Collection Ressources Informatiques

AJAX

Développez pour le Web 2.0

Entrez dans le code :
JavaScript, XML, DOM, XMLHttpRequest...

Luc VAN LANCKER

INFORMATIQUE TECHNIQUE

AJAX

Développez pour le web 2.0

Luc VAN LANCKER

Résumé

Ce livre s'adresse à des lecteurs avertis qui souhaitent découvrir ce qui se cache derrière le terme très médiatisé d'AJAX et ce qu'apporte cette nouvelle méthode de conception de sites Web. Il sera également très utile aux développeurs professionnels pour leur donner une perception globale d'AJAX avant qu'ils n'abordent les kits de développement (framework) utilisés dans leur environnement de travail habituel. Il suppose des connaissances préalables en XHTML et CSS. Son objectif est d'expliquer de façon simple et néanmoins précise, les divers composants d'AJAX et de montrer comment leur utilisation conjointe peut aboutir à des applications Web novatrices. Après une présentation générale d'AJAX, l'auteur détaille **JavaScript** (notions fondamentales, fonctions et méthodes, conditions et boucles, gestionnaire d'évènements, formulaires, manipulation des chaînes de caractères, tableaux...), le **XML** (présentation, syntaxe ...), les **XSL**, le **DOM** (présentation, les nœuds, l'accès aux objets, aux attributs...), l'objet **XMLHttpRequest** (présentation, propriétés et méthodes...). Les deux derniers chapitres proposent la mise en application des composants étudiés dans les chapitres précédents pour réaliser de l'AJAX ainsi que des exemples de développements. Cet ouvrage est écrit dans un style concis et précis avec de nombreux exemples significatifs et illustrations pour donner au lecteur une vision juste des possibilités d'AJAX. Les exemples de code utilisés dans l'ouvrage sont en téléchargement sur le site de l'éditeur.

L'auteur

Depuis les débuts d'Internet, **Luc Van Lancker**, enthousiasmé par l'idée de communication universelle que véhicule ce concept, s'investit complètement dans ce domaine. Après le HTML, le XHTML, les CSS, il était naturel qu'il se passionne aujourd'hui pour l'AJAX. Formateur expérimenté et grand pédagogue, il a su transmettre au lecteur toute sa passion pour cette nouvelle technologie.

Principe et définition

Le terme Ajax est apparu pour la première fois le 18 février 2005, dans un essai de James Garret intitulé "Ajax : une nouvelle approche pour les applications Web".

AJAX est un acronyme d'***A****synchonous* ***J****avaScript* ***A****nd* ***X****ML* (JavaScript et XML asynchrone) qui désigne une approche innovante dans la conception de pages Web dont l'objectif est d'optimiser leur interactivité et leur confort d'utilisation.

AJAX n'est pas une technologie nouvelle, c'est un terme synthétique qui désigne l'utilisation conjointe dans les pages Web de différentes technologies. Ainsi AJAX incorpore :

- le Xhtml et les feuilles de style CSS ;
- le JavaScript ;
- le Document Object Model (DOM) ;
- l'objet XMLHttpRequest ;
- le XML et le XSL.

Le Xhtml et les feuilles de style CSS prennent en charge la présentation des pages de façon standardisée.

Le JavaScript, qui marque ainsi son grand retour dans l'univers de la publication sur le web, est omniprésent dans les applications AJAX.

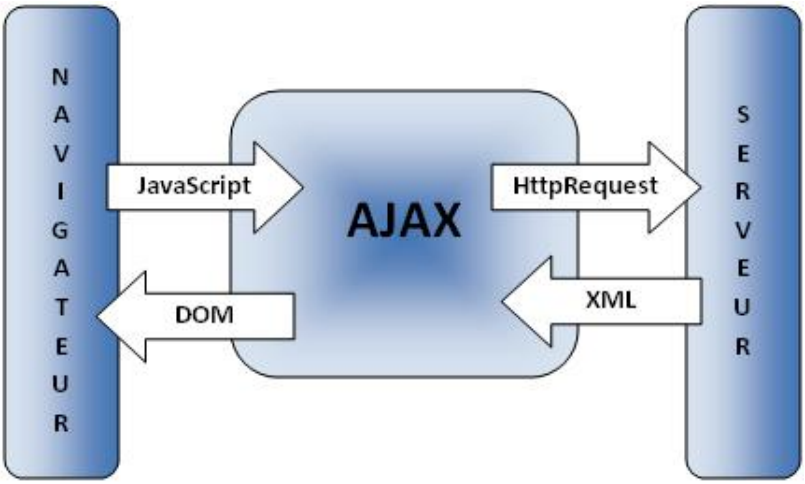
Les objets du document sont pris en charge selon le Document Object Model (le DOM) du consortium W3C.

L'objet XMLHttpRequest permet de lancer des requêtes de la page Web vers le serveur pour y récupérer des fichiers. Son fonctionnement permet d'effectuer ces requêtes de façon asynchrone, soit en arrière-plan de la page et de façon complètement transparente pour l'utilisateur.

Les fichiers récupérés sont au format XML, les apports de ce format dans le domaine de l'échange des données n'est plus à présenter.

Il s'agit donc de techniques, du JavaScript au XML en passant par le DOM et l'objet `XMLHttpRequest` qui sont éprouvées, standardisées et maîtrisées. Ce qui est assez prometteur quant à la compatibilité d'AJAX avec les navigateurs actuellement utilisés sur le Web.

Mais c'est dans l'utilisation conjointe de ces différentes technologies que réside l'originalité du fonctionnement d'AJAX que nous appellerons par la suite, le modèle AJAX.

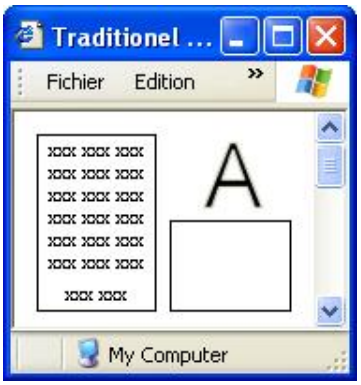


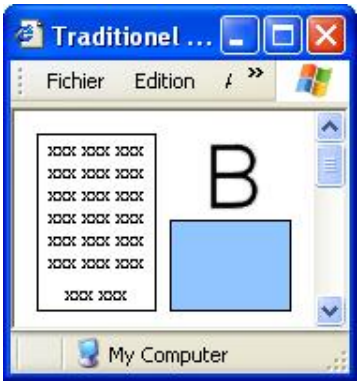
Ce modèle AJAX se décompose comme suit :

- La page Web s'affiche, comme à l'accoutumé, dans le navigateur, appelé aussi le client.
- La moindre interaction de l'utilisateur, par exemple l'encodage d'un formulaire ou le cliquer/déplacer d'un élément, est prise en charge par la gestion des événements de JavaScript.
- Le code JavaScript de la page initie une requête `XMLHttpRequest` vers un fichier XML situé sur le serveur.
- Le fichier XML réclamé, est renvoyé vers le navigateur de l'utilisateur.
- Il est alors pris en charge par le DOM, et toujours par le JavaScript, pour être traité et affiché de façon dynamique dans la page initiale.

Toute cette procédure s'est déroulée de façon transparente pour l'utilisateur, sans nécessiter le rechargement fastidieux de la page.

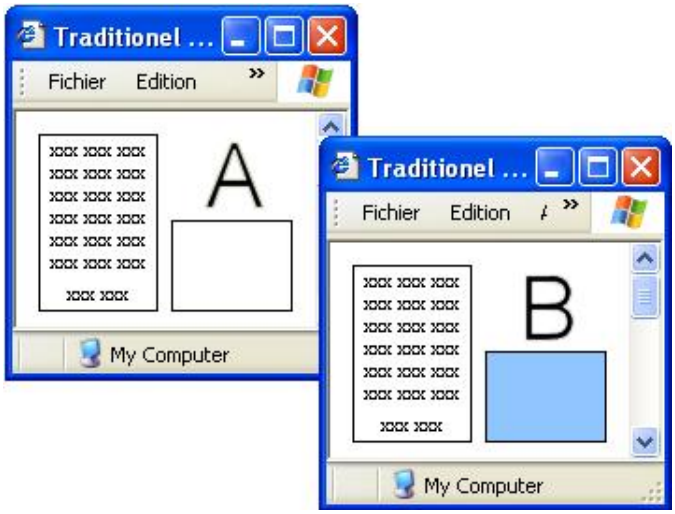
Le modèle AJAX permet ainsi de sortir du schéma traditionnel de la conception de pages Web qui necessitait que chaque nouvelle information réclame une nouvelle page.





Avec AJAX, une nouvelle information peut être récupérée dans un petit fichier situé sur le serveur. Ce fragment d'information peut alors être pris en charge et affiché dans une zone de la page et compléter ainsi la page initiale.

En évitant l'actualisation de la page et la perte de temps que cela entraîne, les applications gagnent ainsi en fluidité. En outre l'ergonomie s'en trouve améliorée car l'utilisateur peut rester concentré sur la lecture de la page ou la tâche accomplie.

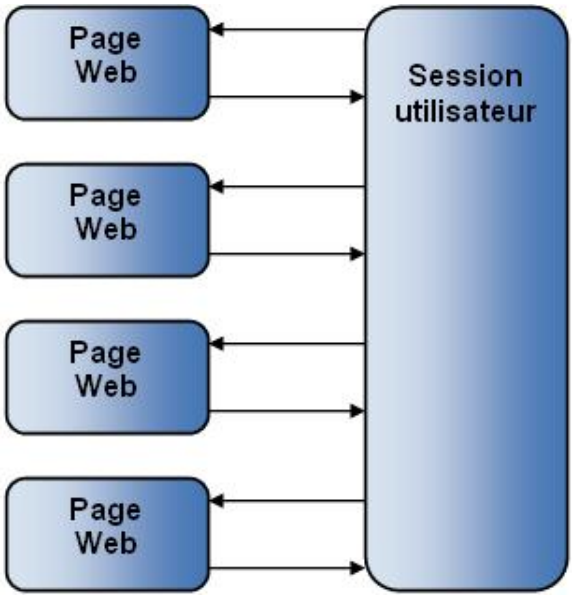


Le schéma traditionnel de la relation client-serveur, mis en place dès les origines du Web par la caractéristique hypertexte des pages Web, est ainsi mis en question. Selon celui-ci, la plupart des interactions de l'utilisateur comme le clic sur un lien ou l'envoi d'un formulaire, déclenchent une requête HTTP vers le serveur Web. Ce dernier, après le traitement éventuel des données, retourne une nouvelle page Html ou Xhtml au client.

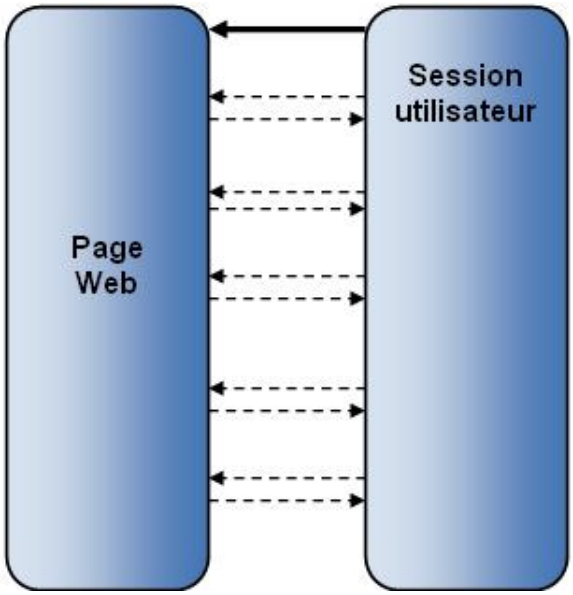
La session de l'utilisateur est alors une succession de requêtes HTTP et de réponses du serveur. Cet aller-retour incessant du client vers le serveur affiche à chaque fois une nouvelle page dans le navigateur.

Avec le schéma AJAX, la page Web initiale comprend une part importante de code AJAX qui permet, soit de prendre en charge une série d'opérations comme la validation de formulaires, soit d'initier des requêtes `XMLHTTPRequest` pour toutes informations complémentaires nécessaires. Les données ainsi retournées viennent s'incorporer de façon asynchrone dans la page initiale en apportant une facilité d'utilisation et une réactivité inconnues à ce jour sur la toile.

Le schéma traditionnel



Le schéma AJAX



La puissance du concept AJAX est telle que l'on peut envisager des applications Web qui se rapprochent des applications logicielles, appelées aussi applications de bureau. Ce qui modifie complètement la richesse de l'interface et la potentialité des services des pages Web.

Mais l'enjeu d'AJAX va plus loin que ces considérations techniques. Son fonctionnement réactif permet de reconsidérer totalement l'ergonomie des sites Web. Alors que pendant près d'une décennie, la réalisation de sites plaisants à l'œil était d'actualité, à présent la conception de sites plaisants à être utilisés est envisageable. L'utilisateur devient ainsi l'élément central des préoccupations des développeurs.

La définition d'AJAX n'est cependant pas à prendre à la lettre. L'approche AJAX est dans la pratique plus souple. Soulignons par exemple, que l'objet `XMLHttpRequest` permet de récupérer, non seulement des fichiers XML, mais aussi de simples fichiers de texte (au sens informatique du terme). Cette caractéristique élargit considérablement le champ des possibilités offertes aux développeurs.

Le concept AJAX est mis en avant et soutenu par Google. De nombreuses applications signées Google en reprennent l'approche. On songe ainsi à Google Maps et Google Suggest que nous allons aborder au point suivant. Il met également à la disposition des concepteurs un kit de développement (en open source). Le fait qu'une figure emblématique du Web comme Google s'investisse ainsi dans les applications AJAX, ne peut que susciter ou renforcer l'intérêt des développeurs de sites Web.

Exemples sur le Web

1. Google Maps

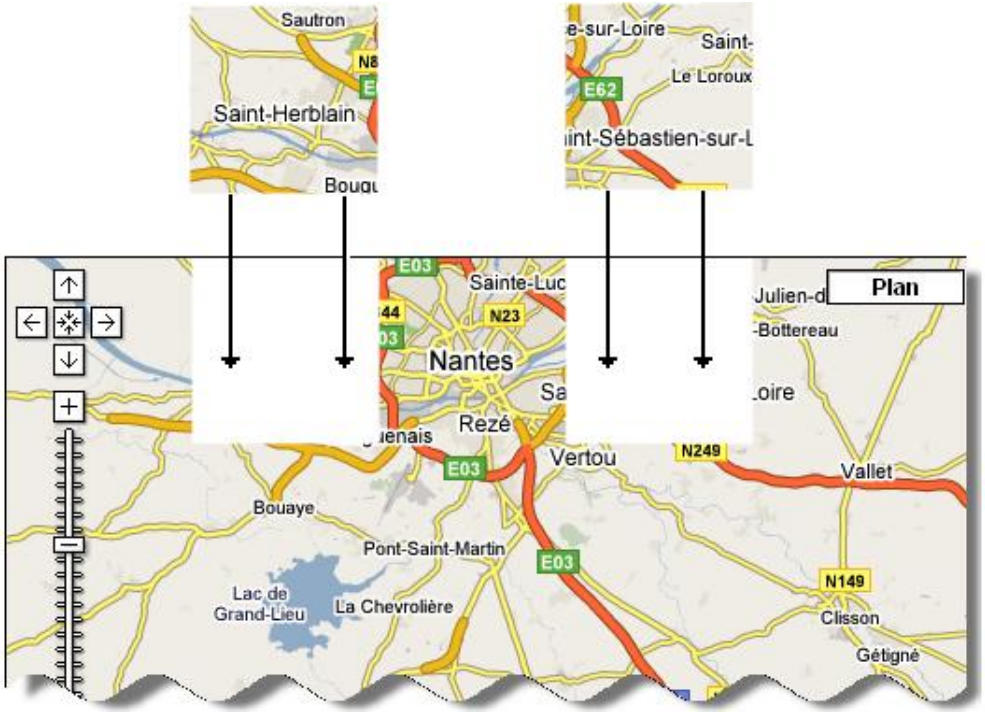
Google Maps est un service en ligne (gratuit) de cartes géographiques, calcul d'itinéraires et de vues satellites.

Ce site permet, à partir de l'échelle d'un continent ou d'un pays, de zoomer par un double clic ou par l'utilisation de la roulette de la souris, jusqu'à l'échelle d'une rue. On peut aussi basculer d'un simple clic, de la carte classique avec le nom des rues, des quartiers et des villes à des images satellites de la même localisation.

Il peut être consulté, dans sa version française, à l'adresse http://maps.google.fr/.

En quoi le site Google Maps reprend-il l'approche AJAX ?

Il faut d'abord savoir qu'une carte est divisée en petits carrés. Pour les cartes classiques, ces petits morceaux au format PNG, ont une dimension de 256 x 256 pixels, et un nombre de couleurs limité à 256. Ce qui en fait des fichiers de taille réduite et ce qui implique un temps de transfert très faible.



Lorsque l'utilisateur déplace la carte dans un sens, par le simple glissement du curseur de la souris, cette action est prise en charge par le JavaScript. Celui-ci récupère la longitude, la latitude et l'intensité du zoom des carrés qui devront être dévoilés. L'objet `XMLHttpRequest` envoie une requête vers le serveur pour réclamer les petites images manquantes. À la réception, le DOM se chargera de les afficher dynamiquement à l'endroit adéquat, parfaitement alignées les unes avec les autres.

Toute cette opération se fait sans rechargement de la page, sans modification de l'adresse de la page, sans apparition de la barre d'avancement du téléchargement dans la barre d'état et sans réclamer l'installation d'un plug-in spécifique !

2. Google Suggest

L'interface est proche de celle de la page d'accueil habituelle de Google. Cependant, cet outil suggère automatiquement une liste de 10 mots ou expressions qui se rapprochent de ce qui est encodé dans la zone de texte consacrée aux mots-clés. Cet outil affiche également en face de chacun des mots ou expressions, le nombre de résultats trouvés dans la banque de données de Google. L'utilisateur peut ensuite effectuer une sélection dans cette liste de suggestions et continuer sa recherche sur Google.

À chaque encodage d'une lettre, cette liste de 10 expressions est recréée et affichée de façon dynamique.

Ce site en version beta est consultable à l'adresse labs.google.com/suggest/.

La vitesse de fonctionnement est assez surprenante. En effet, Google Suggest dialogue avec les serveurs de Google pendant que l'internaute tape sa requête et met à jour l'affichage de façon quasi instantanée, et toujours sans nécessiter un rafraîchissement de la page.

Le fonctionnement d'AJAX dans cette application peut être décomposé comme suit :

- L'encodage dans la zone de texte est récupéré par JavaScript.
- Une requête XMLHttpRequest est initiée avec la valeur de l'encodage comme paramètre (ici le mot `AJAX`).
- Le serveur retourne les informations demandées sous forme de deux tableaux (Array) Javascript : `new Array ("ajax tutorial", "ajax tutorials", "ajax examples", "ajaxian", "ajax framework", "ajax php", "ajax example", "ajax4jsf", "ajax asp.net", "ajax amsterdam"), new array("25,800,000 results", "51,400,000 results", "3,790,000 results", "2,230,000 results", "33,900,000 results", "138,000,000 results", "17,800,000 results", "199,000 results", "9,170,000 results", "3,530,000 results").`
- Le DOM entre alors en action pour créer une liste déroulante de 10 lignes. JavaScript se charge quant à lui de répartir les données du tableau `Array` dans chaque ligne.

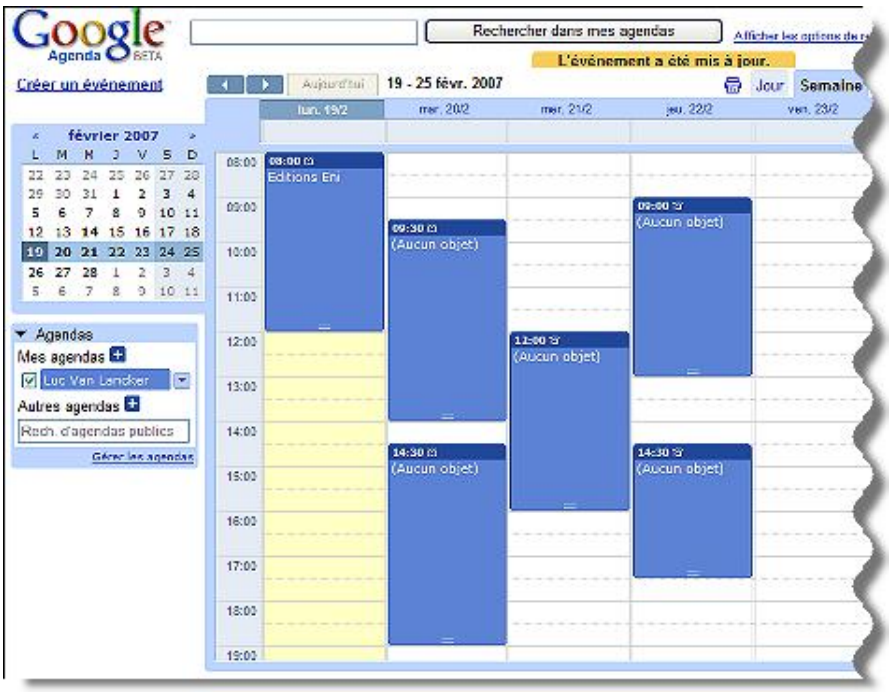
Nous proposons dans le chapitre Ajax par l'exemple, un exemple appelé "Une suggestion d'encodage" qui s'inspire largement de la façon de procéder de Google Suggest.

