnalités du langage. De ce point de vue, les Javascript et Jscript de Netscape et Microsoft sont compatibles avec la norme ECMA Script. Cependant, ECMAScript ne s'arrête pas sur l'aspect interface, ce soin étant laissé aux éditeurs de logiciels qui rivalisent à coups de nouvelles technologies incompatibles les unes avec les autres.

En conclusion, on peut dire que la tendance est à la normalisation mais il demeure quelques problèmes d'incompatibilité entre les versions et les éditeurs.

Qu'y a t-il sous le capot ?

Les principales caractéristiques du langage :

- ✓ Orienté objet (On manipule des objets tels que des fenêtres, des boutons, des images...)
- ✓ Interprété (A l'inverse du Java, Javascript est compilé au moment de l'exécution du programme)
- ✓ Dérivé du C++ (Les fonctions, les opérateurs sont les mêmes mais simplifiés et plus tolérants)
- ✓ Portable (Indépendant du matériel, il peut être exécuté partout dès que le système est pourvu d'un interpréteur Javascript.
- ✓ Inséré dans des application hôtes (Navigateur, applis propriétaires)

Javascript côté client et côté serveur

Javascript peut s'exécuter côté serveur et côté client. Le côté client est utilisé pour agrémenter des pages HTML d'un contenu plus dynamique et interactif. Le côté serveur, est réservé à des applications en relation avec JAVA ou la technologie ASP de Microsoft. Il est possible d'entrer en relation avec des bases de données et les langages de script tels que Javascript sont les acteurs privilégiés de cette communication.

Javascript et HTML

Un script peut se trouver au sein d'une page HTML grâce à la présence de la balise <SCRIPT>. Le corps du script sera soit directement dans la page HTML ou à l'extérieur. Dans ce cas un lien extérieur vers le fichier contenant le programme sera inséré dans la page HTML.

Balise Script

Cette balise marque le début d'un script dans la page *HTML*. Elle peut s'insérer autant de fois que nécessaire dans les balises *HEAD* ou *BODY*. Elle contient plusieurs attributs:

Attribut	Signification
type	Spécifie le langage de script utilisé. Cet attribut doit être un type MIME Internet
langage	Spécifie également le langage mais sous forme d'un identificateur texte simple et non standard
src	Indique la localisation du script lorsqu'il est séparé du document

Attributs de la balise SCRIPT

1er Exemple d'utilisation de la balise *SCRIPT* :

```
<SCRIPT langage="Javascript">
... Contenu du script
</SCRIPT>
```

Ou :

```
<SCRIPT type="text/javascript">
... Contenu du script
</SCRIPT>
```

La dernière méthode n'est compatible qu'avec les versions 4.0 des navigateurs. Elle utilise le format MIME.

2ème Exemple d'utilisation de la balise *SCRIPT* avec référence à un fichier externe :

```
<SCRIPT langage="Javascript" src="myScript.js"> </SCRIPT>
```

Variables et Données

Déclaration de variables (var)

La déclaration d'une variable se fait avant son utilisation grâce au mot-clé **var**.

```
var toto ;
```

Il est possible de déclarer tout type de variable sans en spécifier le type. Cette souplesse est due à la grande permissivité du Javascript

Types primitifs

Booléen (Boolean)

Le type booléen ne peur prendre que 2 valeurs (True et False)

```
titi = true;
tutu = false;
```

Nombre (Number)

Il n'y a pas de distinction entre les réels, les entiers ou les octets. Un nombre peut-être codé de différentes manières :

```
nombre1 = 10.25;
nombre2 = 17;
nombre3 = 1.05e+14
nombre4 = 0xFF52            // Nombre hexadécimal
```

Le nombre est limité aux valeurs inférieures à 10^{308} environ.

Chaîne (String)

Il n'existe pas de type pour représenter un seul caractère. Le seul moyen pour y parvenir est de créer une chaîne avec un seul caractère. La chaîne doit être délimitée par des guillemets simples ou doubles. On utilise les guillemets simples dans le cas où une instruction Javascript est exécutée au sein d'une propriété d'une balise HTML (Appel à une fonction par exemple)

```
chaine = "Salut";
```

ou

```
<A HREF="http://www.diderot.org" onClick="alert('Vous allez vers le site Diderot')">Diderot </A>
```

L'exemple précédent s'insère dans le corps d'une page HTML. Il crée un lien vers le site Diderot et appelle la fonction Javascript alert qui affiche une boîte de dialogue contenant la chaîne transmise à la fonction. On remarquera la présence des guillemets simples pour le passage de la chaîne dans la fonction visant à ne pas perturber le code HTML.

Tableaux

Un tableau s'utilise pour stocker des données différentes dans une seule variable dont l'indice varie. Pour déclarer un tableau, on utilise la syntaxe **new**:

```
var jours= new Array(7);
jour[0] = "Lundi";
jour[1] = "Mardi";
jour[2] = "Mercredi";
...
```

ou encore

```
var jours= new Array("Lundi", "Mardi",  "Mercredi" ... );
```

Les éléments d'un tableau ne sont pas forcément de même type. Un tableau peut très bien contenir des chaînes et des nombres.