3. Google Agenda

Google Agenda, comme son nom l'indique, permet de gérer votre emploi du temps en toute simplicité, de façon très conviviale et colorée. Encoder des événements, modifier votre planning d'un simple cliquer/glisser de la souris, partager (avec votre permission) votre agenda, de disposer de rappels d'événements par un e-mail, par une fenêtre pop-up ou par un message sur votre téléphone portable, voici quelques- unes des fonctionnalités, très variées, de Google Agenda.

Une démonstration de ce nouveau service en ligne est disponible à l'adresse : http://www.google.com/intl/fr/googlecalendar/tour.html.

Le service en lui-même est accessible à partir de la page d'accueil de Google et nécessite l'ouverture (gratuite) d'un compte.



Le nombre important de fonctionnalités de Google Agenda illustre bien la richesse des applications AJAX : celles-ci permettent de mettre en place des sites qui n'ont plus rien à envier aux applications logicielles.

4. Google Document et Tableur

Créer, éditer, ou encore partager des documents demande toujours l'installation sur notre système d'un logiciel, en général Microsoft Office ou OpenOffice. Et si on pouvait faire tout cela avec seulement un navigateur Web ?

Google avec Document et Tableur, propose une solution très alléchante qui va nous permettre de disposer d'une application bureautique, accessible depuis n'importe quel poste relié à l'Internet, sans aucune installation à effectuer.

Google Document et Tableur permet donc de créer rapidement des documents, et surtout de les partager pour faciliter le travail collaboratif.

Le tout peut être enregistré au format Html, RTF, Word, OpenOffice et PDF.

Et ce n'est pas seulement un traitement de texte qui est à votre disposition mais aussi un tableur, avec la mise en forme de formules, même complexes !

Des bruits circulent qu'un outil de présentation, du même style que PowerPoint, de la suite bureautique Office de Microsoft, serait bientôt disponible.

Une visite guidée vous attend à l'adresse :

http://www.google.com/google-d-s/intl/fr/tour1.html.

Le service en lui-même est accessible à partir de la page d'accueil de Google et nécessite l'ouverture (gratuite) d'un compte.

Avantages et inconvénients

Il ne faut pas se laisser aveugler par l'effet AJAX. Tous les sites ne se prêtent pas à cette approche. Surtout que développer un site en AJAX sera plus difficile et plus long que les méthodes de publication utilisées à ce jour. Pour vous aider dans votre choix, voici quelques éléments qui vous aideront à prendre la bonne décision.

1. Avantages

- AJAX, c'est principalement la révolution asynchrone ! La technique qui consiste à charger en arrière-plan des éléments de réponse aux interactions de l'utilisateur, est au cœur du modèle AJAX. Il est certain que le potentiel est énorme et qu'il n'est à ce jour à peine exploré. Gageons que la créativité des développeurs y trouvera un terrain où elle pourra pleinement s'exprimer.

- L'interface est plus réactive car seulement une petite partie est mise à jour, donnant ainsi l'impression à l'utilisateur que les changements se réalisent instantannément.

- Le temps d'attente de l'utilisateur est réduit car il n'est plus nécessaire de recharger toute la page pour obtenir une nouvelle information. Les applications AJAX apportent ainsi une fluidité inhabituelle aux pages Web. Certains qualifient même AJAX comme étant le Web des impatients. Je gage que certains lecteurs vont se reconnaître...

- Le temps de transfert entre le client et le serveur est considérablement réduit car il lui suffit de transférer les éléments de la réponse au lieu d'envoyer toute la page avec des éléments le plus souvent redondants.

- Un site AJAX bien réalisé sera agréable d'utilisation pour le visiteur. Ce qui ne peut que valoriser en termes d'image la communication avec l'utilisateur.

- La maintenance du site sera plus aisée car, grâce à l'architecture mise en place par AJAX, il suffira de ne mettre à jour que des fichiers de taille réduite, parfois simplement en format texte. Cette mise à jour pourra même être confiée à des personnes dont les compétences sont plus orientées bureautique que conception de sites Web.

2. Inconvénients

- Même si Ajax utilise des composants connus comme le JavaScript, le XML, le DOM et l'objet XMLHttpRequest, son code est pointu et délicat à mettre en place. Le fait de développer une application AJAX peut augmenter sensiblement le temps de conception et donc le coût. Il est admis de façon unanime qu'AJAX est plus difficile à mettre en place qu'une application Web classique à cause de l'utilisation conjointe des différentes technologies.

- AJAX comporte beaucoup de JavaScript. Les utilisateurs qui ont désactivé la prise en charge du JavaScript par leur navigateur, n'ont pas accès aux applications AJAX. On peut raisonnablement estimer que cette population est faible mais elle existe.

- Le concept AJAX est orienté vers l'utilisateur. Cependant par divers aspects, il risque de désorienter l'utilisateur moyen. Comme certaines parties de la page ne correspondront plus à la page initiale, les fonctions de l'historique et de la mise en favoris du navigateur risquent de ne pas être représentatives de l'état actuel de la page. De plus, cliquer sur le bouton **Précédent** du navigateur, peut n'avoir aucun effet puisque l'adresse de la page est restée inchangée.

- L'accessibilité des applications AJAX avec les systèmes de lecture d'écran utilisés par les personnes visuellement déficientes est assez problématique. Déjà la prise en compte du JavaScript est très partielle et souvent aléatoire. En outre, vient s'ajouter l'objet XMLHttpRequest qui modifie, en cours de lecture, le contenu de la page. Ainsi, au moins dans un premier temps, il importe de prévoir une page alternative pour les personnes ayant un handicap visuel.

- Comme l''objet `XMLHttpRequest` peut envoyer des requêtes du navigateur vers le serveur de façon transparente pour l'utilisateur, certaines personnes s'inquiètent quant au respect de la vie privée de l'internaute. On peut comprendre leur point de vue mais Internet est et restera un système ouvert dans toutes les implications du terme. La limitation de l'objet `XMLHttpRequest` qui ne peut envoyer ou recevoir des requêtes

que vers le domaine du site Web qui contient la page, pourra (partiellement) les rassurer.

- Il est difficile à l'heure actuelle (courant 2007) d'avoir un avis tranché en matière de sécurité informatique des applications AJAX. Comme toujours lors de l'apparition d'une nouvelle technologie, des détracteurs se font entendre tandis que d'autres se veulent rassurant. Il est un fait que le concept AJAX se base sur des composants connus et que seul le nombre de requêtes entre le navigateur et le serveur risque d'augmenter dans les applications AJAX. Pour le reste, il est simplement trop tôt pour se prononcer.

- Le référencement des pages en AJAX n'est pas simple à réaliser. Comme pour les sites en JavaScript et les pages en Flash, le référencement réclamera toutes les attentions du concepteur. L'indexation par les moteurs de recherche n'est pas moins problématique car avec le système AJAX, une même page peut, selon les actions de l'internaute, avoir plusieurs contenus différents.

Évolution ou révolution

S'il y a une question qui a alimenté, et qui alimentera encore, les forums de discussion, c'est bien celle-là. Tentons de faire le point au sujet des arguments développés.

1. Évolution

Il est évident qu'AJAX est une façon d'utiliser plusieurs technologies préexistantes.

Le JavaScript, omniprésent dans AJAX, est ce vilain petit canard de la publication sur le Web. Inventé en 1996 par Netscape, pauvrement normalisé par le ECMAScript, avec ses bugs et ses incompatibilités, il marque pourtant avec le concept AJAX son retour sur le devant de la scène.

L'utilisation conjointe des feuilles de style CSS, du JavaScript et du DOM sur fond de page Html ou Xhtml n'est pas sans rappeler la définition du DHTML qui lui aussi prônait déjà l'interactivité des pages Web. L'objet `XMLHttpRequest` ne fait qu'apporter au Dhtml la possibilité de récupérer des données du serveur.

Et en ce qui concerne l'objet `XMLHttpRequest`, il n'est pas récent lui non plus car il était déjà été implanté dans Internet Explorer 5.0 en 1999 et passa tout ce temps inaperçu auprès des développeurs.

AJAX et le Web 2.0 ne seraient alors qu'une évolution naturelle de ce formidable outil qu'est la Web après 15 années d'existence. Et le Web, qui s'est implanté dans la vie quotidienne, connaîtra encore bien des évolutions ou des mutations.

2. Révolution

Pourtant on ne peut que s'étonner, au-delà du battage médiatique, du formidable mouvement de synergie créé autour de l'approche AJAX par les professionnels du Web et d'Internet. La mise en place en février 2006 de OpenAJAX Alliance, consortium de firmes réputées comme Adobe, Backbase, BEA, DoJo Foundation, Eclipse Foundation, Google, IBM, Novell, Oracle, SAP et Red Hat pour promouvoir la technologie AJAX en est une preuve.

À la vue des exemples présentés lors du présent chapitre, il faut bien admettre que la mutation des pages Web traditionnelles en véritables plates-formes applicatives va modifier, de façon significative, les usages de la toile. Avec la richesse des interfaces, avec la multiplication des fonctionnalités, avec les avancées de l'ergonomie et de l'usabilité des sites, c'est assurément une nouvelle façon de naviguer qui s'instaure.

La véritable nouveauté d'AJAX ne réside pas tant dans son aspect technologique mais bien dans la nouvelle stratégie mise en place. C'est la première fois depuis l'apparition du Web que l'utilisateur, ou autrement dit le consommateur, est placé au centre des préoccupations des développeurs. Ajax permet la conception de sites réactifs, ergonomiques, intuitifs, axés sur le consommateur du Web et ses nouvelles attentes d'utilisation du Web.

Pré-requis

L'objectif de cet ouvrage, après avoir parcouru les éléments nécessaires, est de faire percevoir au lecteur le fonctionnement concret et les énormes potentialités des applications AJAX.

Force est de constater après la lecture du chapitre Présentation générale d'AJAX que du Xhtml en passant par les feuilles de style CSS, le JavaScript, le XML, le DOM et l'objet `XmlHttpRequest`, les différentes technologies ne manquent pas.

Afin d'éviter de proposer une encyclopédie de la conception de pages Web, il semble judicieux de limiter les matières abordées dans ce livre. À ce stade de l'intérêt du lecteur pour les nouvelles perspectives de la publication sur le Net, il paraît raisonnable de penser que celui-ci possède déjà des notions basiques ou avancées du langage Html ou Xhtml ainsi que des feuilles de style CSS.

AJAX met en œuvre, à travers différents composants comme le DOM et l'objet `XmlHttpRequest`, un code pointu et a priori complexe qui réclamera toute votre attention. Une certaine aisance dans le langage Html et les feuilles de style CSS ne peut que faciliter l'étude du lecteur.

Pour parfaire vos connaissances sur le Xhtml et les feuilles de style CSS, vous pouvez vous reporter à l'ouvrage "XHTML 1 et CSS 1 et 2.1 - Les nouveaux standards du Web" aux Éditions ENI - Ressources Informatiques du même auteur.

La connaissance de langage Html ou Xhtml et des feuilles de style CSS est un pré-requis indispensable à une étude fructueuse de cet ouvrage consacré à AJAX.

Tous les exemples du livre sont écrits en Xhtml 1.0 de type transitional.

Limites de ce livre

La lecture du chapitre Présentation générale d'AJAX a fait découvrir que l'approche AJAX permettait de traiter dans le navigateur de l'utilisateur (application côté-client) des fichiers externes à la page, présents sur le serveur (application côté-serveur).

Les données de ces fichiers externes peuvent être traitées par des langages de programmation comme par exemple, le PHP et l'ASP. Ce qui élargit encore, et de façon considérable, les compétences réclamées pour produire des pages Web en AJAX. À ce niveau, la frontière entre le lecteur averti et le professionnel nous semble franchie.

Ainsi, nous limiterons l'étude de cet ouvrage sur AJAX à des applications côté-client qui récupéreront des fichiers externes, au format XML ou texte, et qui les afficheront de façon asynchrone dans la page Web.

Par ailleurs, la vague d'intérêt suscitée par AJAX fait fleurir de nombreux assistants logiciels à la production d'applications AJAX, appelés frameworks, qui exploitent certains aspects répétitifs du code mis en place par celui-ci. Là aussi, il s'agit d'outils professionnels qui ne seront pas abordés ici.

Outils côté-client

1. Un navigateur Internet Explorer 7 ou 6 et Firefox

Au moment de l'écriture et de l'édition de cet ouvrage (courant 2007), il est raisonnable de se concentrer sur les trois navigateurs que sont Internet Explorer 6, Internet Explorer 7 et Firefox dans sa version 2.0.

Le vieillissant Internet Explorer 6, bien que contesté pour ses problèmes de sécurité et son fonctionnement particulier dans certains domaines des feuilles de style CSS, est encore présent sur de nombreuses machines.

La toute dernière version Internet Explorer 7, présente dans Microsoft Vista et disponible pour Microsoft XP SP2, suscite à ce jour un intérêt certain. D'autant que, proposée par le système de mises à jour automatiques de Microsoft, sa diffusion devrait être plus rapide que celle des versions antérieures d'Internet Explorer. Même s'il n'apporte pas de grandes innovations au niveau du code, on peut déjà mettre en avant une approche différente de l'objet `XmlHttpRequest` qui sera détaillée au chapitre L'objet XmlHttpRequest consacré à ce sujet.

Pour les lecteurs qui ont déjà adopté Internet Explorer 7.0 et qui souhaiteraient néanmoins tester leur code sous Internet Explorer 6.0, Ryan Parman de Skyzyx.com a regroupé le fichier iexplore.exe avec les fichiers du noyau du navigateur pour chacune des versions d'*Internet Explorer*. Ces versions autonomes (standalone) d'Internet Explorer 5.0, 5.5 et 6.0 sont disponibles sur sa page de téléchargement : http://www.skyzyx.com/downloads/

L'incontournable navigateur Firefox ne fait qu'accroître sa part de marché. Il n'est plus à présenter aux développeurs pour son respect rigoureux des standards édités par le W3C et ses nombreuses extensions.

Il existe bien entendu d'autres navigateurs de qualité que nous ne pouvons aborder dans le cadre de ce livre.

2. Un éditeur de texte

Les initiés au Html et à ses langages connexes n'ignorent pas l'importance du code source et de son encodage (manuel) dans un éditeur de texte.

Pour la suite de notre étude, n'importe quel éditeur de texte brut fera l'affaire. Pour les utilisateurs de Windows, l'utilisation du Bloc-notes (notepad) fera parfaitement l'affaire.

Outils côté-serveur

Si pour l'étude du JavaScript, du XML et du DOM, soit respectivement les chapitres Le JavaScript, Introduction au XML et Introduction au XSL, un éditeur de texte et votre navigateur suffiront largement, la suite du livre consacrée à l'objet XmlHttpRequest et à Ajax (chapitres Le DOM (Document Object Model), L'objet XmlHttpRequest, L'approche AJAX et AJAX par l'exemple) nécessite la mise en place d'un serveur Web local, aussi appelé serveur Web personnel ou virtuel.

Ce serveur local vous permettra de tester le code sans passer par la procédure contraignante de son téléchargement par FTP sur votre espace perso, évitant ainsi de se connecter à un serveur externe.

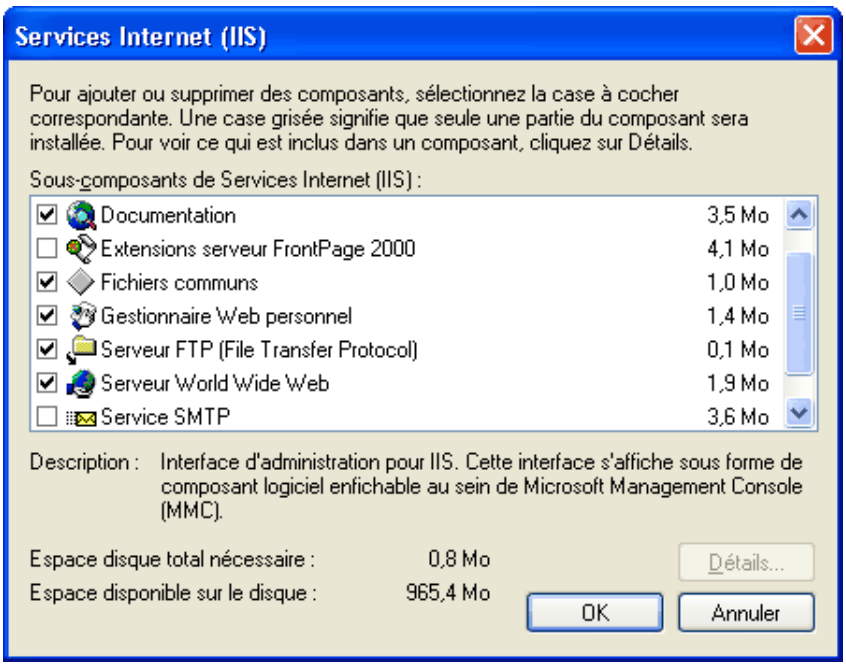
1. Le serveur local Microsoft IIS

Le logiciel serveur Web IIS (Internet Information Services) est présent sur le CD d'installation de Windows XP version Pro, Windows Vista et Windows Server 2003.

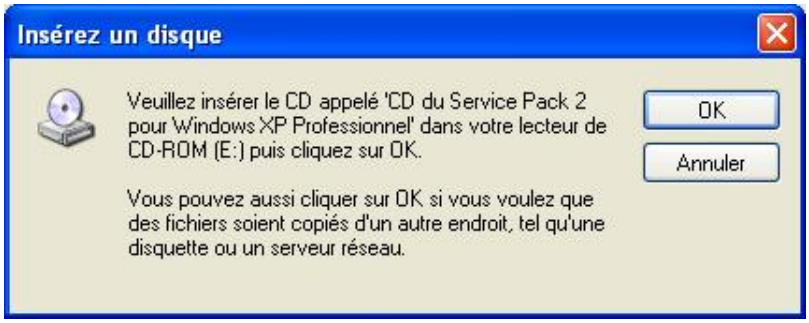
- L'installation d'IIS est assez simple : **Démarrer** - **Paramètres** - **Panneau de configuration** - **Ajout/Suppression de programmes**.

- Dans la fenêtre d'ajout/suppression de programmes, activez **Ajouter ou supprimer des composants de Windows**.

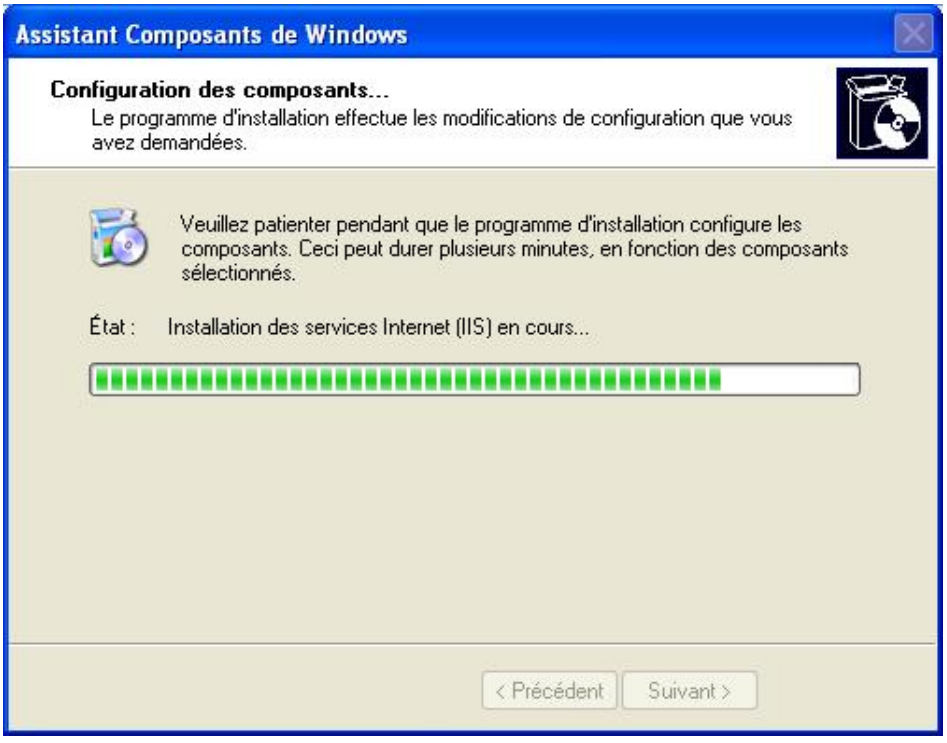
- Cochez la case **Services Internet (IIS)** et allez dans **Détails**...