Pour créer des tableaux à plusieurs dimensions, il suffit de créer des tableaux de tableaux :

```
var matrice = new Array(2);

matrice[0] = new Array(2);
matrice[1] = new Array(2);

matrice[0][0] = 1;
matrice[1][0] = 2;
matrice[0][1] = 3;
matrice[1][1] = 4;
```

Date

Il s'agit d'un type d'objet permettant de traiter les dates et les heures. Ces objets ont des méthodes qui autorisent un programme javascript à connaître l'heure système de l'ordinateur client.

Au départ, un objet *date* ne contient rien. Il dispose de plusieurs méthodes visant à traiter les dates sous toutes leurs formes.

Voici une vue d'ensemble non exhaustives des méthodes des objets de type *date* :

Attribut	Signification
getDate	Retourne le jour du mois en utilisant l'heure locale
getDay	Retourne le jour de la semaine en utilisant l'heure locale (0 pour dimanche ...)
getHours	Retourne un entier correspondant à l'heure du jour en utilisant l'heure locale (0 à 23)
getMinutes	Retourne un entier correspondant à la minute de l'heure en utilisant l'heure locale (0 à 59)
getSeconds	Retourne un entier correspondant à la seconde en utilisant l'heure locale (0 à 59)
getTime	Retourne un entier correspondant au nombre de millisecondes écoulées depuis le 1er Janvier 1970.
setTime, setDate ...	Méthodes permettant de fixer la date ou l'heure pour les objets concernés

Pour plus de renseignements, consultez la documentation électronique Microsoft (JSCript v5 documentation) fournie lors du stage. Vous y trouverez toutes les méthodes pouvant être appelées avec des objets *Date*.

Voici un exemple qui crée un objet *date* et lui affecte l'heure courante et affiche le nombre de jours depuis le 1er Janvier 1970 dans une boîte de dialogue :

```
var myDate = new Date;

myDate.setTime ( myDate.getTime() );
alert ("Nombre dejours depuis 01/01/70 :" + ( myDate.getTime() * 1000 * 3600 * 24 ) );
```

Conversion de types

Il existe des fonctions de conversion de type qui permettent de spécifier explicitement la conversion à effectuer. Voici des exemples de conversion explicites ou implicites :

```
var a = String(21.34)          // Donne a = "21.34"
var b = Number ("12.56")       // b = 12.56
var c = + ("0xFE99");          // c = 65177
var d = "14.1" – 5 ;           // Conversion implicite de "14.1" en nombre : d = 9.1
```

Opérateurs

Les opérateurs arithmétiques, relationnels, logiques, bit à bit ou spéciaux obéissent aux règles rencontrées en langage C. Les opérateurs sont utilisés pour effectuer des tests ou manipuler des données. Il est possible de combiner plusieurs opérations en une seule. Voici un récapitulatif des principaux opérateurs :

Opérateurs arithmétiques

Les opérateurs +, -, *, / , % (Modulo) s'utilisent comme ci-dessous :

```
z = 1 + 5 ; y = 4 * 5 ; x = 14 / 2 ; w = 9 – 5 ; e = 4 % 3;
```

Jscript 1.0 tronque les valeurs réelles en entiers avant de leur appliquer l'opérateur %. Ainsi , 5.5%2.3 aura pour résultat 1

Opérateurs relationnels

Utilisés pour les comparaisons, les opérateurs relationnels sont tous binaires. Dans une clause de type *if* , la condition pour l'exécuter est que cette condition soit vraie (égale à 1)

```
if ( variable == 2 )
        alert( 'variable est bien egale a 2');
```

Dans l'exemple ci-dessus, l'opérateur == est utilisé pour comparer variable au nombre 2.

Opérateur	Signification
> et <	supérieur à et inférieur à
>= et <=	supérieur ou égal à et inférieur ou égal à
== et !=	égal à et différent de

Opérateurs logiques

Permet d'effectuer des opérations logiques entre des variables (&&, || , !) . On notera que les signes sont doublés pour ne pas confondre avec les opérateurs bit à bit.

```
if (   (variable1 == 2) && (variable2==3 )
        alert( 'variable1 est bien egale a 2 et variable2 est bien egale a 3');
```

Opérateurs bit à bit

Ces opérateurs traitent les nombres comme s'il s'agissait de chaînes de 32 bits. Les opérateurs NOT, AND, OR, XOR, décalage vers la gauche, la droite signé et non signé s'utilisent grâce à la syntaxe suivante :

- ✓ NOT : ~
- ✓ AND : &
- ✓ OR : |
- ✓ XOR : ^
- ✓ Décalage à gauche : <<
- ✓ Décalage à droite : >>
- ✓ Décalage à droite non signé : >>>

Opérateurs unaires spéciaux

Il s'agit d'opérateur d'incrémentation ou de décrémentation de variables. Ces opérations sont soit postfixés (Exécutés après) ou soit prefixés (Exécutés avant).

```
b = a + (d++ );
```

d ne sera incrémenté qu'après avoir effectué l'opération a+d (ancienne valeur de d). Il s'agit d'une post décrémentation. Les 2 expressions ci-dessous sont équivalentes

```
b = a + ( --d );
b = a + (d – 1 );
```

Opérateurs d'affectation

Le signe égal est l'opérateur d'affectation principal. Il est possible d'utiliser une forme contractée pour éviter de l'écriture :

```
a += 3 ;
a = a + 3;
```

Les 2 expressions ci-dessus sont équivalentes.