- Cliquez sur **OK** pour lancer l'installation. Windows va vous demander le CD d'installation.



L'installation peut alors se poursuivre.



Le serveur IIS est maintenant installé.

Vous pouvez trouver sur le disque dur C, un nouveau dossier **Inetpub** contenant à son tour différents sous-dossiers.

Les fichiers du serveur local doivent être déposés ou enregistrés dans le dossier **Inetpub\wwwroot**.

Pour accéder à un fichier, par exemple index.htm, on encodera dans la barre d'adresse du navigateur localhost/index.htm.

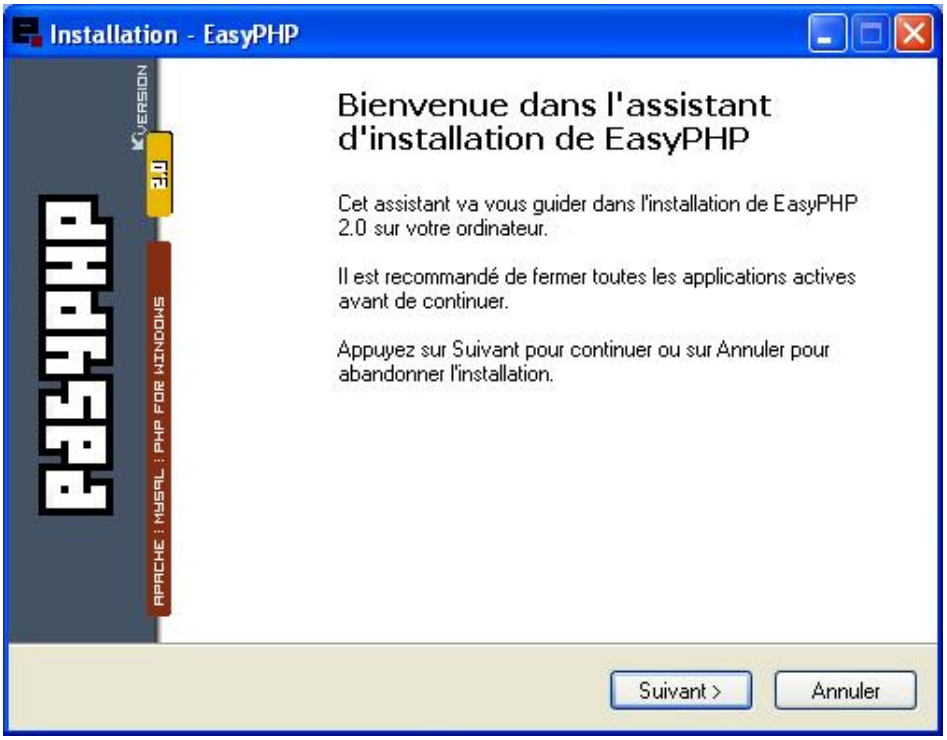
Si vous avez une version de XP édition familiale, vous ne pouvez pas installer IIS. Retenez dans ce cas, l'option **EasyPhP** présentée ci-après.

2. Le serveur local EasyPHP

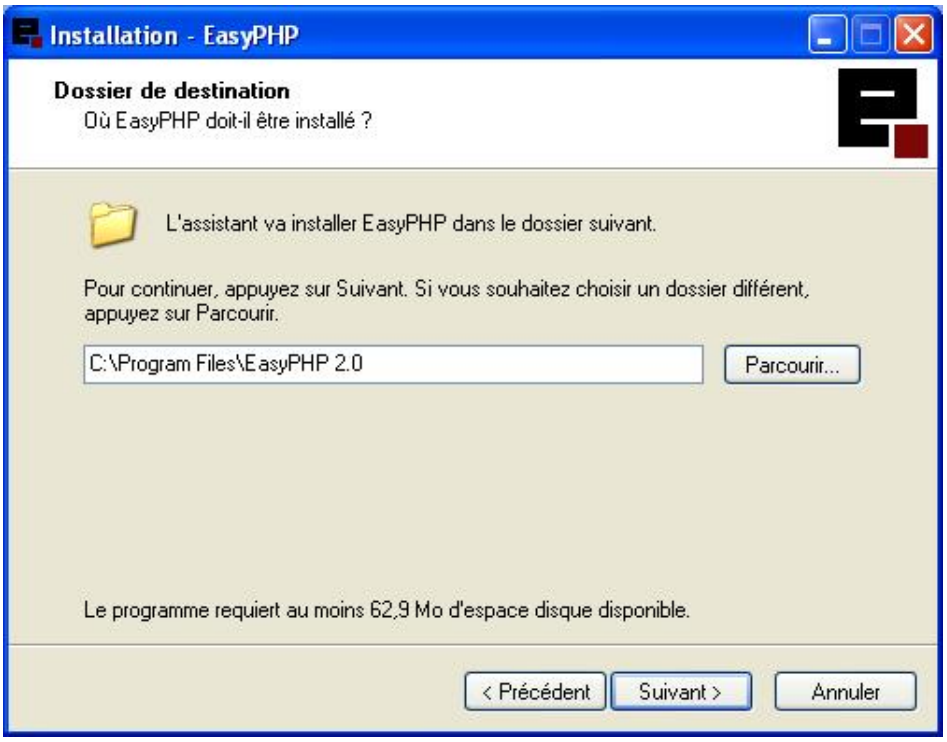
EasyPHP est une plate-forme de développement Web comprenant le renommé serveur Apache ainsi qu'un interpréteur de scripts PHP, le serveur de bases de données MySQL et l'outil d'administration PhpMyAdmin. Initialement prévu pour le développement de scripts PHP en local, son serveur Apache vous permet de tester en local vos applications AJAX.

EasyPHP est téléchargeable à l'adresse : www.easyphp.org/telechargements.php3. Précisons que EasyPHP est un outil qui a largement fait ses preuves et qu'il est entièrement gratuit.

Après avoir sélectionné la langue qui sera utilisée par l'assistant d'installation, celle-ci débute.



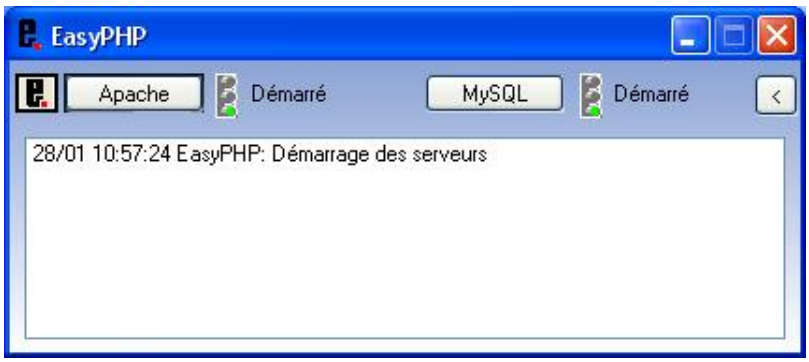
- La suite de programmes sera accessible sur le disque dur **C - Program Files -Easy PHP**.



L'installation est terminée.

- Les fichiers à tester devront être situés dans le sous-dossier **Program Files\EasyPHP\www**.

- Pour afficher ceux-ci dans le navigateur, il faut avant toutes choses démarrer le serveur local : **Programmes - EasyPHP - EasyPHP**.



On encode ensuite dans la barre d'adresse du fichier, par exemple index.php, localhost/index.php ou 127.0.0.1/index.php.

Introduction au JavaScript

1. Définition

JavaScript est un langage de script qui, incorporé aux balises Html ou Xhtml, permet d'agrémenter la présentation et l'interactivité des pages Web.

Détaillons et complétons cette définition.

Le JavaScript est un langage de script, distinct du langage Html. Le JavaScript est en quelque sorte un "petit" langage de programmation dont les lignes de code viennent s'ajouter au langage Html. Ce langage, inspiré plus ou moins directement du langage C, ne pose pas de problèmes de compréhension à ceux qui ont déjà touché à la programmation. Par contre, les utilisateurs issus de la bureautique ou de l'infographie risquent d'être un peu déroutés devant les éléments classiques de la programmation comme les variables, les instructions conditionnelles, les boucles, les fonctions et autres méthodes.

Le Html avec sa philosophie de structuration de contenu est essentiellement statique. Très rapidement, le besoin s'est fait sentir d'y ajouter un peu de mouvement et surtout de l'interactivité avec l'utilisateur. C'est ce que permet le JavaScript.

Citons pour exemple :

- animer du texte ou des images,
- réagir à l'action de la souris,
- détecter le type et la version du navigateur,
- vérifier la saisie dans les formulaires,
- effectuer des calculs simples,
- demander une confirmation,
- afficher la date et l'heure,
- gérer les menus de navigation,
- rediriger le visiteur vers une autre page,
- etc.

Les lignes de code du JavaScript sont gérées et exécutées directement par le navigateur lui-même, sans devoir faire appel aux ressources du serveur qui héberge la page. C'est ce qu'on appelle une application "côté client". Il existe d'autres langages, plus évolués comme le PHP ou l'ASP qui eux, traitent les informations sur le serveur avant de les renvoyer sous forme de page Html au navigateur de l'utilisateur. Ce sont les applications "côté serveur". Ces dernières sortent du cadre fixé à notre étude. Signalons simplement qu'il existe bien une version "côté serveur" de JavaScript mais que nous n'aborderons pas. Nous nous limitons donc à la version la plus répandue et la plus abordable, soit le JavaScript "côté client".

La tradition veut que l'on précise dans une introduction au JavaScript que celui-ci (malgré la confusion qui peut naître à partir du nom) n'a rien de commun avec le langage Java. Ce dernier, est un véritable langage de programmation plus performant et bien plus complexe dont le code est compilé sous forme d'applets Java. Celles-ci peuvent s'exécuter à partir d'une page Web pour autant que la Machine Virtuelle Java soit installée sur le poste du visiteur. Pour rappel, le code du Java-Script, inclus dans la page Html, est quant à lui directement "interprété" par le navigateur au moment de son exécution.

Conséquence directe de l'inclusion du JavaScript dans les pages html, le code JavaScript sera visible par une simple consultation du code source de la page. Ceci peut faire le désespoir des concepteurs débutants mais ne constitue pas vraiment un handicap étant donné que le JavaScript n'est destiné qu'à de petites applications, finalement bien connues de tous. Pour les applications plus professionnelles, celles-ci seront réalisées avec des langages plus évolués côté serveur. Ce qui préservera alors la confidentialité du code.

Le JavaScript est un langage orienté objet. Nous y reviendrons en détail plus avant dans notre étude. Notons

simplement que JavaScript permet d'accéder directement aux différents éléments du document et de les manipuler. Comme principaux éléments ou objets de la page Html, on peut citer la fenêtre du navigateur, le document Html, les images, les cadres, les formulaires, les éléments de formulaire ou tout autre élément que vous aurez au préalable identifié.

L'objectif de cette partie n'est pas de faire de vous des programmeurs émérites en JavaScript mais de fournir des bases solides qui vous permettront de comprendre le fonctionnement de la technique AJAX ainsi que les manipulations de données qu'elle induira.

2. Bref historique

Le langage JavaScript est apparu en 1995 avec le navigateur Netscape 2. Le JavaScript 1.0 était né. Les possibilités offertes par ce langage ont très vite séduit les utilisateurs et les créateurs du Web, créant à l'époque un vif intérêt sinon une petite révolution.

Du côté de chez Microsoft, alors en retard technologiquement dans le domaine de l'Internet, JavaScript a été intégré très rapidement au navigateur Internet Explorer 3. En effet, il ne fallait pas de licence pour utiliser le JavaScript. Par contre, dans le climat de guerre froide entre les deux constructeurs, il n'était pas question pour Microsoft d'acheter une licence à Netscape pour en adapter le code. C'est alors que JScript est né, variante du JavaScript selon Microsoft, qui reprend la plupart des fonctions du JavaScript et qui en élargit le champ d'application.

Les choses étaient mal parties sur le plan de la compatibilité. Heureusement, les deux firmes acceptèrent de participer à une standardisation. L'organisme choisi fut l'**ECMA** (European Computer Manufactures Association). En 1997, le JavaScript a été standardisé sous le vocable ECMAScript sous la référence ECMA-262. En fait, cette standardisation n'a porté que sur la syntaxe de base, n'apportant qu'une compatibilité minimale et chaque éditeur a continué ses développements indépendamment l'un de l'autre.

Avec Netscape 3 (fin 1996) est apparue la version JavaScript 1.1. Puis avec Netscape 4 (1997), la version 1.2. Ces versions furent également reconnues par les versions suivantes d'Internet Explorer tandis que JScript voyait lui aussi apparaître de nouvelles versions.

Commence alors le long déclin de Netscape. JavaScript lance encore des versions 1.3, 1.4 et 1.5 mais elles n'ont pas connu le retentissement des versions précédentes.

Parallèlement à l'évolution (sans révolution) du JavaScript, il y a un autre facteur de divergence. Nous avons vu, dans la définition, que le langage JavaScript était un langage orienté objet. La façon de définir et d'accéder à ces objets a également connu une approche différente selon l'éditeur du navigateur. Le W3C a réagi à cette incompatibilité an lançant le DOM (Document Object Model). Cette standardisation a été globalement bien suivie, mais il subsiste des différences notoires d'interprétation (voir chapitre Le DOM (Document Object Model)).

Devant ce bel imbroglio, il importe d'adopter quelques règles d'ordre pratique.

Globalement, JavaScript est bien reconnu par tous les navigateurs.

Dans la nébuleuse des différentes versions, la plupart des concepteurs ont choisi la version JavaScript 1.2 vraiment compatible.

Pour des applications assez basiques, le problème de compatibilité ne se posera pas.

Dès que l'on passe à des applications plus sophistiquées, on peut avancer que le JavaScript est hautement incompatible. À tel point qu'il faut parfois écrire un code adapté à chaque navigateur et éventuellement à chaque version de celui-ci !

3. Limites

Presque toutes les pages Web intègrent quelques éléments de JavaScript. Ce qui ne semble plus créer de réels problèmes de sécurité, d'où son succès auprès des créateurs de sites Web et des internautes.

Mais cette relative sécurité a un prix qui se traduit par de sévères limites techniques.

La principale limite de JavaScript est qu'il ne permet pas de lire et d'écrire sur le disque dur ou tout autre périphérique

du visiteur. La seule exception est celle qui permet à JavaScript de lire et d'écrire sur le disque dur uniquement dans la zone réservée aux cookies. C'est la seule interaction que JavaScript peut avoir avec votre disque dur.

En outre, aucune instruction n'est prévue en JavaScript pour générer une connexion vers un autre ordinateur ou un serveur. Cela n'est possible que par la création d'un objet XmlHttpRequest qui est un élément indispensable à la mise en place d'applications de type AJAX (voir chapitre L'objet XmlHttpRequest).

Il ne sera donc pas possible de concevoir en JavaScript des applications telles qu'un forum de discussion, un sondage ou une boutique en ligne. Toutes ces applications nécessitent un langage plus puissant, généralement implanté côté serveur, comme le PHP ou l'ASP.

Même si cela est en partie excessif, il faut être conscient que le JavaScript est un "petit" langage de programmation et qu'il n'a été conçu que pour compléter le HTML normal.

L'action du JavaScript est éphémère et ne dure que le temps consacré à la visite de la page par l'utilisateur.

---

4. Outils pour le JavaScript

Le code source des pages Web présente cet avantage qu'il ne nécessite pas de logiciels coûteux. JavaScript ne déroge pas à la règle.

Pour l'apprentissage du JavaScript, il vous faut :

- un simple éditeur de texte,
- un navigateur compatible JavaScript,
- et des connaissances du langage Html.

Le JavaScript, comme la plupart des langages de programmation s'écrit en texte brut. Ainsi un simple éditeur de texte comme le Bloc-notes de Windows (notepad) fait parfaitement l'affaire. L'utilisation de la fenêtre de code d'un éditeur Html est également possible.

Le JavaScript étant interprété par le navigateur, n'importe quel navigateur compatible JavaScript peut être adopté. Ce qui est le cas de la plupart des navigateurs du marché. Internet Explorer 6 et 7 ou Firefox, retenus dans le cadre de cet ouvrage, sont bien entendu parfaitement adaptés au JavaScript. Veillez cependant à ce que la fonction JavaScript ne soit pas désactivée dans les options de votre navigateur.

Puisque le code JavaScript vient se greffer dans le code source de la page, une connaissance approfondie des balises Html ou Xhtml est recommandée, sinon indispensable.

5. JavaScript et le Xhtml

Le JavaScript est interprété par le navigateur. Il faut donc l'informer que le code qu'il risque de rencontrer est du JavaScript.

a. Balises meta

Au niveau des balises meta, on peut déterminer le ou les langage(s) complémentaire(s) utilisé(s) dans le document Xhtml.

On ajoutera aux autres balises meta, la balise :

```
<meta http-equiv="Content-Script-Type" content="text/javascript" />
```

Cette balise signale que la page comporte (ou peut comporter) un script et que ce script, en mode texte, est du JavaScript.

b. Code JavaScript

Le code JavaScript sera signalé au navigateur en entourant celui-ci des balises `<script> ... </script>`

```
<script type="text/javascript">
//<![CDATA[
Code JavaScript
//]]>
</script>
```

La balise `<script type="text/javascript">` détermine qu'il s'agit d'un script JavaScript. En effet, il existe d'autres langages de script qui peuvent s'implémenter dans une page Html. On pense au VBscript de Microsoft qui est un langage issu du Visual Basic mais qui n'a pas eu le même succès que le JavaScript.

Dans la pratique, on rencontre fréquemment la déclaration abrégée `<script>`. Celle-ci est valable car la plupart des navigateurs ont repris le JavaScript comme langage de script par défaut.

Parfois, la version du JavaScript est ajoutée à cette balise pour forcer le navigateur à respecter les spécifications de la version ainsi mentionnée. Ce qui donne par exemple `<script type="text/javascript" language="javascript1.2">`. Cette précision n'est plus très utilisée car les navigateurs récents se "débrouillent" pour gérer le JavaScript indépendamment de sa version.

En Html, on a pris l'habitude d'entourer le code JavaScript des balises `<!-- ... //-->` pour cacher le code à l'intention des navigateurs qui ne reconnaissent pas le JavaScript. Remarquons que les balises `<!-- ... -->` sont des commentaires Html et les barres obliques // du commentaire JavaScript.

En Xhtml, il faut entourer le code JavaScript par `//<![CDATA[ ... //]]>`. Cette notation, issue du XML, permet d'informer le navigateur de ne pas analyser et interpréter ce qu'il y a entre le `<![CDATA[` initial et le `]]>` final. L'analyseur syntaxique va seulement récupérer le contenu de cette section, sans chercher à repérer des balises, des entités ou toute autre spécificité syntaxique XML. En effet, sans `//<![CDATA[`, les signes < inférieur à et > supérieur à du JavaScript seraient reconnus comme les signes < de début de balise et > de fin de balise du Xhtml.

6. Un premier script

Encodons le code suivant dans un éditeur de texte :

```
<!DOCTYP<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Premier script</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type='text/JavaScript'>
//<![CDATA[
document.write("Mon premier JavaScript");
//]]>
</script>
</body>
</html>
```

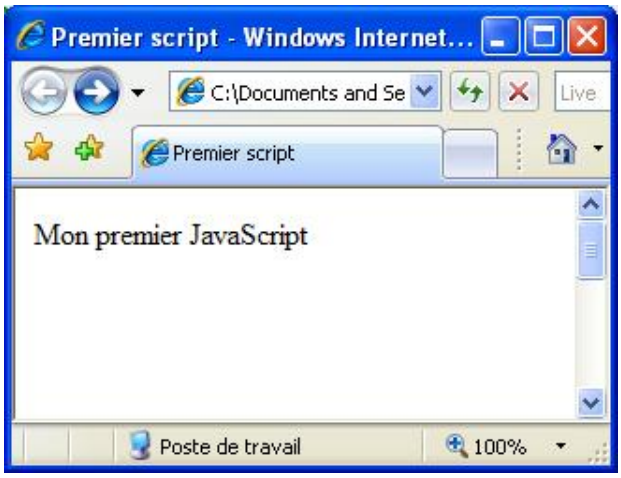
Commentaires :

- le doctype n'a aucune influence sur l'exécution du JavaScript. Ce qui n'est pas le cas des feuilles de style.
- l'instruction `document.write("Mon premier JavaScript");` demande d'écrire (write) le texte repris entre les parenthèses dans le document.

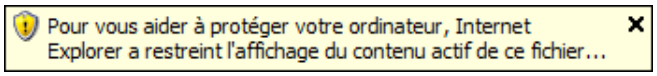
- remarquez que l'instruction se termine par un point-virgule.

On enregistre ce fichier au format htm.

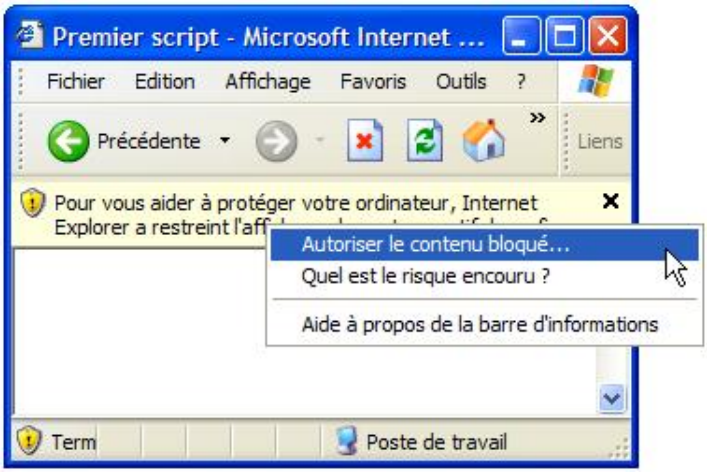
Ensuite, on affiche la page locale dans le navigateur.



La version Internet Explorer 6.0 pour Windows XP SP2 risque de vous apporter quelques désagréments lors de la conception de vos scripts en local.Soucieux (mais à ce stade, un peu trop !) de la sécurité des pages affichées, Internet Explorer affiche le message d'information suivant.



Pour visionner votre page comportant un script, il faut autoriser le contenu ainsi bloqué.



Ce message d'information n'apparaîtra plus (sauf exceptions) une fois que la page sera en ligne.

Il est à noter que cet avertissement de sécurité a été supprimé dans Internet Explorer 7.0.

7. JavaScript interne

On peut se poser la question ; "Où inclure les balises `<script> ... </script>` dans le document Xhtml ?".

Il n'y a hélas, pas de règle précise qui stipule à quel endroit d'un fichier Html ou Xhtml, le code JavaScript doit être défini. La seule réponse est "Là où le navigateur en aura besoin" !

Pour rappel, la page Html est chargée par le navigateur de façon séquentielle (de haut en bas). Il lit d'abord l'en-tête (le contenu des balises <head>) et ensuite le corps (<body>), ligne par ligne. Tous les éléments du "body" sont interprétés dans l'ordre d'écriture des balises.

Si on fait appel à du code JavaScript, alors que celui-ci n'a pas encore été défini dans la page et donc interprété par le navigateur, ce dernier ne peut, en toute logique, pas le prendre en charge (erreur sur la page).

On peut ainsi retrouver le code JavaScript à plusieurs endroits de la page :

- il n'est pas rare que du code JavaScript soit inclus, totalement ou en partie, dans les balises <head>. En effet, ce code sera alors lu en premier par le navigateur et le concepteur peut en toute sécurité faire appel à ces fragments de code, n'importe où à l'intérieur de la balise <body> car il sera déjà disponible.
- on retrouve parfois du code JavaScript immédiatement après la balise <body> lorsque celui-ci est plus spécifique au corps du document.
- si le JavaScript traite les données d'un formulaire, le code peut apparaître après la balise <form>.
- des instructions comme `document.write("...");` peuvent être incluses dans le texte de la page à l'endroit souhaité.
- etc.

Le code JavaScript doit être encodé avant ou au plus tard au moment où il devra être exécuté.

En outre, depuis que le JavaScript est reconnu comme le langage de script par défaut de la plupart des navigateurs, du code JavaScript peut-être inclus directement dans la balise Html ou Xhtml, fonctionnant alors comme un attribut de celle-ci.

Cette façon de procéder n'est pas adoptée pour un code programme compliqué mais seulement pour un appel à des événements, méthodes, fonctions déterminés.

Quelques exemples :

```
<a onclick="javascript:history.back();">Retour page précédente</a>
```

Avec l'objet history de JavaScript, vous avez accès aux pages Web qui ont été visitées par l'utilisateur et qui sont sauvegardées dans l'historique du navigateur. Avec la méthode back(), on demande, au clic sur le lien, de remonter d'une position dans l'historique, revenant ainsi à la page précédente.

```
<h1 onmouseover="history.back()">Retour page précédente</h1>
```

Ici on demande de revenir à la page précédente par le simple survol de la souris (onmouseover).

```
<form>
<input type="button" value="retour" onclick="history.back()">
</form>
```

Quand l'utilisateur clique sur le bouton de formulaire, on revient à la page précédente.

8. JavaScript externe

Il est également possible de noter le code JavaScript dans un ou des fichiers séparés de la page Xhtml. Ce qui présente l'avantage de clarifier le code du document Xhtml et de respecter la règle de séparation du contenu et de la présentation. Il est ainsi possible d'appeler le même code JavaScript à partir de fichiers Xhtml séparés sans avoir à le réécrire.

Ce fichier externe est un fichier de texte brut encodé, par exemple, avec le Bloc- notes de Windows. Ce type de fichier comporte l'extension .js. Il contient uniquement du code JavaScript, et donc sans les balises <script>.

Ce fichier est appelé dans la page Xhtml par :

```
<script src="fichier_externe.js"></script>
```

Cette balise d'appel se place généralement entre les balises <head> ... </head>.

Reprenons notre exemple du point Un premier Script de ce chapitre.

a. Le fichier externe

Ce fichier externe, que nous appellerons write.js, comporte la seule ligne de code :

```
document.write("Mon premier JavaScript externe");
```

b. La page Xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Script externe</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script src="write.js"></script>
</head>
<body>
</body>
</html>
```

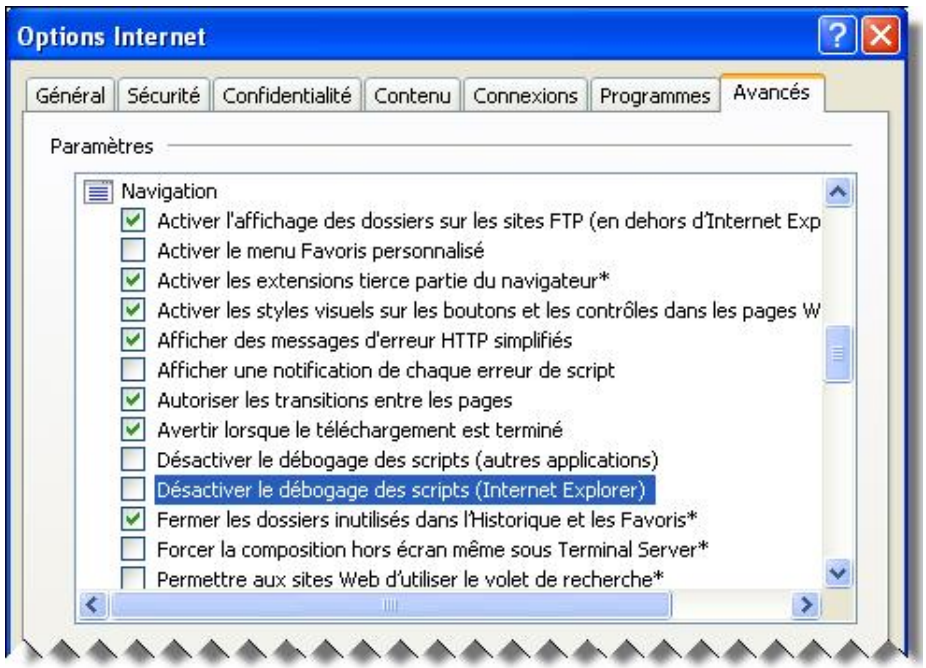
Le résultat est identique à la capture d'écran proposée au point Un premier Script.

9. Quelques conseils pour le débogage

Vous serez inévitablement déroutés lors de vos premières applications en JavaScript. En effet, les navigateurs permettent beaucoup de "souplesse" dans le code Html et les feuilles de style CSS. Avec le JavaScript, on passe à la rigueur d'un langage de programmation. La moindre erreur sera sanctionnée ! Et une erreur est si vite arrivée... Un point à la place d'une virgule, une faute de frappe, une majuscule absente ou indue, une parenthèse manquante, un guillemet au lieu d'une apostrophe, une accolade oubliée, voilà un florilège de petites erreurs qui empêchent l'exécution d'un script.

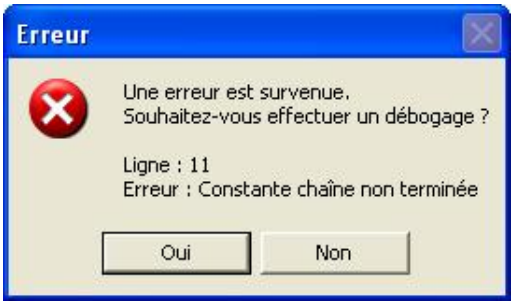
Pour faciliter l'apprentissage, l'auteur conseille d'activer le débogueur de script par défaut de votre navigateur.

- Dans le cadre d'Internet Explorer, **Outils - Options Internet** - Onglet **Avancés**.

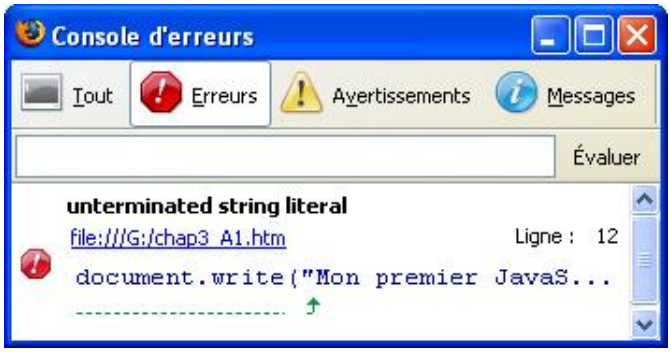


Il est peut-être nécessaire d'installer le "Script Debugger" de Microsoft. Ce programme est disponible en téléchargement pour différentes versions du système d'exploitation de Microsoft à l'adresse : http://www.microsoft.com/downloads/results.aspx?productID=&freetext=script+debugger&DisplayLang=en ou plus simplement, par une recherche sur Google avec comme mots-clés "Script Debugger download".

Un message d'erreur du type affiché ci-dessous vous fait gagner un temps précieux et vous aide à corriger votre script.



- Si vous utilisez le navigateur Firefox, vous avez à votre disposition la console JavaScript, **Outils - Développement Web - Console JavaScript** ou **Outils - Console d'erreurs** (selon les versions).



Notions fondamentales

Avant de passer à l'étude proprement dite de tout langage de programmation, aussi modeste soit-il comme le JavaScript, il importe de parcourir quelques règles d'écriture et de syntaxe.

1. La casse

Le JavaScript est sensible à la casse (case sensitive). Il fait donc la différence entre les majuscules et les minuscules.

Ainsi, la variable `var = numéro` n'est pas égale à la variable `var = Numero` ni à `var = NUMERO`.

L'alphabet ASCII composé de 128 caractères est utilisé. Ainsi le code doit être écrit sans caractères accentués.

Les espaces, tabulations, retours chariot, lignes vierges sont (sauf exceptions) ignorés dans le code.

2. Les commentaires

Il est toujours utile d'annoter le code avec des commentaires en vue de faciliter sa lisibilité, surtout à la relecture.

En JavaScript, ces commentaires peuvent se présenter sous deux formes différentes :

- par une double barre oblique (//) devant le commentaire. Cette notation est bien adaptée pour les commentaires qui se notent sur une seule ligne.

```
// Ceci est un commentaire
```

- par les signes /* ... */. Ce qui permet d'étendre le commentaire sur plusieurs lignes.

```
/* Pour les bavards, JavaScript a prévu
des commentaires sur plusieurs lignes */
```

3. Le point-virgule

Chaque commande ou instruction JavaScript se termine par un point-virgule.

Exemple : `document.write ("ligne : " + compt + "<br>");`

Bien que cela ne soit plus obligatoire en toutes circonstances, il reste prudent de maintenir cette règle de façon systématique lorsqu'on est débutant en JavaScript.

4. Les constantes

Les constantes sont fixées par la valeur indiquée dans le code.

Les valeurs reconnues par le JavaScript sont :

Les nombres entiers ou à virgule flottante

Notons que pour ces derniers, le point remplace la virgule habituellement utilisée en bureautique.

2005

16.6666

Les chaînes de caractères

Elles sont constituées d'une suite de caractères quelconques, encadrée par des guillemets ou des apostrophes. Une chaîne commencée avec des guillemets doit se terminer avec des guillemets, même chose pour l'apostrophe.

"CSS + DHTML"

'3.1416'

Si la chaîne de caractères contient un guillemet, une apostrophe ou une barre oblique à gauche (backslach), ces derniers risquent d'être interprétés comme du code JavaScript. Les caractères cités s'encodent ainsi précédés d'une barre oblique à gauche.

Soit :

\' pour l'apostrophe.

\" pour le guillemet.

\\ pour la barre oblique à gauche.

```
document.write ("Je lui ai dit \; \"Va à l\'école \"").
```

Une chaîne de caractères peut être vide. C'est l'équivalent de la valeur zéro pour une constante numérique. Dans ce cas, elle s'écrit au moyen de deux guillemets ou de deux apostrophes rigoureusement consécutifs (sans espace entre eux).

les valeurs logiques ou booléennes, soit *true* pour vrai et *false* pour faux.

null lorsqu'il n'y a pas de valeur attachée à la variable.

undefined. C'est la valeur d'une variable qui n'a pas été initialisée.

5. Les variables

Contrairement à la plupart des autres langages de programmation, JavaScript n'est que faiblement typé. En effet, il n'est pas nécessaire d'en déclarer le type (comme par exemple avec int, float, double, char de PHP) et une variable peut à tout moment changer de type.

Les variables se déclarent de deux façons :

a) explicitement par le mot clé var.

```
var indice
```

b) implicitement par son apparition à gauche du signe égal.

```
b = 256
```

Le nom de la variable doit respecter la syntaxe suivante :

- la variable doit commencer par une lettre ou un souligné "_".
- la variable peut comporter un nombre indéterminé de lettres, de chiffres ainsi que des caractères _ et $.
- les espaces ne sont pas autorisés.
- le nom de variable ne peut utiliser des mots dits "réservés". En effet, ces mots sont utilisés dans le code JavaScript même. On ne peut nommer une variable, par exemple, var, true, false, else, etc. La liste complète de ces mots réservés est fournie en annexe.
- pour rappel, le JavaScript est sensible aux majuscules et minuscules.

Exemples de déclaration de variables valides :

- `var pi;`
- `var code_postal;`
- `var formulaire1;`

- `var result$;`

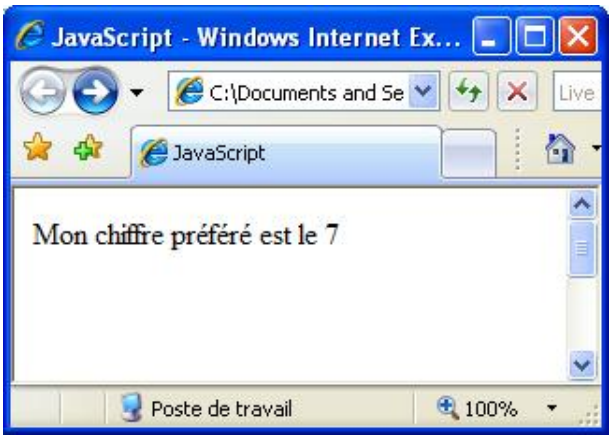
On peut initialiser une variable en même temps que sa déclaration au moyen du signe égal (=) suivi d'une valeur numérique, d'une chaîne de caractères ou d'une valeur booléenne. Ce qui évite que la valeur *undefined* soit retournée en cours d'exécution du script.

Exemples :

- `var pi = 3.1415926535;`
- `var code_postal = 59000;`
- `var formulaire1 = "Ville";`
- `var result$ = true;`

Exemple :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type='text/JavaScript'>
//<![CDATA[
var texte = "Mon chiffre préféré est le ";
var variable = 7;
document.write(texte + variable);
//]]>
</script>
</body>
</html>
```



6. Les opérateurs

JavaScript comporte de nombreux opérateurs selon le type de valeurs.

a. Les opérateurs arithmétiques

Ce sont les opérateurs classiques de toutes les opérations de calcul.

Opérateur	Nom	Signification	Exemple
++	plus	addition	x + 100
-	moins	soustraction	x - 1
*	multiplié	multiplication	x * 2
/	divisé	division	x / 10
%	modulo	reste de la division par	356 % 5
=	égal	affectation de valeur	i = 4

Commentaires :

- lorsque l'opérateur + porte sur deux opérateurs dont l'un au moins est une chaîne de caractères, il joue le rôle d'opérateur de concaténation (voir plus loin).
- l'opérateur = affecte simplement une valeur. On le confond souvent avec l'opérateur d'égalité ==.

b. Les opérateurs de comparaison

Ces opérateurs sont souvent utilisés lors de tests conditionnels.

Opérateur	Nom	Signification
<	inférieur	x < 10
<=	inférieur ou égal	x <= 10
==	égal	x == 10
>=	supérieur ou égal	x >= 10
>	supérieur	x > 10
!=	différent	x != 10

Pour rappel, le = est un opérateur d'attribution de valeur tandis que le == est un opérateur de comparaison. Cette confusion est une source fréquente d'erreur de programmation.

Le résultat renvoyé par une opération de comparaison est une valeur booléenne *true* (vrai) ou false (faux).

c. Les opérateurs booléens (ou logiques)

Ces opérateurs sont également utilisés dans les tests conditionnels mais logiques cette fois. Ils portent sur des opérandes booléens et le résultat renvoyé est également une valeur booléenne (true ou false).

Opérateur	Nom	Signification	Exemple
&&	et	les deux conditions sont vérifiées	condition1 && condition2
||	ou	une des deux conditions est vérifiée	condition1 || condition2
!	!	la condition n'est pas vérifiée	! condition1

d. Les opérateurs associatifs

Ces opérateurs associent deux opérateurs d'affectation. C'est une forme abrégée pour noter les opérateurs de calcul. Ceci est plutôt réservé à des programmeurs plus confirmés.

Opérateur	Nom	Exemple	Signification
+=	plus égal	x += y	x = x + y
-=	moins égal	x -= y	x = x - y
*=	multiplié égal	x *= y	x = x * y
/=	divisé égal	x /= y	x = x / y

e. Les opérateurs d'incrémentation

Un classique des langages de programmation mais souvent déroutant pour les novices. Cet opérateur d'incrémentation est très utilisé pour faire varier d'une unité les compteurs implémentés dans le code.

Ainsi pour une valeur de départ x = 0, au premier passage x++ vaut 1 (x=x + 1 ou 1=0 + 1). Au second passage, x++ vaut 2 (x=x +1 ou 2=1 + 1). Au troisième passage, x++ vaut 3 (x=x +1 ou 3=2 + 1). Et ainsi de suite.

Opérateur	Nom	Exemple	Signification
x++	incrémentation	x = x++	x++ est le même que x=x+1
x--	décrémentation	x = x--	x-- est le même que x=x-1

f. Les opérateurs de concaténation

Utilisé avec des chaînes de caractères, cet opérateur ajoute une chaîne de caractères à la suite d'une autre chaîne de caractères.

Opérateur	Nom	Exemple	Signification
+	concaténation	"chaine1" + "chaine2"	"chaine1 chaine2"

Lorsque vous concaténez des chaînes de caractères, il ne faut pas oublier les espaces au début ou à la fin de chacune d'elles, sous peine de coller les chaînes.

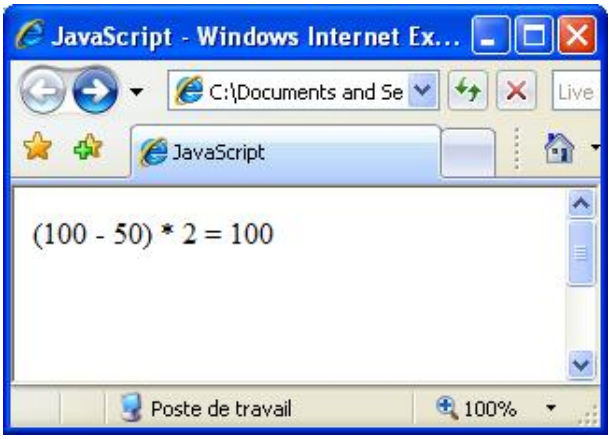
g. Autres opérateurs

Il existe encore d'autres opérateurs mais ils dépassent largement le cadre de cet ouvrage.