Autres opérateurs

Il existe un opérateur ternaire d'affectation conditionnelle. Cet opérateur est un moyen rapide de donner à une variable une valeur choisie entre deux. C'est un raccourci pour écrire des instructions *if* simples *(Voir paragraphe suivant)*

Les 2 expressions suivantes sont équivalentes :

```
x = ( a > b ) ? c : d  ;     // Si a > b alors x prend la valeur c sinon x prend la valeur d
if ( a > b ) x = c else x = d ;
```

Ce genre d'opérateur est à manier avec précaution car il diminue la lisibilité des programmes et il est source d'erreur de programmation.

Contrôle de flux

Héritées du langage C , les mécanismes de contrôle de flux donnent au programmeur la possibilité d'effectuer des boucles ou des clauses if else.

If Else

```
if ( condition )
        instruction ou bloc
// ou
if ( condition )
        instruction ou bloc
else
        instruction ou bloc
```

Par exemple :

```
if ( x < 5 )
        alert(' x est strictement inférieur à 5');
else
        alert(" x est supérieur ou égal à 5");
```

while

Permet d'effectuer une boucle sous certaines conditions

```
while ( condition )
        instruction ou bloc
```

Voici un programme permettant d'incrémenter x jusqu'à ce qu'il soit égal à 5

```
while ( x != 5)
        x = x + 1;
```

for

Permet d'effectuer des boucles avec un nombre d'itération déterminé

```
for ( initialisation ; condition, modification)
        instruction ou bloc
```

Voici un exemple permettant d'écrire les 10 premiers chiffres dans une page HTML

```
for ( compte = 1 ; compte <= 10 ; compte++)
        document.write ("Chiffre ", compte );
```

Les fonctions

Les fonctions permettent de subdiviser les tâches à effectuer en petits sous-programmes. Ces fonctions peuvent prendre des paramètres et renvoyer tout type de données.

```
function nom ( paramètres )
        {
        instructions et éventuellement return pour renvoyer une données
        }
```

Voici une fonction renvoyant une chaîne dépendant du nom passé en paramètres

```
function bonjour ( nom )
        {
        return ( " Bonjour  " + nom);
        }
```

La fonction pourra être appelé par ailleurs comme ceci et chaine recevra "Bonjour Didier" :

```
chaine = bonjour ( "Didier");
```

Les objets

Introduction

Un objet comporte certaines similitudes avec les structures du langage C. Un objet est un regroupement de variables et de fonctions qui peuvent être réutilisées autant que nécessaire.

```
var contact = new Object;

contact.nom = "VALET";
contact.prenom = "Génaël";
contact.tel = "0146511521"
```

L'exemple ci-dessus crée un objet dont les variables nom, prenom et tel sont associés. Rien ne nous empêche de créer un tableau d'objets qui composerait un répertoire téléphonique. Les variables associées à l'objet sont des propriétés.

Pour libérer la place qu'un objet occupe en mémoire, il suffit d'appeler la fonction delete :

```
delete contact;
```

Les méthodes des objets

Les objets peuvent également posséder des méthodes qui seront exécutées lorsqu'on fait appel à eles. Ces méthodes appartiennent à l'objet et son liés à lui.

L'exemple d'objet contact peut être agrémenté d'un méthode d'affichage définie avant :

```
function afficher()
        {
        document.write(" Le nom est : " + this.nom);
        document.write(" Le prénom est : " + this.prenom);
        document.write(" Le tél est : " + this.tel );
        }

var contact = new Object;
contact.nom = "DUPOND";
contact.prenom = "Lajoie";
contact.tel = "0144441122";
contact.afficher_contact = afficher();

// Pour afficher l'objet, il suffira de

contact.afficher_contact();
```

En résumé, une méthode est une fonction contenue dans l'objet. On remarquera l'utilisation de la variable this qui fait référence à l'objet qui a invoqué la méthode et qui permet d'accéder directement aux variables de l'objet appelant.

Prototypes et constructeurs

Dans l'exemple précédent, nous avons été obligé de définir un objet contact et de définir ses propriétés ainsi que sa méthode. Pour créer un nouveau contact, il faudrait saisir le même code. Dans le cas d'enregistrements multiples, le code à saisir peut devenir trop copieux. Heureusement, il existe la propriété prototype qui , associée aux fonctions spéciales constructeur permettent de remédier à ce problème.

Voici notre exemple d'illustration des fonctions spéciales constructeur qui s'inspire de notre objet contact :

```
function afficher()
        {
        document.write(" Le nom est : " + this.nom);
        document.write(" Le prénom est : " + this.prenom);
        document.write(" Le tél est : " + this.tel );
        }

function creer_contact(nom , prenom, tel)
        {
        this.nom = nom;
        this.prenom = prenom;
        this.tel = tel;
        }

// Vient alors la déclaration

var personne1 = new  creer_contact ("DURANT", "Emile", "0223234545");
personne1.afficher_contact = afficher;

personne1.afficher_contact();
```

L'objet sera alors automatiquement crée avec ses propriétés par l'instruction new qui s'occupera d'appeler la fonction constructeur créer_contact .