Exemple

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type='text/JavaScript'>
//<![CDATA[
var variable = (100 - 50) * 2;
document.write("(100 - 50) * 2 = " + variable);
//]]>
</script>
```

```
</body>
</html>
```



h. Priorité des opérateurs

Comme en algèbre, lorsqu'une expression comporte plusieurs opérateurs, celle-ci est évaluée selon la priorité respective des opérateurs.

Voici, pour les opérateurs repris dans notre étude, la liste des priorités, de la plus basse à la plus haute.

Opérateur	Symbole
virgule	,
association et affectation	= += -= *= /= < <= > >=
ou logique	||
et logique	&&
comparaison	= != < <= >= >
addition/soustraction	+ -
multiplication/division	* /
différent et incrémentation	! ++ --
parenthèses	( )

En cas de priorité égale d'opérateurs consécutifs, l'opération est effectuée de gauche à droite.

On peut toujours modifier une priorité d'opérateurs par l'emploi de parenthèses car celles-ci ont le plus haut niveau de priorité.

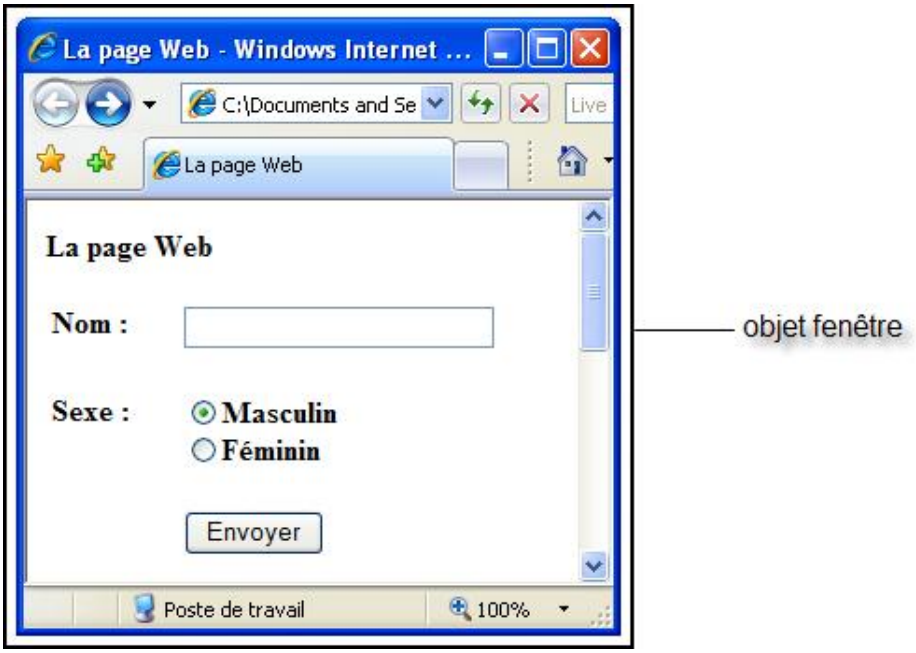
7. Un peu de théorie objet

JavaScript est un langage orienté objet. Il va ainsi diviser la page affichée à l'écran en objets mais surtout, va vous permettre d'accéder à ces objets et de les manipuler.

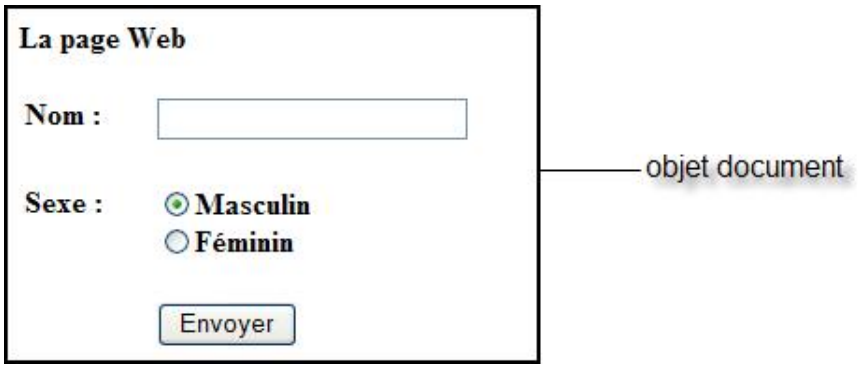
Illustrons tout d'abord différents objets qui peuvent être contenus dans une page Web.

Soit la page suivante.

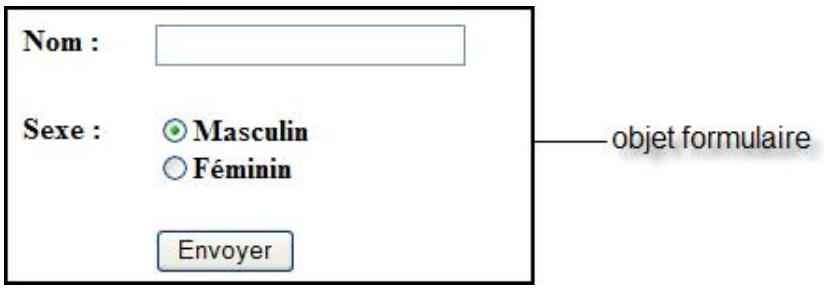
Cette page s'affiche dans une fenêtre du navigateur. C'est l'**objet fenêtre** (*window*).



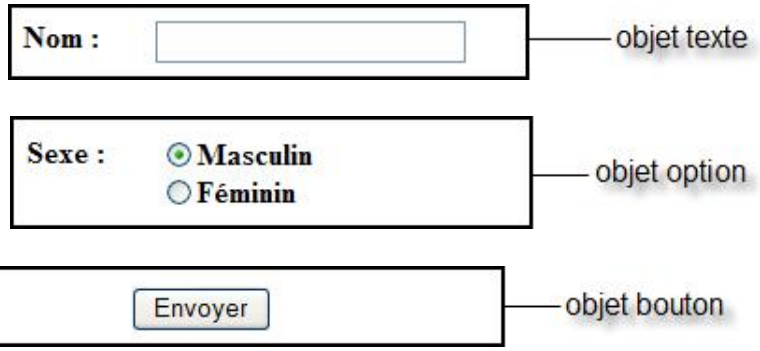
Dans cette fenêtre, il y a un document Html. C'est l'**objet document**. Autrement dit (et c'est là que l'on voit apparaître la notion de hiérarchie des objets JavaScript), l'objet fenêtre contient lui-même l'objet document (document).



Dans ce document, on trouve un formulaire au sens Html du terme. C'est l'**objet formulaire** (form). Autrement dit, l'objet fenêtre contient un objet document qui lui-même contient un objet formulaire.



Dans ce formulaire, on trouve trois éléments : une zone de texte, des boutons d'option et un bouton d'envoi. Ce sont respectivement l'**objet texte**, l'**objet option** (radio) et l'**objet bouton**. Soit l'objet fenêtre contient l'objet document qui contient l'objet formulaire qui contient à son tour l'objet texte, l'objet option et l'objet bouton.



La hiérarchie des objets de cet exemple est donc :

fenêtre -> document -> formulaire -> texte, option, bouton

Pour accéder à un objet, il faut donner le chemin complet de l'objet en allant du contenant le plus extérieur à l'objet référencé.

Soit par exemple pour le bouton option "Masculin"

```
(window).document.form.radio[0].
```

Nous avons mis l'objet window entre parenthèses car window occupe toujours la première place dans la hiérarchie des objets. Il est repris par défaut en JavaScript et devient donc facultatif.

Cette façon de procéder est la première manière mise en place par le JavaScript pour accéder à certains objets. Depuis 1998 et l'apparition du DOM (*Document Object Model*) d'autres procédés ont vu le jour pour l'accès aux objets. On peut citer par exemple, les méthodes getElementByID, getElementsByName ou getElementsByTagName (voir chapitre Le DOM (Document Object Model)- Accéder aux noeuds).

La notation utilisée ci-avant, appelée a posteriori DOM niveau 0, accompagne nos premiers pas en JavaScript, on lui préfère cependant en cours d'étude, des méthodes plus récentes pour accéder aux objets.

Fonctions et méthodes

1. Déclaration d'une fonction

Une fonction est un groupe de lignes de code de programmation, écrites par le concepteur et destinées à exécuter une tâche bien spécifique. On peut, si besoin est, l'utiliser à plusieurs reprises dans la page. En outre, l'usage des fonctions améliore grandement la lisibilité de votre script.

Pour déclarer ou définir une fonction, on utilise le mot (réservé) function.

La syntaxe d'une déclaration de fonction est la suivante :

```
function nom_de_la_fonction(arguments) {
... code des instructions ...
}
```

Remarques :

- le nom de la fonction suit les mêmes règles que celles qui régissent le nom des variables (voir dans ce chapitre - Notions fondamentales - Les variables). Pour rappel, JavaScript est sensible à la case. Ainsi function_a() ne sera pas égal à Function_a().
- tous les noms de fonctions dans un script doivent être uniques.
- la mention des arguments est facultative mais dans ce cas, les parenthèses doivent rester. C'est d'ailleurs grâce à ces parenthèses que l'interpréteur JavaScript distingue les variables des fonctions. Nous reviendrons plus en détail sur les arguments et autres paramètres.
- lorsqu'une accolade est ouverte, elle doit impérativement, sous peine de message d'erreur, être refermée. Prenez la bonne habitude de fermer directement vos accolades et d'écrire le code entre elles. De nombreux scripts ne fonctionnent pas pour des erreurs d'accolades fermantes.

Le nombre d'accolades ouvertes (voir fonctions, tests conditionnels, etc) doit toujours être égal au nombre d'accolades fermées.

- le fait de définir une fonction n'entraîne pas l'exécution des commandes qui la composent. Ce n'est que lors de l'appel de la fonction que le code de programme est exécuté.
- la déclaration de fonction se place souvent dans l'en-tête du document Html ou Xhtml soit entre les balises `<head> ... </head>`. Elle est ainsi toujours disponible et peut être appelée à tout moment dans le document.

Exemple :

```
<head>
<script type="text/javascript">
//<![CDATA[
function message(){
document.write("Bienvenue");
}
//]]>
</script>
</head>
```

2. Appel d'une fonction

Le script de la fonction ne s'exécute que lorsque celle-ci est appelée.

L'appel d'une fonction se fait par le nom de la fonction (avec les parenthèses). On y adjoint les paramètres éventuels.

Soit par exemple :

```
<script type="text/javascript">
//<![CDATA[
message();
//]]>
</script>
```

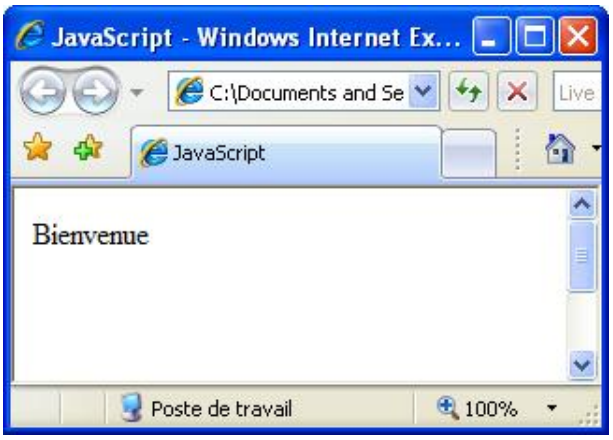
Autre possibilité, l'appel de la fonction se réalise souvent au moyen de gestionnaires d'événements comme le chargement de la page, le clic d'un bouton. Nous retenons ici l'événement *onload* qui appelle la fonction message() au chargement de la page.

```
<body onload="message();">
```

Il faut veiller à ce que la fonction soit déjà définie et connue de l'interpréteur avant son exécution.

La page complète serait :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function message() {
document.write("Bienvenue");
}
//]]>
</script>
</head>
<body onload="message();">
</body>
</html>
```



3. Passage de paramètre(s)

Il est possible de passer des paramètres à une fonction, c'est-à-dire de lui fournir une valeur ou le nom d'une variable afin que la fonction puisse effectuer les opérations programmées à l'aide de ces paramètres.

Lorsqu'on passe plusieurs paramètres à une fonction, il faut les séparer par des virgules, aussi bien dans la déclaration que dans l'appel.

Soit une fonction qui calcule le cube d'un nombre :

```
function calcul(nombre)
var cube = nombre*nombre*nombre;
}
```

Cette fonction doit avoir une valeur de départ pour effectuer son opération. Cette valeur lui est fournie en argument lors de l'appel de la fonction.

Lors de l'appel : `calcul(5);`

Ou sous forme de variable :

```
var nombre = 5
calcul(nombre);
```

Appliqué à notre exemple précédent, le script pourrait devenir :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function message(texte) {
document.write(texte);
}
//]]>
</script>
</head>
<body onload="message('Bienvenue');">
</body>
</html>
```

Nous avons entouré la chaîne de caractères 'Bienvenue' par des apostrophes car les guillemets étaient déjà utilisés pour l'attribut `onload="..."`.

4. Variable locale ou globale

Il existe deux types de variables, les variables locales et les variables globales. Une variable globale est accessible partout dans le script. Une variable locale n'est accessible que dans la fonction qui l'a créée.

C'est ce qu'on appelle la portée des variables. Cette distinction est une source fréquente d'erreurs, souvent difficiles à déceler, dans les scripts.

a. Variable locale

Une variable déclarée dans une fonction (donc à l'intérieur des accolades de la fonction) par le mot clé var a une portée limitée à cette seule fonction. On ne peut pas exploiter cette variable ailleurs dans le script. On l'appelle donc variable locale.

```
function calcul(nombre)
var cube = nombre*nombre*nombre;
}
```

Ainsi, la variable *cube* est dans cet exemple une variable locale. Si vous y faites référence ailleurs dans le script, cette variable est inconnue pour l'interpréteur JavaScript (message d'erreur).

Si la variable est déclarée contextuellement (sans utiliser le mot var), sa portée est globale une fois que la fonction a été exécutée.

```
function calcul(nombre) {
cube = nombre*nombre*nombre
}
```

La variable *cube* déclarée contextuellement est ici une variable globale après l'exécution de la fonction `cube()`.

b. Variable globale

Les variables déclarées tout au début du script, en dehors et avant toutes fonctions, sont toujours globales, qu'elles soient déclarées avec var ou de façon contextuelle.

```
<script type="text/javascript">
//<![CDATA[
var cube=1
function calcul(nombre) {
var cube = nombre*nombre*nombre
}
//]]>
</script>
```

La variable *cube* est bien globale.

Pour la facilité de gestion des variables, on ne peut que conseiller de les déclarer en début de script (comme dans la plupart des langages de programmation). Cette habitude vous met à l'abri de complications certaines.

5. L'instruction return

Selon l'écriture du code, le résultat d'une fonction peut être retourné par l'instruction return.

Soit la fonction multiple() qui calcule le produit de deux nombres (a et b).

```
function multiple(a,b) {
x = a*b;
return x;
}
```

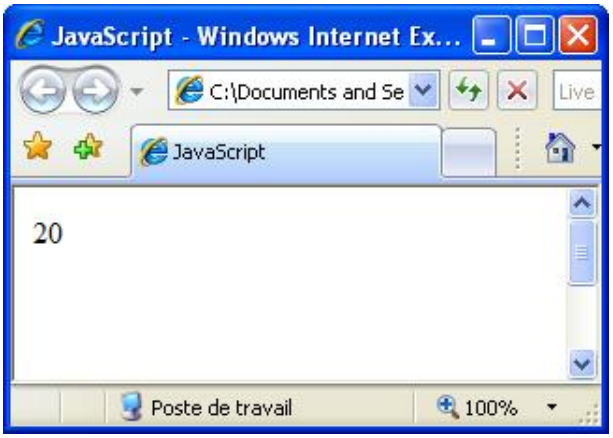
À l'appel de la fonction, on lui passe les paramètres correspondant aux arguments a et b, soit `multiple(4,5)`.

```
resultat=multiple(4,5);
```

Comme la valeur retournée par la fonction multiple() est 20, celle-ci est stockée dans la variable resultat.

Le script complet devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function multiple(a,b) {
x = a*b;
return x;
}
//]]>
</script>
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
resultat=multiple(4,5);
document.write(resultat);
//]]>
</script>
</body>
</html>
```



6. Quelques méthodes JavaScript

Les méthodes sont des fonctions prédéfinies dans le langage JavaScript et dédiées à un objet particulier.

Nous avons jusqu'ici utilisé la méthode JavaScript write(), spécifique à l'objet document.

Passons en revue quelques autres méthodes (de l'objet window). Cela permet de varier nos exemples dans la suite de notre exposé.

a. Alert()

La méthode `alert()` de l'objet fenêtre affiche une boîte de dialogue. Celle-ci comporte un message qui reproduit la valeur (variable et/ou chaîne de caractères) de l'argument qui lui a été fourni. Cette boîte de dialogue bloque le programme en cours tant que l'utilisateur n'aura pas cliqué sur **OK** pour fermer celle-ci.

Régal des programmeurs débutants en JavaScript, cette méthode est certes spectaculaire mais d'un rôle marginal dans un site Web. Par contre, `alert()` est très utile pour vous aider à débugger les scripts et y retrouver d'éventuelles erreurs de programmation.

Sa syntaxe est :

```
alert(variable);
alert("chaîne de caractères");
alert(variable + "chaîne de caractères");
```

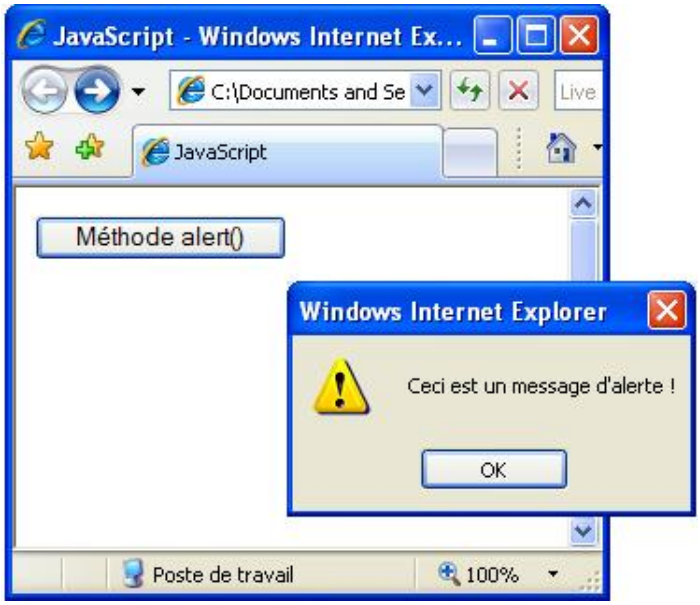
Si vous souhaitez écrire sur plusieurs lignes, il faut utiliser le signe \n.

Exemple

Une boîte d'alerte va se déclencher lorsque le bouton est cliqué.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function alerte() {
alert ("Ceci est un message d'alerte !");
}
//]]>
</script>
</head>
<body>
<form>
<input type=button value="Méthode alert()" onclick="alerte()">
```

```
</form>
</body>
</html>
```



b. confirm()

Cette méthode de l'objet window affiche une boîte de dialogue avec deux boutons ; **OK** et **Annuler**. En cliquant sur **OK**, la méthode renvoie la valeur *true* et bien entendu *false* si on a cliqué sur **Annuler**. Ce qui peut permettre, par exemple, de définir des options dans un programme.

Exemple

Le script lance une boîte de confirmation. La valeur renvoyée est mise dans la variable a. Si la valeur de a est true (`if (a == true)`) une boîte d'alerte s'affiche, si ce n'est pas le cas, une autre boîte de dialogue s'affiche.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
var a = confirm("Cliquez OK ou Annuler");
if(a == true){
alert ("Vous avez cliqué OK");
}
else {
alert ("Vous avez cliqué Annuler");
}
//]]>
</script>
</body>
</html>
```

c. prompt()

Dans le même style que les précédentes méthodes de l'objet window, JavaScript vous propose une autre boîte de dialogue, appelée boîte d'invite. Elle est composée d'un champ comportant une entrée à compléter par l'utilisateur. Cette entrée peut aussi posséder une valeur par défaut.

La syntaxe est :

```
prompt("texte de la boîte d'invite","valeur par défaut");
```

En cliquant sur **OK**, la méthode renvoie la valeur encodée par l'utilisateur ou la valeur proposée par défaut. Si l'utilisateur clique sur **Annuler**, la valeur *null* est alors renvoyée.

La méthode `prompt()` est parfois utilisée pour saisir des données fournies par l'utilisateur.