Il reste a automatiser l'association de la fonction afficher comme méthode de l'objet C'est le rôle de prototype

```
function afficher()
        {
        document.write(" Le nom est : " + this.nom);
        document.write(" Le prénom est : " + this.prenom);
        document.write(" Le tél est : " + this.tel );
        }

function creer_contact(nom , prenom, tel)
        {
        this.nom = nom;
        this.prenom = prenom;
        this.tel = tel;
        }

créer_contact.prototype.afficher_contact = afficher();

// Vient alors la déclaration

var personne1 = new  creer_contact ("DURANT", "Emile", "0223234545");
var personne2 = new créer_contact("DUPONT","Lajoie","0145451212");
```

Dans l'exemple ci-dessus , personne1 et personne2 auront tous les deux une méthode afficher_contact() disponible.

Exemples d'objets du navigateur

Les objets représentent le principal intérêt de Javascript. Leurs méthodes ou leurs propriétés permettent de les manipuler à loisir. Tout le problème consiste à connaître leur existence. Prenons l'exemple d'un programme Javascript qui s'exécute dans une page Web et donc dans un navigateur.

L'objet document représente la page HTML actuellement affichée dans le navigateur. On peut donc accéder facilement à l'URL permettant de l'atteindre :

```
alert( document.location );
```

L'exemple ci-dessus affichera une boîte de dialogue dans laquelle s'affichera l'adresse internet du document en cours.

Rien ne nous empêche d'ailleurs de modifier cette URL.

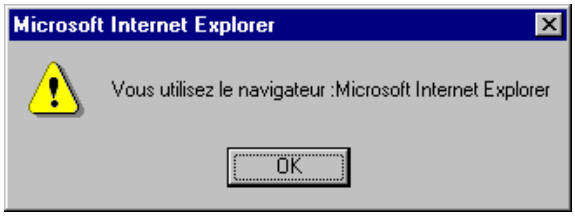
```
document.location = "http://www.diderot.org/sommaire.htm";
```

Cette instruction permet au navigateur de charger la page sommaire.htm du lycée Diderot.

Un autre exemple consiste en un programme qui permettrait de savoir quel navigateur l'utilisateur utilise pour naviguer :

```
alert ( " Vous utilisez le navigateur " + navigator.appName);
```

Une boîte de dialogue s'affiche en précisant la version du navigateur utilisé :



Modèle objet du navigateur

Le modèle objet d'un navigateur représente la hiérarchie des objets que l'on peut utiliser qui appartiennent à l'application hôte (Le navigateur). Les exemples du paragraphe précédent illustrent cette notion.

Cependant, le modèle objet varie d'un navigateur à l'autre. Nous allons essayer de recenser quelques objets importants et de spécifier pour quelles versions de navigateur ils sont utilisables. La liste ne sera absolument pas exhaustive car chaque éditeur de logiciel (Microsoft, Netscape) développe ses propres objets.

Objet Document

L'objet Document est le plus intéressant des objets fournis par le navigateur. La liste des objets contenus dans *document* dépend du navigateur utilisé. Les différences entre Netscape et Internet Explorer sont multiples et il vaut mieux se référer aux documents de référence de chaque éditeur (*ClientReferenceJavascriptNCS13.pdf* pour Netscape et le fichier d'aide *Microsoft Windows Script*) .

Le livre *Le guide du programmeur JavaScript* chez *Eyrolles* référence également les différences mais il commence à dater.

Objet Navigator

L'objet *Navigator* contient quelques méthodes et propriétés utiles pour connaître toutes les informations sur le navigateur utilisé. Voici un récapitulatif des méthodes et des propriétés les plus utilisées :

Propriétés	Description
appCodeName	Nom de code du navigateur
appName	Nom du produit
appVersion	Numéro de version
cookieEnabled	Renvoie true si les cookies sont autorisés
platform	Nom de plate-forme
userAgent	Nom du navigateur indiqué dans l'en-tête des requêtes HTTP adressées au serveur par le client

Propriétés de l'objet Navigator

Méthodes	Description
javaEnabled()	Renvoie *true* si la machine virtuelle java est présente sur le système et *false* dans le cas contraire.

Méthodes de l'objet Navigator

Voici ce que donne l'affichage de telle propriétés avec IE5 sous Windows NT :

```
appCodeName : Mozilla
appName : Microsoft Internet Explorer
appVersion : 4.0 (compatible; MSIE 5.0; Windows NT)
userAgent : Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
cookieEnabled : true
Language : undefined
mimeTypes ( Netcape 4 et + seulement ) :
platform : Win32
plugins ( Netcape 4 et + seulement ) :
```

Voici la même chose avec Netscape version 4.7 sous NT :

appCodeName : **Mozilla**
appName : **Netscape**
appVersion : **4.7 [fr] (WinNT; I)**
userAgent : **Mozilla/4.7 [fr] (WinNT; I)**
cookieEnabled : **undefined**
Language : **fr**
mimeTypes (Netcape 4 et + seulement) **:[object MimeTypeArray]**
platform : **Win32**

Le programme Javascript permettant d'obtenir ce résultat se nomme *version.js* et la page HTML correspondante est *version.html*

HTML et Javascript

L'intérêt de Javascript côté client est de pouvoir faire fonctionner des programmes en relation avec le HTML. La balise Script décrite dans la 1ère partie de ce document permet d'insérer du code Javascript dans une page HTML.