Exemple

On va demander le prénom du visiteur par une boîte d'invite et l'afficher sur la page Web.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
var prenom = prompt("Encodez votre prénom", "Votre prénom ici");
document.write("<h2>Bonjour " + prenom + "</h2>");
//]]>
</script>
</body>
</html>
```

La boîte d'invite :

Le document Xhtml :

d. Une minuterie

JavaScript met à votre disposition une minuterie (ou plus précisément un compte à rebours) qui permet de déclencher une fonction après un laps de temps déterminé.

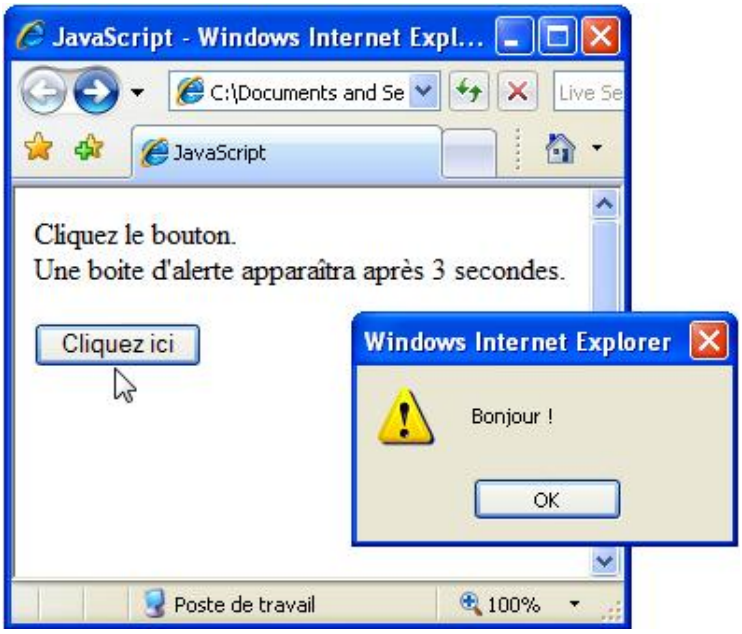
La syntaxe de mise en route du temporisateur est :

```
nom_du_compteur = setTimeout<$I[]setTimeout>("fonction_appelée()", temps en mil
liseconde)
```

Ainsi, `setTimeout("demarrer()",5000)` va lancer la fonction `demarer()` après 5 secondes.

Exemple

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function display(){
alert("Bonjour !")
}
//]]>
</script>
</head>
<body>
Cliquez le bouton.<br />
Une boîte d'alerte apparaîtra après 3 secondes.
<form>
<input type="button" onclick="setTimeout('display()',3000)"
value="Cliquez ici">
</form>
</body>
</html>
```



Pour arrêter le temporisateur avant l'expiration du délai fixé, il suffit d'utiliser la méthode`clearTimeout (nom_du_compteur)`.

Conditions et boucles

1. Les conditions if ... else

Les langages de programmation n'échappent pas aux tests conditionnels. C'est ce qui en fait d'ailleurs une de leur richesse.

L'expression *if* (si) permet d'exécuter ou non, une série d'instructions en fonction du résultat d'un test.

```
if (condition vraie) {
une ou plusieurs instruction(s);
}
```

Si la condition est vérifiée (true), les instructions s'exécutent. Si elle ne l'est pas (false), les instructions ne s'exécutent pas et le programme passe à la commande suivante.

Remarquons que les instructions sont comprises entre une accolade ouvrante et fermante.

Voici une forme plus évoluée :

```
if (condition vraie) {
instructions 1;
}
else {
instructions 2;
}
```

Si la condition est vérifiée (true), le bloc d'instructions 1 s'exécute. Sinon (else), soit lorsque la condition renvoie la valeur false, le bloc d'instructions 2 s'exécute.

Exemple

Le visiteur entre un nombre compris entre 0 et 99. Le script annonce si le nombre encodé est inférieur ou égal à 50 ou supérieur à 50.

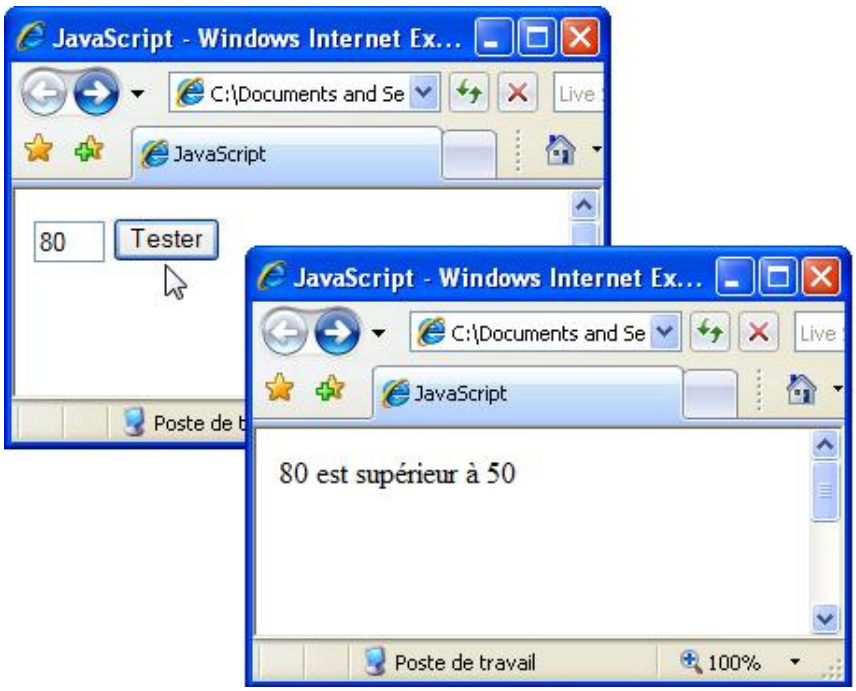
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function test(){
var a = document.formulaire.input.value;
if (a <= 50) {
document.write(a +"  inférieur ou égal à 50");
}
else {
document.write(a + " est supérieur à 50");
}
}
//]]>
</script>
</head>
<body>
<form action="" name="formulaire">
<input type="text" name="input" size="2" maxlength="2" value="" />
<input type="button" value="Tester" onclick="test()" />
</form>
</body>
</html>
```

- le formulaire comporte une ligne de texte limitée à 2 positions (voir `maxlength="2"`) pour encoder un chiffre compris entre 0 et 99.

- en cliquant le bouton **Tester**, le visiteur appelle la fonction test() définie entre <head> ... </head>.
- dans la variable `test()`, la variable a, chiffre encodé dans la zone de texte, est récupérée. Pour ce faire, le formulaire est identifié et la valeur de zone de texte, nommée input, est extraite (cf. ce chapitre - Notions fondamentales - Un peu de théorie objet).

Ce chemin s'écrit en JavaScript, `document.formulaire.input.value`.

- la condition "a inférieur ou égal à 50" (a <= 50) est testée ;
- si cette condition est réalisée, l'information "chiffre encodé inférieur ou égal à 50" est écrite dans le document.
- si cette condition n'est pas réalisée (*else*), l'information "chiffre encodé est supérieur à 50" est écrite dans le document.
- remarquez les deux accolades de fin. La première termine le test if ... else. La seconde termine la fonction test ().



Il est également possible de concevoir de multiples tests conditionnels.

```
if (condition 1) {
instruction(s) à exécuter si la condition 1 est vraie
}
else if (condition 2) {
code à exécuter si la condition 2 est vraie
}
else
{
code à exécuter si tous les tests sont faux
}
```

Pour ceux qui aiment les notations concises, on peut aussi écrire :

```
(expression) ? instruction a : instruction b
```

Si l'expression entre parenthèses est vraie, l'instruction a est exécutée. Si l'expression entre parenthèses est fausse, c'est l'instruction b qui est exécutée. Remarquons le double point entre les deux instructions a et b.

Le script précédent deviendrait alors :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function test(){
var a = document.formulaire.input.value;
if (a <= 50) {
document.write(a +"  inférieur ou égal à 50");
}
else {
document.write(a + " est supérieur à 50");
}
}
//]]>
</script>
</head>
<body>
<form action="" name="formulaire">
<input type="text" name="input" size="2" maxlength="2" value="" />
<input type="button" value="Tester" onclick="(document.formulaire.input.value <=
50) ? document.write(document.formulaire.input.value + ’ est inférieur ou égal à
 50’) : document.write(document.formulaire.input.value + ’ est supérieur à 50’)"
 </form>
</body>
</html>
```

Le test conditionnel if ... else est un pilier du JavaScript et de la programmation. Il sera fréquemment utilisé dans les scripts d’une certaine complexité.

2. La boucle for

L’expression *for* permet d’exécuter un bloc d’instructions un certain nombre de fois (boucle) en fonction de la réalisation d’un certain critère. L’instruction for prévoit d’emblée un compteur et une condition pour l’interruption.

La syntaxe générale est :

```
for (valeur initiale ; condition ; progression) {
instructions;
}
```

Prenons un exemple concret :

```
for (i=0; i<5; i++) {
document.write(i + "<br>")
}
```

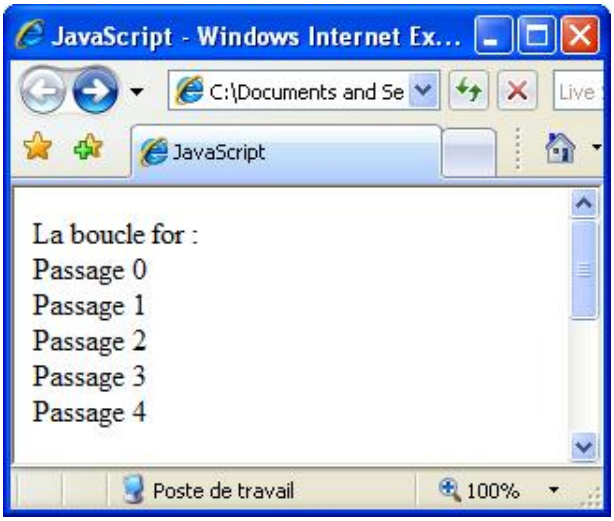
Au premier passage, la variable i vaut 0 (sa valeur initiale). Elle est bien inférieure à 5. Les instructions s’exécutent. La variable i est ensuite incrémentée d’une unité par l’opérateur d’incrémentation i++ (i vaut alors 1) et la boucle for continue son exécution.

La variable i (qui vaut 1) est toujours inférieure à 5. L’instruction est à nouveau exécutée. La variable est augmentée de 1 par l’incrémentation.

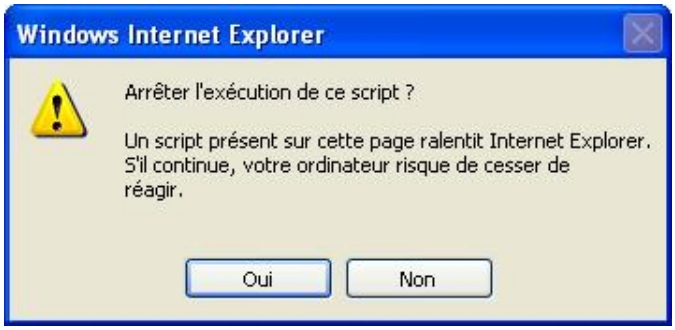
Et ainsi de suite jusqu’à obtenir 5 pour i. La variable i ne remplit alors plus la condition i inférieur à 5. La boucle s’interrompt et le programme continue après l’accolade de fermeture.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
```

```
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
La boucle for :<br />
<script type="text/JavaScript">
//<![CDATA[
for (i=0; i<5; i++) {
document.write("Passage " + i + "<br>")
}
//]]>
</script>
</body>
</html>
```



Le piège de la mise en place des boucles est de concevoir une boucle qui ne s'arrête jamais (boucle infinie). Un message d'erreur apparaît alors, pour que le navigateur rende la main.



Cette instruction *for* est très utilisée en JavaScript.

3. La boucle while

L'instruction *while* permet d'exécuter une instruction (ou un groupe d'instructions) un certain nombre de fois.

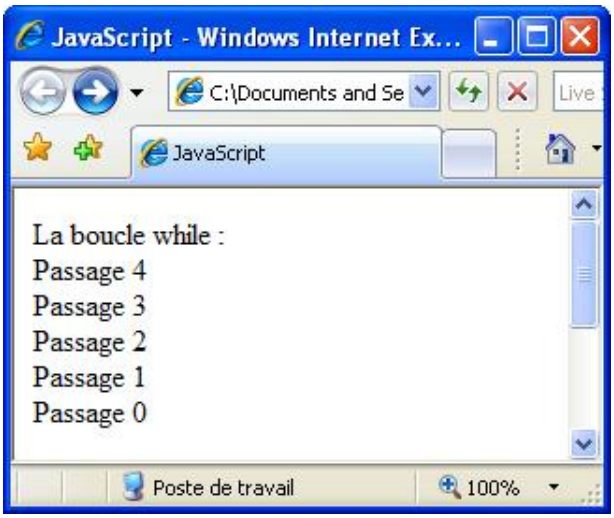
```
while (condition vraie) {
instruction(s)
}
```

Aussi longtemps que la condition est vérifiée, JavaScript continue à exécuter les instructions entre les accolades. Une fois que la condition n'est plus vérifiée, la boucle est interrompue et le script continue l'exécution.

Exemple

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
La boucle while :<br />
<script type="text/JavaScript">
//<![CDATA[
compt = 4;
while (compt>=0) {
document.write("Passage " + compt + "<br>");
compt--;
}
//]]>
</script>
</body>
</html>
```



Le résultat est identique à l'instruction *for*. Dans la pratique, l'instruction for est souvent préférée à while car elle intègre un compteur.

Le risque de boucle infinie est d'autant plus présent avec while que le concepteur doit créer et incrémenter lui-même le compteur.

4. L'instruction break

L'instruction break permet d'interrompre prématurément une boucle for ou while.

Pour illustrer ceci, reprenons notre exemple :

```
for (i=0; i<5; i++) {
if (i == 3)
break;
document.write("Passage " + i + "<br>")
}
document.write("fin de la boucle");
```

Le fonctionnement est semblable à l'exemple du point 2 sauf lorsque le compteur vaut 3. À ce moment, on sort de la boucle par l'instruction break et le message "fin de la boucle" s'affiche.

Ce qui donne à l'écran :

```
ligne : 0
ligne : 1
ligne : 2
fin de la boucle
```

5. L’instruction continue

L’instruction continue permet de sauter une instruction dans une boucle for ou while et de continuer ensuite le bouclage (sans sortir de celui-ci comme le fait break).

Soit l’exemple :

```
compt=0;
while (compt<5) {
if (compt == 3)
{
compt++
continue;
}
document.write ("ligne : " + compt + "<br>");
compt++;
}
document.write("fin de la boucle");
```

Ici, la boucle démarre. Lorsque le compteur vaut 3, l’instruction `document. write` ne s’exécute pas grâce à l’instruction `continue` (ligne : 3 n’est pas affiché) et la boucle poursuit son processus. Notons que compt a été incrémenté (compt++) avant l’instruction `continue` pour éviter un bouclage infini et que le navigateur ne rende la main.

Ce qui donne à l’écran :

```
ligne : 0
ligne : 1
ligne : 2
ligne : 4
fin de la boucle
```

Gestionnaires d'événement

Les gestionnaires d'événement vont apporter l'élément d'interactivité qui caractérise le JavaScript. Par leur intermédiaire, les actions du visiteur mettent en œuvre les outils de programmation du JavaScript.

La gestion de ces événements (déclenchés généralement par l'internaute) fait le lien entre le langage Xhtml et le JavaScript.

1. La notion d'événement

Passons en revue différents événements implémentés en JavaScript. Cette liste n'est cependant pas limitative.

`onAbort`

À l'arrêt du chargement de la page ou d'une image par le bouton "Stop" du navigateur.

`onBlur`

À la perte du focus par un élément Html (généralement lorsqu'on quitte un élément de formulaire).

`onClick`

Au clic de la souris sur un élément Html.

`onDbClick`

Au double clic de la souris sur un élément Html.

`onError`

À l'apparition d'une erreur dans la page.

`onFocus`

À la prise du focus d'un élément de formulaire.

`onKeyDown`

À la pression d'une touche du clavier.

`onKeyUp`

Au relâchement d'une touche du clavier qui a été enfoncée.

`onKeyPress`

À l'encodage par une touche du clavier (combinaison de onKeyDown et onKeyUp).

`onLoad`

Au chargement de la page par le navigateur.

`onMouseDown`

À la pression du bouton de la souris.

`onMouseMove`

Au déplacement de la souris.

`onMouseOut`

À l'abandon d'un élément Html par la souris.

`onMouseOver`

Au survol d'un élément Html par la souris.

`onMouseUp`

Au relâchement du bouton de la souris (suite de onMouseDown)

`onMove`

Au déplacement de la fenêtre.

`onReset`

Au clic du bouton "reset" (bouton Annuler) d'un formulaire.

`onResize`

Au redimensionnement de la fenêtre du navigateur.

`onScroll`

À l'utilisation de la barre de défilement.

`onSelect`

À la sélection d'un texte dans un élément Html.

`onSubmit`

Au clic du bouton *submit* (bouton Envoyer) d'un formulaire.

`onUnLoad`

Au moment où l'utilisateur quitte la page.

Ces événements peuvent être associés à une multitude de balises que nous ne détaillons pas ici.

Ces gestionnaires d'événements sont notés sous forme d'attributs dans les balises Html ou Xhtml. Ils ont d'ailleurs été repris par le W3C dans les standards du Html 4.0 et du Xhtml 1.0. Ils peuvent donc être notés dans le code, sans majuscule(s). C'est la convention que nous allons adopter par la suite.

La syntaxe est :

```
événement="function()"
```

Soit un événement (ils commencent tous par "on"), le signe égal et la fonction associée par le concepteur, entourée par des apostrophes.

Exemple

```
onclick="alert('Vous avez cliqué sur cet élément')"
```

De façon littérale, au clic de l'utilisateur, une boîte d'alerte s'ouvre avec le message indiqué dans le script.

2. L'événement onClick

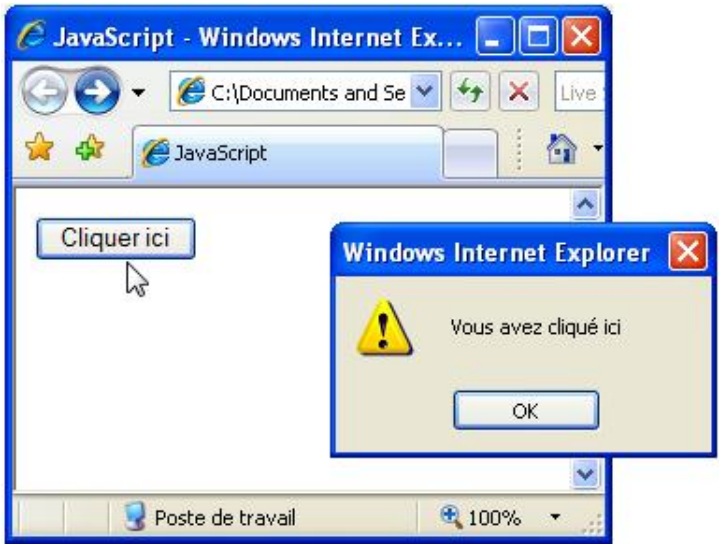
Le clic de la souris est un événement fréquemment utilisé. Il survient lorsque le visiteur clique, par exemple, un lien ou un élément de formulaire.

Exemple

Au clic du bouton, une boîte d'alerte surgit.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<form action="">
<input type="button" value="Cliquer ici" onclick="alert('Vous avez
cliqué ici')" />
</form>
</body>
</html>
```



Le clic de la souris, dans le cas d'un lien par la balise `<a> ... </a>`, est déjà utilisé par le Html ou le Xhtml pour atteindre la page spécifiée. Pour associer une action JavaScript au clic d'un lien, il a fallu introduire une notation particulière du genre :

```
<a href="javascript:alert('Bonjour !');">Cliquer ici !</a>
```

Ainsi dans l'attribut *href*, on indique au navigateur qu'il ne s'agit pas d'un lien mais d'une instruction JavaScript. D'où la notation javascript, double point et l'instruction retenue.

On rencontre aussi la notation :

```
<a href="javascript:void(0)" onclick="alert('Bonjour !'))"> Cliquer ici !</a>
```

Pour passer outre à toute action à partir de l'attribut href, l'expression `javascript:void(0)` est ici utilisée.

3. L'événement onFocus

Un événement est généré lorsqu'un élément est activé, par exemple une ligne de texte d'un formulaire.

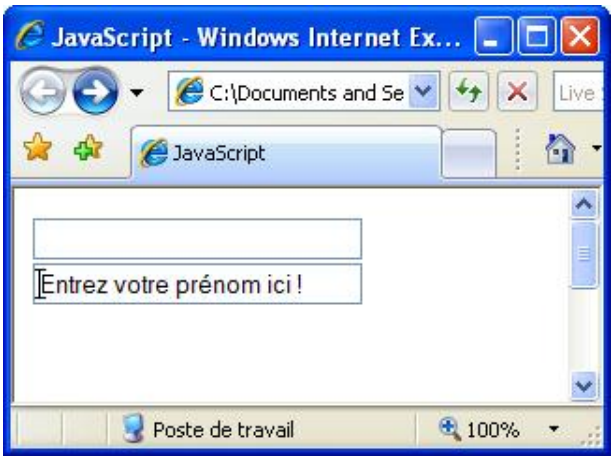
Exemple

Lorsque l'utilisateur clique dans une zone de texte, une inscription apparaît dans celle-ci.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<form action="" name="form">
```

```
<input name="t1" onfocus="document.form.t1.value='Entrez votre nom ici
!'" /><br />
<input name="t2" onfocus=" document.form.t2.value='Entrez votre prénom
ici !'" />
</form>
</body>
</html>
```

Après avoir cliqué dans la seconde ligne de texte, la capture d'écran suivante est obtenue.