Ce code peut intervenir à plusieurs endroits d'un page HTML :

- ✓ Dans une balise script
- ✓ Comme valeur d'un attribut de gestion d'événement d'une balise (BODY, IMG , A , ...)

Pour plus de précisions sur le HTML, veuillez vous reporter au document *Langage HTML* donné en début de formation.

Formulaires

Il est possible de gérer certaines actions ou événement avec des formulaires par Javascript. Tous les contrôles d'un formulaire possède des attributs où Javascript peut intervenir. Voici une liste des contrôles possibles :

- ✓ INPUT : Contrôle de saisie
  - ✓ **text** : champ texte
  - ✓ **password** : champ mot de passe
  - ✓ **checkbox** : champ case à cocher
  - ✓ **radio** : champ bouton radio
  - ✓ **submit** : bouton soumettre formulaire
  - ✓ **reset** : bouton pour remettre à zéro tous les champs du formulaire
  - ✓ **file** : demande à l'utilisateur de désigner un fichier. Lorsque le formulaire est soumis, le contenu de ce fichier sera transmis au serveur comme une valeur de n'importe quel autre contrôle.
  - ✓ **hidden** : champ ne devant pas apparaître dans le formulaire
  - ✓ **image** : bouton graphique
  - ✓ **button** : champ de type bouton
- ✓ **BUTTON** : Bouton d'action
- ✓ **SELECT et OPTION** : Listes déroulantes
- ✓ **TEXTAREA :** Zone de saisie de texte multilignes et multicolonnes

Événements intrinsèques

Ce sont des actions provoquées par un événement particulier. Ce sont des points d'entrée pour un script. Cela signifie qu'un script particulier peut-être exécuté chaque fois que l'événement se produit.

Voici une liste non exhaustive des événements possibles :

Attribut / Signification
onLoad survient lorsque le navigateur vient de finir le chargement de la fenêtre, ou de tous les cadres. Cet attribut peut être utilisé dans des éléments *BODY* et *FRAMESET (Voir P**Erreur! Signet non défini.**et P**Erreur! Signet non défini.**)*

onUnload survient lorsque le navigateur la fenêtre ou un cadre. Cet attribut peut être utilisé dans des éléments *BODY* et *FRAMESET (Voir PErreur! Signet non défini.et PErreur! Signet non défini.)*
onClick , onDblclick survient lors du clic ou double clic de souris sur un l'objet concerné.
onMouseOver survient lorsque la souris passe au dessus de l'objet concerné
onMouseDown survient lorsque le bouton de la souris est enfoncé mais pas relâché
onMouseUp survient lorsque le bouton de la souris est relâché (On suppose qu'il avait été enfoncé)
onMouseOut survient lorsque le pointeur de la souris sort de l'objet
onMouseMove survient lorsque le pointeur de souris est déplacé sur l'objet concerné

Voici un exemple permettant d'exécuter une fonction Javascript lorsque l'on clique sur un bouton de formulaire. La fonction ouvre une boîte de dialogue délivrant un message :

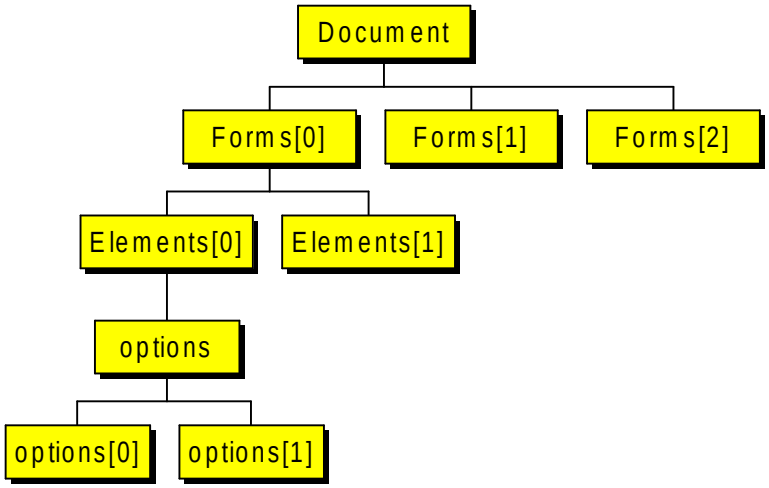
```
<html>
<head>
<title>Gestion d'évènement dans un formulaire</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#FFFFFF" >

<form name="form1" method="post" action="">
 <input type="submit" name="Submit" value="Submit" onClick="alert('Vous avez cliqué sur le
bouton submit')">
</form>

</body>
</html>
```

Objets formulaires

Il est possible d'accéder aux contrôles d'un formulaire en Javascript par l'intermédiaire des méthodes et des propriétés inhérentes au langage orienté objet du langage.