4. L'événement onLoad et onUnLoad

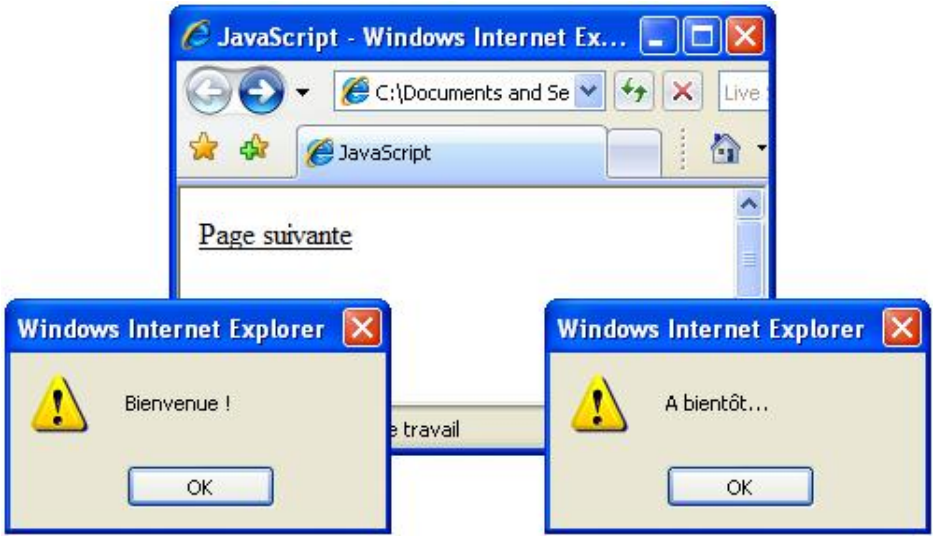
Au chargement (*load*) de la page ou à la sortie (onunload) de la page, des événements sont générés.

Exemple

Un message d'accueil au chargement de la page et un autre message à la sortie de celle-ci.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body onload="alert('Bienvenue !')" onunload="alert('A bientôt...')">
<a href="xxx.htm">Page suivante</a>
</body>
</html>
```

Ce script associe deux événements dans la même balise. Son écriture est simple. Il suffit d'encoder simplement les différents événements à la suite, car ils sont, depuis le Html 4.0, considérés comme de simples attributs.



5. L'événement onMouseOver et onMouseOut

L'événement `onMouseOver` se produit lorsque le pointeur de la souris passe au-dessus d'un lien ou d'une image, sans cliquer dessus.

L'événement `onMouseOut`, généralement associé à un événement `onMouseOver`, se produit lorsque le pointeur quitte la zone sensible (lien, image ou balise `<div>`).

Exemple

Ces deux événements sont souvent utilisés pour réaliser un effet, désormais classique, d'affichage d'une autre image au survol de l'image originale par le pointeur de la souris. Cet effet porte le nom de *rollover* dans la documentation anglo-saxonne.

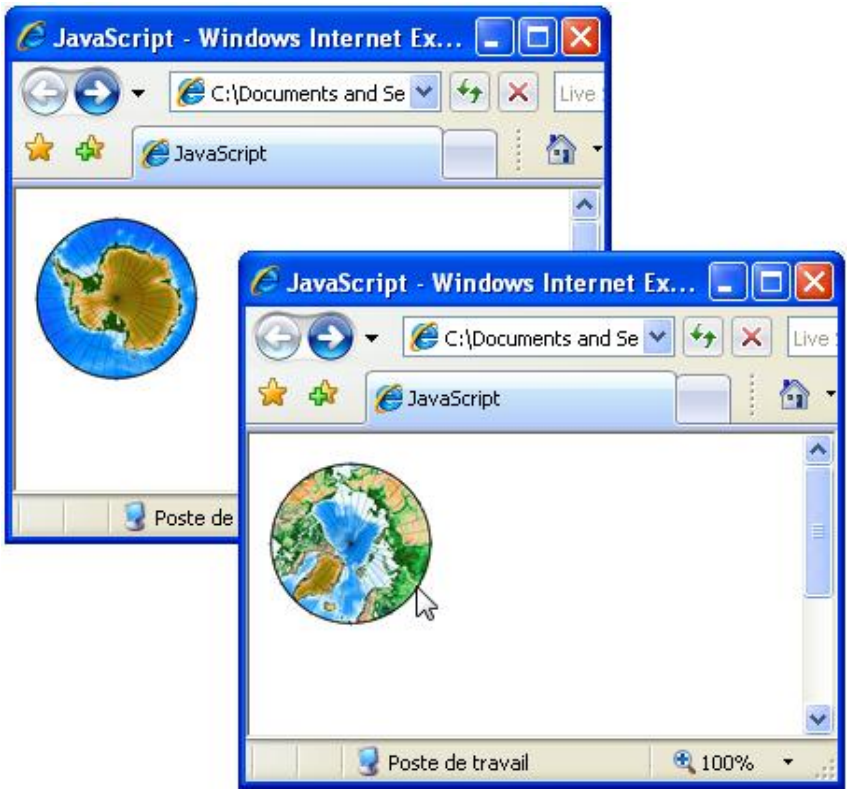
Il importe que la dimension des images concernées par le script soit rigoureusement identique en hauteur et largeur.

Soit les images :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<img name="image" alt="mappemonde" src="monde1.png" onmouseover="document.image.
src='monde2.png';" onmouseout="document.image.src='monde1.png';" />
</body>
</html>
```

La balise image `<img>` affiche l'image initiale monde1.png (`src="monde1. png"`). On a pris soin de donner un nom à cette balise image (`name="image"`). Au survol de la souris (onmouseover), l'image monde2.png apparaît. Pour ce faire, on

indique à JavaScript qu'il doit chercher dans le document l'objet "image" et l'attribut *src* qui fournit l'adresse de la nouvelle image. Soit le chemin complet, document.image.src. Lorsque le pointeur de la souris quitte l'image (onmouseout), l'image originale revient.



6. L'événement onSubmit

L'événement créé par le clic sur le bouton d'envoi d'un formulaire peut judicieusement être mis à profit pour, par exemple, effectuer des vérifications concernant l'encodage des champs de ce formulaire.

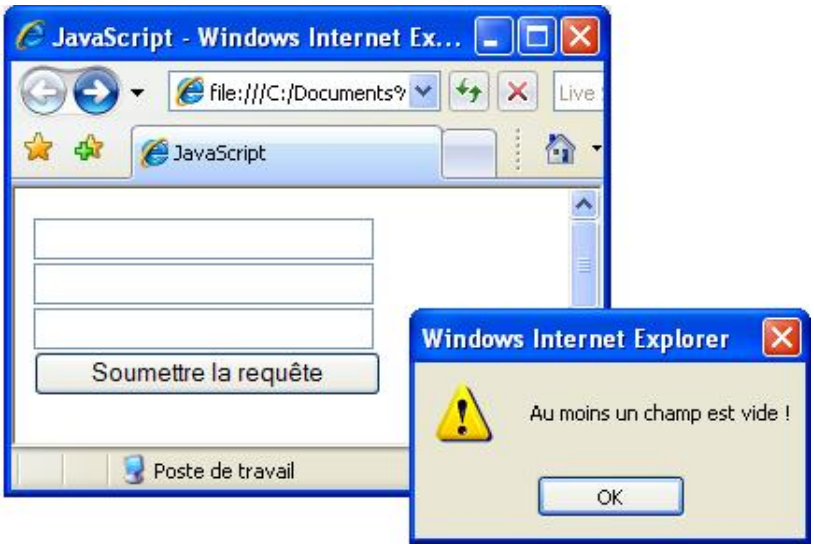
Exemple

Ce script vérifie si les champs du formulaire ont bien été complétés.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/javascript">
//<![CDATA[
function verif() {
a = document.formulaire.texte1.value;
b = document.formulaire.texte2.value;
c = document.formulaire.texte3.value;
if ((a && b && c) =="") {
alert("Au moins un champ est vide !");
}
}
//]]>
</script>
</head>
<body>
<form action="" name="formulaire" onsubmit="verif()">
<input name="texte1" size="26" /><br />
<input name="texte2" size="26" /><br />
<input name="texte3" size="26" /><br />
<input type="submit" />
```

```
</form>
</body>
</html>
```

Au clic du bouton d'envoi du formulaire, la fonction `verif()` est appelée (`onsubmit="verif()"`). Pour la lisibilité du script, la valeur des différentes lignes de texte du formulaire est sauvegardée dans une variable. L'accès à cette valeur s'effectue par l'écriture du chemin désormais habituel document.formulaire. textex. value. Par un test conditionnel *if*, on vérifie si une des lignes de texte est vide soit si a et b et c est vide (&& pour le et logique). Dans ce cas, une boîte d'alerte apparaît.



Formulaires

Avec JavaScript, les formulaires Html ou Xhtml prennent une toute autre dimension. N'oublions pas qu'en JavaScript, on peut accéder à chaque élément d'un formulaire pour y lire et/ou écrire une valeur, par exemple. Tous ces éléments renforcent les capacités interactives des pages Web.

1. La ligne de texte

La zone de texte est l'élément d'entrée/sortie par excellence de JavaScript.

La ligne de texte est créée par la balise <input type="text" />.

La ligne de texte possède trois propriétés :

Propriété	**Description**
name	Indique le nom du contrôle par lequel on pourra accéder.
defaultvalue	Indique la valeur par défaut qui sera affichée dans la zone de texte.
value	Indique la valeur en cours de la zone de texte. Soit celle encodée par l'utilisateur ou si celui-ci n'a pas rentré de valeur, la valeur par défaut.

a. Lire une valeur

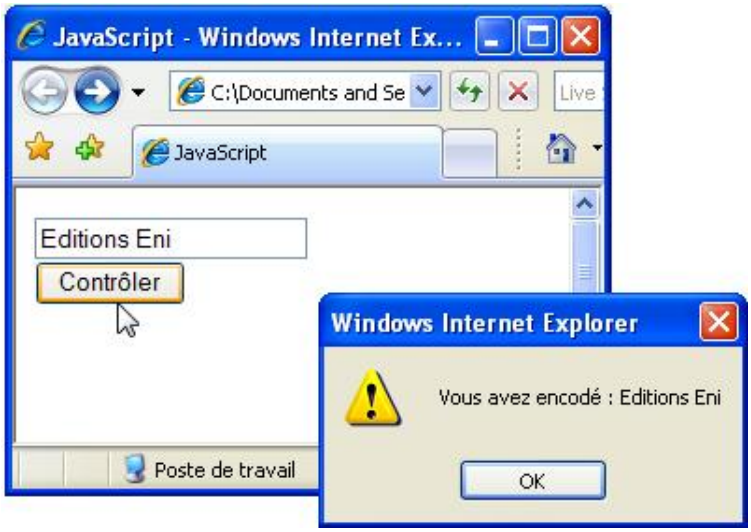
Encoder une valeur dans la zone de texte et la reproduire dans une boîte d'alerte.

Le script complet est :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function controle(formulaire) {
var test = document.formulaire.input.value;
alert("Vous avez encodé : " + test);
}
//]]>
</script>
</head>
<body>
<form action="" name="formulaire">
<input type="text" name="input" value="" /><br />
<input type="button" value="Contrôler" onclick="controle(formulaire)" />
</form>
</body>
</html>
```

Lorsque le bouton **Contrôler** est cliqué, JavaScript appelle la fonction `controle()` à laquelle le formulaire est transmis comme argument `onclick="controle (formulaire)"`.

Cette fonction, définie dans les balises `<head>`, prend sous la variable test, la valeur de la zone de texte `var test = document.formulaire.input.value`. Cette variable est alors fournie comme argument d'une boîte d'alerte.



b. Reproduire une valeur

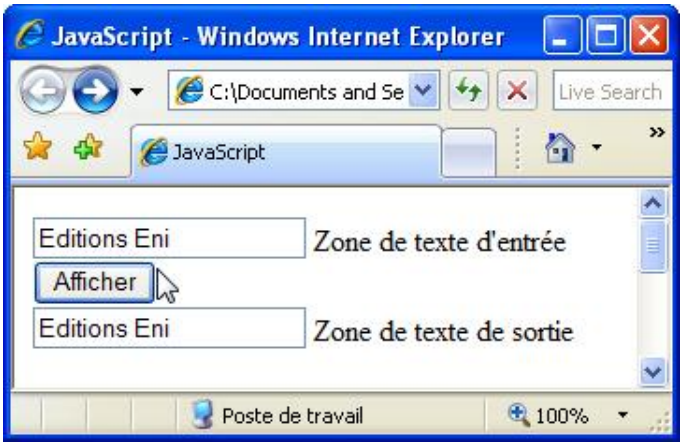
Entrer une valeur quelconque dans la zone de texte d'entrée. Afficher cette même valeur dans la zone de texte de sortie après avoir cliqué un bouton.

Voici le code :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function afficher(formulaire) {
var testin =document.formulaire.input.value;
document.formulaire.output.value=testin;
}
//]]>
</script>
</head>
<body>
<form action="" name="formulaire">
<input type="text" name="input" value="" /> Zone de texte d'entrée <br />
<input type="button" value="Afficher" onclick="afficher(formulaire)" /><br />
<input type="text" name="output" value="" /> Zone de texte de sortie
</form>
</body>
</html>
```

Lorsque le bouton **Afficher** est cliqué, JavaScript appelle la fonction `afficher()` à laquelle le formulaire est transmis comme argument (`onclick="afficher (formulaire)"`).

Cette fonction `afficher()` prend sous la variable testin, la valeur de la zone de texte d'entrée (document.formulaire.input.value). À l'instruction suivante, la variable testin est affectée à la valeur de la zone de texte de sortie en JavaScript :(`document.formulaire.output.value = testin`).



c. Tester un formulaire vide

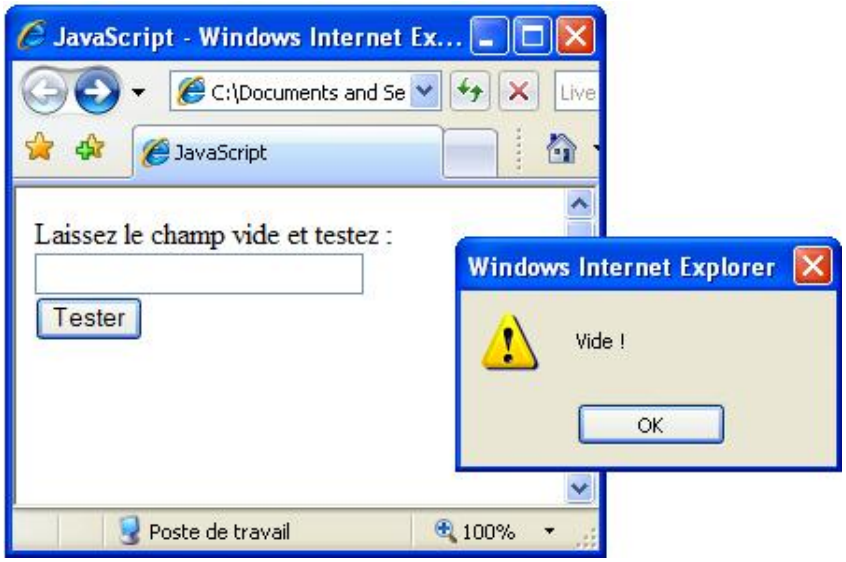
Il est parfois impératif qu'un élément de formulaire soit rempli par le visiteur. Ce petit script teste si le visiteur a omis d'y encoder des données.

Nous avons déjà abordé cette problématique dans le chapitre précédent mais nous utilisons ici une autre façon de procéder.

Le script est :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function valider(){
input = document.form.entree.value.length;
if (input == 0){
alert("Vide !");
}
}
//]]>
</script>
</head>
<body>
<form action="" name="form">
Laissez le champ vide et testez :<br />
<input type="text" name="entree" size="25" value="" /><br />
<input type="button" value="Tester" onclick="valider()" />
</form>
</body>
</html>
```

La fonction `valider()` appelée par le clic du bouton (`onclick="valider()"`) récupère dans la variable `input`, la longueur de ce qui est encodé dans la zone de texte (`input = document.form.entree.value.length`). Si cette longueur est égale à 0 (`input == 0`), l'utilisateur est averti par une boîte d'alerte que la zone est vide.

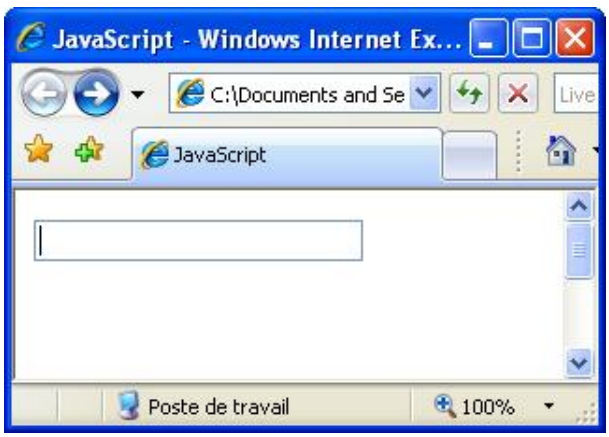


d. Donner le focus

Il est souvent convivial de placer directement le curseur dans la première zone de texte d'un formulaire.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body onload="document.form.entree.focus();">
<form action="" name="form" >
<input type="text" name="entree" size="25" value="" />
</form>
</body>
</html>
```

Au chargement de la page (événement *onload* du body), le focus est donné à la zone de texte dont le chemin a été déterminé par `document.form.entree`.



Le curseur est bien positionné directement dans la zone de texte.

e. Encodage d'un nombre

Certaines valeurs encodées ne peuvent comporter que des nombres.

Ce script permet de le vérifier.

La fonction JavaScript isNaN(argument) est utilisée : elle évalue l'argument pour déterminer s'il ne s'agit pas d'un nombre (NaN pour *Not a Number*).

Le code devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function verif() {
n=document.form.nombre.value;
if (isNaN(n) || n == 0) {
alert("Entrer un nombre SVP !");
}
else {
alert("Entrée valide");
}
}
//]]>
</script>
</head>
<body>
<form action="" name="form">
Code postal : <input type="text" name="nombre" size="3" maxlength="5" /><br />
<p><input type="button" name="bouton" value="Tester" onclick="verif()" /><p>
</form>
</body>
</html>
```

Une zone de texte de 5 caractères maximum (`maxlength="5"`) est prévue pour l'encodage du code postal. Celui-ci doit comporter uniquement des chiffres et donc doit pouvoir être interprété comme un nombre. Le clic sur le bouton **Tester** appelle la fonction `verif()`.

Cette fonction initialise d'abord la variable `n` avec la valeur de la zone de texte dédiée au code postal (`document.form.nombre.value`). Un test conditionnel (`if`) est effectué pour vérifier si la valeur de `n` n'est pas un nombre (`isNaN(n)`) ou est vide (`n == 0`). Dans ce cas, l'encodage du code postal est incorrect et une boîte d'alerte invite le visiteur à entrer un nombre correct. Sinon, l'entrée est valide.

f. Calcul automatique

Les zones de texte peuvent également être utilisées pour recevoir des données ou contenir le résultat d'opérations

effectuées par le script.

Dans l'exemple suivant, nous allons demander au visiteur la quantité de produits commandés. Au clic dans la zone de texte du prix total, celui-ci est automatiquement affiché.

Il est peut-être préférable, à ce stade, de déjà jeter un coup d'oeil à la capture d'écran suivante.