Accès aux objets FORMS

Pour accéder , on manipule l'objet *document.forms* du navigateur :



Il existe 2 méthodes pour accéder par programme aux formulaires. La 1ère consiste à considérer les objets par leur numéros. Dans ce cas, voici comment on accèderai au 1er élément du 1er formulaire :

```
variable = document.forms[0].elements[0].value
```

La 2ème méthode (plus propre) est de donner un nom à chaque contrôle ainsi qu'au formulaire. De cette manière chaque valeur d'attribut est accessible par Javascript. Voici un exemple de page HTML contenant un formulaire avec un champ de saisie de texte ainsi qu'un bouton de validation :

```
<html>
<head>
<title>Gestion d'évènement dans un formulaire</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF" >

<form name="MyForm" method="post" action="">
  Nom : <input type="text" name="nom" value="Saisie du nom">
  <input type="submit" name="Envoyer" value="Submit" >
</form>

</body>
</html>
```

ce qui donne à peu près ceci :

On accède à la valeur du champ texte. Dans l'exemple suivant, une boîte :

```
message = "Vous avez tapé le nom :" + document.MyForm.nom.value
alert( message );
```

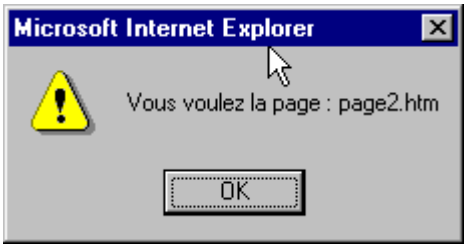
Vous trouverez cet exemple dans le fichier *form1.htm*

Cas du contrôle SELECT

Dans le cas d'un contrôle liste déroulante (Balise SELECT) , le code HTML se présente de la manière suivante :

```
<html>
<head>
<title>Gestion d'évènement dans un formulaire</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#FFFFFF" >

<script language="JavaScript">
function click_choix()
{
MonLien = document.MyForm.liens.selectedIndex;
Message = "Vous voulez la page : ";
alert (Message + document.MyForm.liens.options[MonLien].value);
}
</script>

<form name="MyForm" method="post" action="">
  Liens :
   <select name="liens" onChange="click_choix();" >
      <option >CLiquez sur une option</option>
      <option value="page1.htm">Option 1</option>
      <option value="page2.htm">Option 2</option>
      <option value="page3.htm">Option 3</option>
  </select>
</form>
</body>
</html>
```

On remarquera que l'attribut *OnChange* de la balise SELECT fait appel à la fonction *click_choix()* qui affichera une boîte de dialogue délivrant un message de ce style si l'option 2 est choisie:



Evènement *focus*

Il est possible de donner la main à un utilisateur grâce à la méthode *focus()* en rendant un contrôle de formulaire prêt à la saisie. Cette méthode s'applique aux contrôles de formulaire et peut s'appeler de la façon suivante :

```
document.MyForm.texte1.focus();
```

Dans cet exemple, le contrôle *texte1* du formulaire *MyForm* verra son focus être activé. C'est un moyen utile d'aide à la saisie de formulaire.

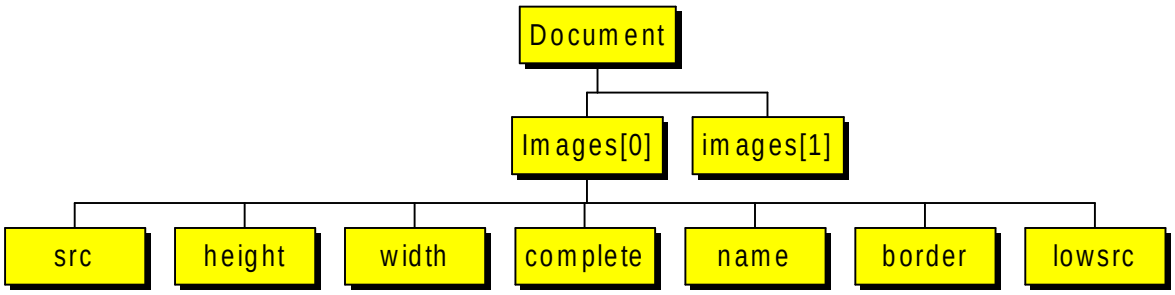
Gestion des images

Il est possible de gérer les images dynamiquement par les événements intrinsèques vus pour les formulaires. Il faut savoir néanmoins que cette gestion n'est possible qu'avec les versions 4.0 et supérieure des navigateurs. En fait, seul IE4 sait gérer les images complètement. Ce n'est qu'avec les versions 4.x que Netscape a adopté entièrement cette technologie.

Néanmoins, lorsque l'image est intégrée à une balise <A> , les évènements *OnMouseOver* , *OnMouseOut* et *OnClick* sont disponibles dès la version 3.0 de Netscape et Internet Explorer 4.

Objet Image

L'objet Image possède des propriétés et des méthodes permettant de le manipuler. On peut également associer un objet Image à une balise HTML *IMG* présente dans la page. Les attributs d'une balise *IMG* se présentent ainsi :



src : URL de l'image

height , width : Hauteur et Largeur

complete : vaut *TRUE* lorsque le navigateur a fini le chargement de l'image

name : Nom de l'image

border : Taille de la bordure de l'image

lowsrc : URL d'une image basse résolution de remplacement

Déclaration d'une image

La fonction *new* déclare un objet de type Image . On peut ensuite accéder aux propriétés de l'image en suivant le modèle indiqué plus haut

```
MonImage = new Image;

MonImage.src = "http://www.diderot.org/images/oiseau.gif";
MonImage.height = 154;
```

L'exemple ci-dessus crée une image , spécifié son URL et fixe sa taille.

Le fait de déclarer une image et de lui affecter son attribut *src* oblige le navigateur à charger l'image vers la mémoire locale.

Relation avec les balises IMG

Pour effectuer une relation entre un objet image et une balise *IMG*, il suffit d'affecter les propriétés dynamiquement :

```
document.Image5.src = MonImage.src
```

L'exemple ci-dessus remplace l'image *image5* par la valeur *src* de l'objet *MonImage*.

Dynamic HTML

Il s'agit d'une technologie permettant de faire évoluer les navigateurs et rendre les pages web plus interactives. En fait, chaque navigateur possède *un modèle objet (Document Object Model)* ou chaque objet peut se manipuler à volonté. Le but de ce cours n'est pas de faire un cours sur DHTML mais plutôt de manipuler des objets du navigateur par Javascript .

Le DHTML est composé de 4 technologies qui étudiées une à une dépassent l'objectif de ce stage :

- ✓ Javascript
- ✓ Les styles (C.S.S : Cascading Style Sheet)
- ✓ HTML
- ✓ le D.O.M : Document Object Model

Vous trouverez en annexe et dans les ressources fournies lors du stage, tous les moyens d'obtenir des informations plus précises sur chaque technologie.

Utilisation des balises *<DIV> et <LAYER>*

Le modèle objet des navigateurs Netscape et IE ne considèrent pas de la même façon ces 2 balises HTML. La balise *<LAYER>* n'est prise en charge que par Netscape et tend à disparaître. La balise *<DIV>* est la plus populaire et la plus utilisée.

Pour plus de précision sur l'emploi de la balise *DIV* , se reporter au document *Langage HTML* fourni en début de stage.

Manipuler une balise DIV avec Netscape

Le modèle objet du navigateur Netscape défini les balises *DIV* et *LAYER* comme un objet *LAYER* appartenant à l'objet *DOCUMENT*. Pour accéder à un tel objet , il faudrait coder comme suit :

```
document.layers[0]
```

ou encore en désignant la balise par son nom (Attribut *ID*)

```
document.MyDIV
```

Pour écrire en HTML dans une balise *DIV* par l'intermédiaire de Javascript, on utilise la méthode *write* comme dans un document normal. En fait la balise *DIV* est considérée comme un objet document à part entière.

```
document.MyDIV.document.write ( " <A href='http://www.diderot.org' > Diderot  </A> ");
```

L'exemple ci-dessus insère un lien dans la balise *DIV*.

Manipuler une balise DIV avec Internet Explorer

IE contient un modèle objet plus développé en ce qui concerne les balises. En effet, l'objet *all* contient toutes les balises exceptées *HEAD* et *BODY*.

On accède donc à ces balises de la manière suivante :

```
document.all.MyDIV
```

Pour écrire dedans, il existe plusieurs méthodes donc la plus courante est *InnerHTML()* qui permet d'écrire directement du code HTML. Elle s'oppose à *InnerText()* qui elle n'écrit que du texte et concerne l'intérieur de la balise en question *(Pour plus de renseignements, se reporter au MacFarlane)*.

```
document.all.MyDIV.InnerHTML = "< A href='http://www.diderot.org' > Diderot  </A> ");
```

L'exemple ci-dessus insère un lien dans le bloc DIV nommé *MyDIV*.

Les Cookies

Côté client, le cookie est **fichier texte** géré par le navigateur. Les informations qu'il contient sont disponibles pour Javascript qui est libre de les manipuler. Les cookies représentent un danger relatif puisque n'importe quelle page HTML d'un site web peut déposer sur le client un cookie. On en déduit qu'un programme suffisamment bien fait aurait le pouvoir d'examiner tous les cookies d'un

client pour déterminer les sites que l'utilisateur a visité.

En fait, il existe autant de fichiers que d'URL différentes. Si un utilisateur consulte plusieurs sites et que tous ces sites gèrent les cookies, alors , il y aura autant de fichiers cookies que de sites visités.

Un cookie est constitué de :

- ✓ Un nom : Il s'agit d'une chaîne de caractères désignant le nom du cookie
- ✓ Une valeur : Chaîne de caractère unicode (Voir méthodes *escape* et *unescape*)
- ✓ Un domaine : Nom de domaine permettant de limiter la visibilité à un domaine spécifique. C'est la garantie qu'un autre site ne pourra pas consulter vos cookies.
- ✓ Un chemin d'accès : Permet de limiter le cookie à une partie du domaine spécifié ci-dessus.
- ✓ Un délai d'expiration : Fixe le délai de nettoyage des cookies par le navigateur. Sans celui-ci, le nombre des cookies pourrait augmenter et poser des problèmes de place. Si ce délai n'est pas fixé, le cookie sera effacé dès que le navigateur est fermé.
- ✓ Un drapeau de sécurité : Attribut valant *TRUE* ou *FALSE* qui force le navigateur à faire une requête *HTTP* sécurisée (Type SSL) si il a la valeur *TRUE.*

Les navigateurs possèdent des objets *cookie* que l'on peut consulter. Il ne faut pas oublier que les cookies dépendent de leur URL. Un site quelconque ne peut théoriquement pas consulter les cookies des autres sites.

Cookies Pourquoi faire ?

Les cookies sont utiles pour des application Web demandant un niveau de sécurité moyen. Ils sont employés dans les cas suivants :

- ✓ Pour des authentifications sur des sites dont certaines pages sont réservées aux membres
- ✓ Pour des authentifications sur des sites bancaires pour consulter des informations confidentielles (Appui supplémentaire de la technologie SSL)
- ✓ Pour des sites désireux de laisser des traces de passage à des fins commerciales
- ✓ Pour des sites effectuant des statistiques sur le nombre de visites
- ✓ Bien d'autres cas encore

Créer un cookie

La création d'un cookie requiert une bonne connaissance de la manière dont le navigateur fixe les *cookies* . La première chose à retenir est que un *cookie* n'est sauvegardé que si son délai d'expiration est fixé. Dans le cas contraire, il est effacé par le navigateur dès que celui-ci est fermé.

Voici un exemple de fonction permettant la création de cookie :

```
//  nom : Chaîne de caractère      valeur : valeur du cookie
// expiration : exprimée en milliseconde     chemin : du style /root/sdir1/sdir2
// domaine : nom de domaine auquel se limite le cookie
// securise : TRUE ou FALSE

function SetCookie( nom , valeur , expiration , chemin , domaine , securise )

{
var expir = "expires=" +  expir.
document.cookie = name + "=" + escape (value) +
}
```

L'exemple précédent utilise l'objet *cookie* de l'objet *document*. La chaîne de caractères contient tous les cookies pour l'URL courante séparés par des point virgules. Il faut prévoir un programme permettant de décoder les cookies.

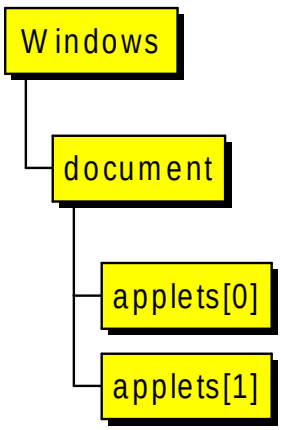
Heureusement, ce travail a déjà été fait et le fichier *cookies.js* contient toutes les fonctions permettant de les traiter *(GetCookie() , SetCookie() ...)*.

Javascript et Java

JAVA diffère de JAVASCRIPT pour la simple raison que l'un est un langage compilé et que l'autre est un langage interprété. Ils ne correspondent pas aux mêmes besoin. JAVA est idéal lorsque les besoins en sécurité sont importantes et JAVASCRIPT est parfait pour de petites applications WEB interactives. Le but de cette partie n'étant pas de philosopher sur les avantages et inconvénients des 2 langages mais plutôt de décrire la façon de les faire communiquer.

Points de connexion

Dans le modèle objet du navigateur, les objets JAVA possèdent leurs méthodes et propriétés que l'on peut exploiter en JAVASCRIPT.



Les APPLETs sont disponibles par l'intermédiaire de l'objet *document* du modèle objet. Il est possible de passer des paramètres dont le type subira quelques conversions.

Le principe de conversion dépend du type de paramètre passé. par exemple lorsque JAVASCRIPT envoie une variable de type *Number* , JAVA convertit cette valeur en un *FLOAT*.

Faire communiquer JAVASCRIPT et JAVA par l'intermédiaire des balises permet d'accéder aux paramètres des Applets et de les changer dynamiquement. La communication ne s'arrête pas là puisqu'il est possible d'accéder aux variables internes et aux méthodes et propriétés de l'applet.

Pour peu que les données transmises soient compatibles, le programme Javascript pourra executer ces propriétés et méthodes sans soucis.

Voici un exemple :

```
<HTML> <HEAD>
<SCRIPT >
document.MyApplet.dire_bonjour();
</SCRIPT>
</HEAD>

<BODY>
<APPLET name="MyApplet"  code="bonjour.class"  width=200 height=100 > </APPLET>
</BODY>
</HTML>
```

Dans cet exemple, le script fait appel à la méthode de l'applet JAVA *dire_bonjour()*.

JAVA, un langage exigeant

Lorsqu'on fait appel à une méthode d'un objet d'une applet JAVA, la machine virtuelle effectue les conversions nécessaires des données JAVASCRIPT envoyées en données typées utilisables par JAVA. Tant que la conversion est possible, tout se passe pour le mieux. Mais dès le nombre d'arguments passés est incorrect ou que la conversion n'est pas possible, le navigateur renvoie une erreur et l'applet ne peut s'executer.

Voici un récapitulatif des possibilités de conversion de données :

Javascript	JAVA
Types primitifs	

Undefined	Impossible à convertir
Null	*String* ou *Object*
Boolean	Boolean
Number	Float
String	String

Conversion de type de JAVASCRIPT vers JAVA

Références

Voici quelques références à consulter pour en savoir plus :

- ✓ Le guide du programmeur en Javascript - *Eyrolles – MacFarlane*
- ✓ Norme ECMAScript – Fichier *Ecma-Script262.doc*
- ✓ Document Object Model – Fichier *DOM.pdf*
- ✓ Dynamic HTML avec Netscape 4 – Fichier *dynamicHTML_NC4.pdf*
- ✓ Référence Javascript Netscape - *ClientReferenceJavascriptNCS13.pdf*
- ✓ Sites web à consulter – *Répertoires sites web du CD*
- ✓ Langage HTML 4 – *Le langageHTML2.doc*
- ✓ Norme W3C sur HTML – *html40-fr.zip*