Le fichier devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function calcul(form) {
a = document.form.elem1.value;
b= document.form.elem2.value;
document.form.elem3.value = a* b;
}
function annul(form) {
document.form.elem1.value="";
document.form.elem3.value="";
}
//]]>
</script>
</head>
<body>
<form action="" name="form">
<table border="1" width="300">
<tr>
<td width="100">
<p align="center">Quantité</p>
</td>
<td width="100">
<p align="center">Prix unitaire</p>
</td>
<td width="100">
<p align="center">Prix total</p>
</td>
</tr>
<tr>
<td>
<p align="center"><input type="text" name="elem1" size="2" value="" />
</p>
</td>
<td>
<p align="center"><input type="hidden" name="elem2" value="2.56" />2,56
```

```
€</p>
</td>
<td>
<p align="center"><input type="text" name="elem3" size="5" value=""
onfocus="calcul(form)" onblur="annul(form)" /> €</p>
</td>
</tr>
</table>
</form>
Cliquez n'importe où pour réinitialiser le tableau.
</body>
</html>
```

En cliquant dans la zone de texte du prix total, soit en donnant le focus, la fonction calcul() est appelée (`onfocus="calcul(form)"`).

Dans cette fonction `calcul()`, la valeur encodée dans la zone de texte relative aux quantités (`a = document.form.elem1.value`) est sauvegardée dans la variable a et la valeur du prix unitaire encodée dans l'élément de formulaire caché `<input type="hidden" />` (`b = document.form.elem2.value`) est sauvegardée dans la variable b. La multiplication de a et de b donne le prix total affiché dans la zone de texte prévue à cet effet (`document.form.elem3.value = a * b`).

Nous nous sommes servi de ce script pour illustrer l'événement onBlur. À la fin du script, la zone de texte du prix total a le focus. En cliquant n'importe où dans la page, cette zone perd le focus. C'est ce que désigne le terme de *Blur*. À cet événement est associée la fonction annul(form) : `onblur="annul(form)"`.

Cette fonction remet une chaîne de caractères vides dans les deux zones de texte (`document.form.elem1.value=""` et `document.form.elem3.value=""`). Ce qui réinitialise le script.

2. Les boutons de choix unique

Les boutons de choix unique, appelés aussi boutons radio, sont utilisés pour noter un choix, et seulement un seul, parmi un ensemble de propositions.

Les boutons de choix unique sont créés par les balises <input type= "radio" />.

Propriété	**Description**
name	Indique le nom du contrôle. Tous les boutons portent le même nom.
index	L'index ou le rang du bouton radio démarrant à 0.
checked	Indique l'état en cours de l'élément radio (sélectionné ou non).
defaultchecked	Indique l'état du bouton sélectionné par défaut.

value	Indique la valeur de l'élément radio.

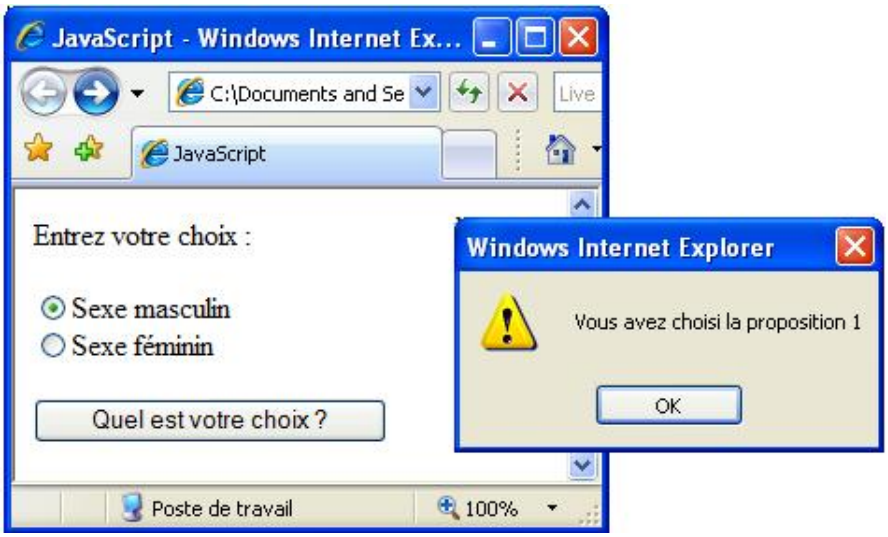
Exemple

Un formulaire renseignant le sexe du visiteur.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>>
<title>JavaScript</title>
<meta http-equiv="Content-Type"Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function choixsexe(form) {
if (form.choix[0].checked) {
alert("Vous avez choisi la proposition " + form.choix[0].value);
}
if (form.choix[1].checked) {
alert("Vous avez choisi la proposition " + form.choix[1].value);
}
}
//]]>
</script>
</head>
<body>
Entrez votre choix :
<form name="form" action="">
<input type="radio" name="choix" value="1" />Sexe masculin<br />
<input type="radio" name="choix" value="2" />Sexe féminin<br />
<p><input type="button" value="Quel est votre choix ?"
onclick="choixsexe(form)" /></p>
</form>
</body>
</html>
```

Dans le formulaire nommé `form`, deux boutons radio sont déclarés. Notez que le même nom est utilisé pour les deux boutons. Ensuite, un bouton déclenche par `onclick` la fonction `choixsexe()`.

Cette fonction vérifie le bouton radio coché. Ces données des boutons sont disponibles grâce aux indices établis par rapport au nom des boutons radio soit choix[0], choix[1]. La propriété *checked* du bouton est testée par `if (form.choix[x].checked)`. Dans l'affirmative, une boîte d'alerte s'affiche. Ce message reprend la valeur (voir value) attachée à chaque bouton par le chemin form.choix[x].value.



3. Les boutons de choix multiples

Les boutons de choix multiples (aussi appelés *checkbox*) sont utilisés pour noter un ou plusieurs choix parmi un

ensemble de propositions.

Les boutons de choix multiples sont créés par la balise <input type="checkbox" />.

Propriété	Description
name	Indique le nom du contrôle. Toutes les cases à cocher portent un nom différent.
checked	Indique l'état en cours de l'élément case à cocher (sélectionné ou non).
defaultchecked	Indique l'état du bouton sélectionné par défaut.
value	Indique la valeur de l'élément case à cocher.

Exemple

Soit la question à choix multiples suivante :

```
Sélectionner les balises Html :
- <h1> ... </h1>.
- <a> ... </a>.
- <op> ... </op>.
- <b> ... </b>.
```

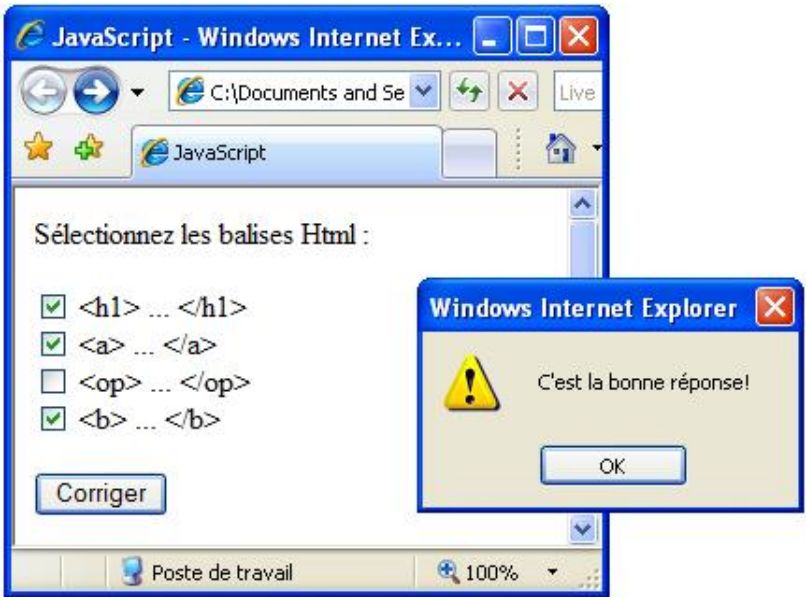
La bonne réponse est la première, seconde et quatrième proposition.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>>
<title>JavaScript</title>
<meta http-equiv="Content-Type"Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function choixsexe(form) {
if (form.choix[0].checked) {
alert("Vous avez choisi la proposition " + form.choix[0].value);
}
if (form.choix[1].checked) {
alert("Vous avez choisi la proposition " + form.choix[1].value);
}
}
//]]>
</script>
</head>
<body>
Entrez votre choix :
<form name="form" action="">
<input type="radio" name="choix" value="1" />Sexe masculin<br />
<input type="radio" name="choix" value="2" />Sexe féminin<br />
<p><input type="button" value="Quel est votre choix ?"
onclick="choixsexe(form)" /></p>
</form>
</body>
</html>
```

Dans le formulaire nommé `form`, quatre cases à cocher sont déclarées. Notez qu'un nom différent est utilisé pour les quatre boutons. Vient ensuite un bouton qui déclenche `onclick` la fonction reponse().

Cette fonction teste les cases à cocher sélectionnées. Pour avoir la bonne réponse, il faut que les cases 1, 2 et 4 soient cochées. Le nom des cases à cocher permet de les identifier, notamment la propriété *checked* du bouton par `form.nom_de_la_case.checked`. Dans l'affirmative, la valeur renvoyée est *true*. Dans la négative, la valeur renvoyée est *false*.

Un test conditionnel vérifie par le "et logique" (&&) si les propositions 1, 2 et 4 sont vraies et la proposition 3 est fausse. Dans ce cas, une boîte d'alerte s'affiche pour annoncer la bonne réponse. Dans la négative, une autre boîte d'alerte invite à refaire le test.



4. Le menu déroulant

La liste de sélection permet de proposer diverses options sous la forme d'un menu déroulant, dans lequel l'utilisateur peut sélectionner une option. Une fois l'option sélectionnée, elle reste alors affichée.

Le menu déroulant est créé par la balise `<select> ... </select>` et les éléments de la liste par une ou plusieurs balise (s) `<option> ... </option>`.

Propriété	Description
name	Indique le nom de la liste déroulante.
length	Indique le nombre d'éléments de la liste.
selectedIndex	Indique le rang à partir de 0 de l'élément de la liste qui a été sélectionné par l'utilisateur.
defaultselected	Indique l'élément de la liste sélectionné par défaut.

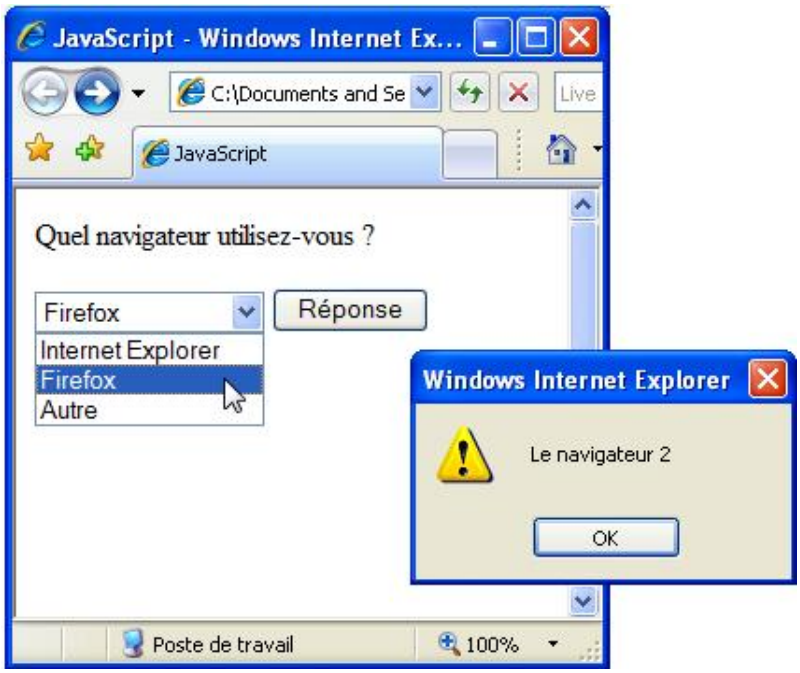
Un exemple habituel :

Une liste déroulante permet de choisir le navigateur. Les options proposées sont Internet Explorer, Firefox et autre.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function liste(form) {
alert("Le navigateur " + (form.list.selectedIndex + 1));
}
//]]>
</script>
</head>
<body>
Quel navigateur utilisez-vous ?
<form name="form" action="">
<select name ="list">
<option value="1">Internet Explorer</option>
<option value="2">Firefox</option>
<option value="3">Autre</option>
```

```
</select>
<input type="button" value="Réponse" onclick="liste(form)" />
</form>
</body>
</html>
```

Dans le formulaire nommé form, on déclare une liste de sélection par la balise `<select> ... </select>`. Entre ces balises, on déclare les différentes options de la liste par autant de balises `<option> ... </option>`. Ensuite un bouton déclenche la fonction `liste()`. Cette fonction fait ressortir l'option sélectionnée. Le chemin complet de l'élément sélectionné est `form.list.selectedIndex`. Comme l'index commence à 0, il faut ajouter 1 pour obtenir le rang exact de l'élément.



5. Le bouton d'envoi

Le bouton d'envoi, créé par la balise `<input type="submit" />`, permet de transmettre les données du formulaire selon les spécifications de l'attribut `action="..."` de la balise `<form>`.

Propriété	Description
name	Indique le nom du contrôle.
value	Par défaut, le texte du bouton d'envoi est déterminé par votre navigateur, pour exemple **Soumettre la requête** sous Internet Explorer et **Envoyer** sous Firefox. Il est cependant possible de personnaliser le texte par défaut du bouton par l'attribut value="valeur".
disabled	Permet de désactiver le bouton d'envoi.

6. Le bouton de réinitialisation

Il est utile de prévoir pour l'utilisateur la possibilité d'annuler tout encodage effectué dans le formulaire et ainsi de réinitialiser (*reset*) un formulaire vierge.

Cette opération est réalisée par la balise `<input type="reset" />`.

Tout comme pour le bouton de soumission, il est possible de modifier le texte par défaut du bouton par l'attribut `value="valeur"`.

Les autres attributs sont identiques à ceux du bouton d'envoi.

7. Le bouton de commande

Le bouton d'envoi et le bouton de réinitialisation, ont des fonctions bien définies par le langage Html ou Xhtml, ce sont respectivement les fonctions consistant à envoyer le formulaire selon les instructions définies par l'attribut *action* de la balise `<form>` ou à réinitialiser un formulaire vide d'encodage.

Il faut cependant prévoir des boutons dont la fonction pourrait être définie par le concepteur de la page, généralement grâce au JavaScript.

Pour ce faire la balise `<input type="button" />` (qui prise isolément ne fait rien) est utilisée. L'action est alors définie par un gestionnaire d'événement, généralement du type *onclick*.

Propriété	Description
name	Avec l'attribut name="nom", vous pouvez attribuer un nom au bouton. C'est grâce à ce nom que JavaScript peut accéder au bouton.
value	Permet de définir le texte du bouton. L'attribut value est, dans le cas présent, obligatoire.

Exemple

Certains utilisateurs (impatients) ont la fâcheuse manie de cliquer plusieurs fois sur le bouton d'envoi. Ce script calmera leur ardeur.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
var nombreclic=0;
function compteclic(form) {
nombreclic++;
if (nombreclic>1) {
alert("Vous avez déjà cliqué ce bouton.\nLe formulaire est en cours
de traitement.");
}
}
//]]>
</script>
</head>
<form name="form" action="">
<input type="button" value="Cliquez-moi !"
 onclick="compteclic(form)" />
</form>
</body>
</html>
```

Au clic du bouton d'envoi, la fonction `compteclic()` est appelée (`compteclic(form)`).

Cette fonction incrémente d'une unité (`nombreclic++`) la variable nombreclic qui a été préalablement initialisée à 0 (`var nombreclic=0`). Le script effectue alors un test (`if`) pour vérifier si la variable nombreclic est supérieure à 1 (`nombreclic>1`). Dans ce cas, une boîte d'alerte s'affiche (`alert("Vous avez déjà cliqué ce bouton)`).

8. L'instruction this

L'instruction this est un raccourci qui permet de référencer l'objet courant. Elle est souvent utilisée lorsque du code JavaScript est utilisé au sein d'une balise Html, ce qui permet alors de faire référence à l'objet défini par cette balise.

Exemple

```
<form name="form">
<input type="button" value="Cliquez-moi !"
onclick="compteclic(this.form)">
</form>
```

ou

```
<form name="form">
<input type="button" value="Cliquez-moi !"
onclick="compteclic(this)">
</form>
```

Manipulation des chaînes de caractères

L'objet`String` permet de manipuler des chaînes de caractères. JavaScript met à la disposition du programmeur, une série de propriétés et de méthodes qui permettent de manipuler les caractères de la chaîne.

La manipulation des caractères est très utilisée dans les applications AJAX.

Instruction	Description
`length`	Propriété qui renvoie un entier indiquant la longueur de la chaîne de caractères.
`charAt()`	Méthode qui permet d'accéder à un caractère isolé d'une chaîne.
`indexOf()`	Méthode qui renvoie la position d'une chaîne partielle à partir d'une position déterminée, en commençant au début de la chaîne, soit en position 0.
`lastIndexOf()`	Méthode qui renvoie la position d'une chaîne partielle à partir d'une position déterminée, en commençant à la fin de la chaîne, soit en position length - 1.
`subString(x,y)`	Méthode qui renvoie une chaîne partielle située entre la position x et la position y-1.
`replace(x,y)`	Remplace x par y.
`toLowerCase()`	Transforme toutes les lettres en minuscules.
`toUpperCase()`	Transforme toutes les lettres en majuscules.

1. Length()

La propriété *length* retourne un entier qui indique le nombre d'éléments dans une chaîne de caractères. Si la chaîne est vide (""), le nombre vaut zéro.

La syntaxe est simple :

```
X = variable.length;
X = ("chaîne de caractères").length;
```

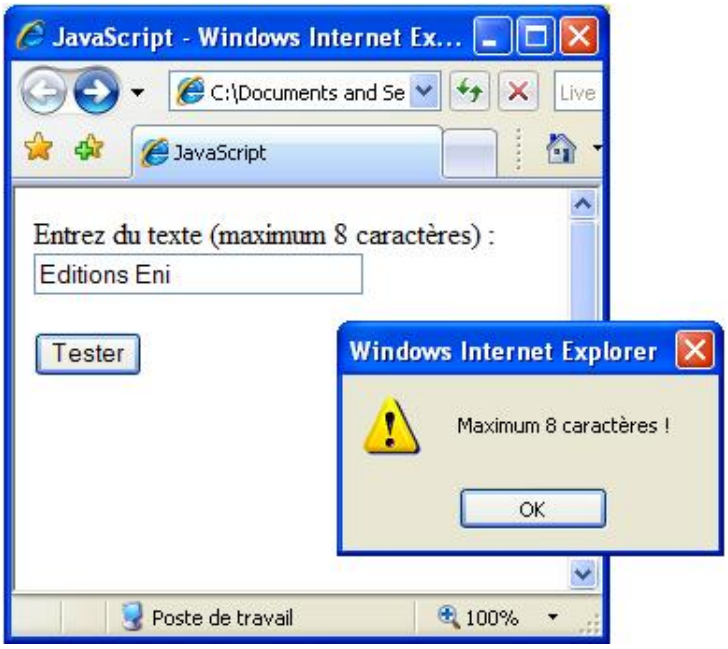
Exemple

Ce script vérifie que les caractères encodés dans une zone de texte ne dépassent pas la limite fixée à huit caractères.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function valider(){
input = document.form.entree.value;
if (input.length>8) {
alert("Maximum 8 caractères !");
}
}
//]]>
</script>
</head>
```

```
<body>
<form name="form" action="">
Entrez du texte (maximum 8 caractères) :<br />
<input type="text" name="entree" size="25" value="" />
<p><input type="button" value="Tester" onclick="valider()" /></p>
</form>
</body>
</html>
```

La fonction `valider()` reprend d'abord dans la variable `input` la valeur de la zone de texte (`input = document.form.entree.value`). La longueur de la chaîne de caractères est fournie par la propriété *length* associée à la variable `input` (`input.length`). Grâce à un test, on vérifie si cette longueur dépasse les 8 caractères, dans ce cas une boîte d'alerte est déclenchée.



Il faut noter que la propriété *length* est valable pour les chaînes de caractères, mais aussi pour connaître la longueur ou le nombre d'éléments :

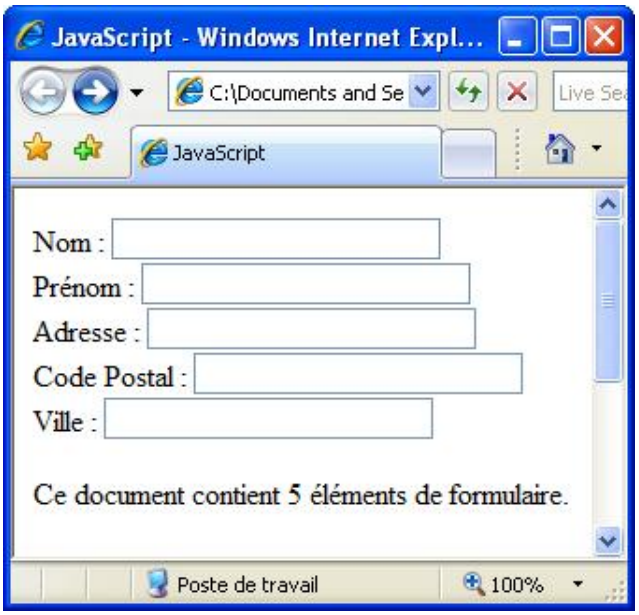
- de formulaires. Combien y a-t-il de formulaires différents dans le document ?
- de boutons d'option. Combien y a-t-il de boutons radio dans le formulaire ?
- de cases à cocher. Combien y a-t-il de cases à cocher dans le formulaire ?
- d'options dans un menu déroulant. Combien y a-t-il d'options dans une balise `<select>` ?
- de cadres, d'ancres, de liens,
- etc.

Exemple

Ce script indique le nombre d'éléments de formulaire présents dans le document Xhtml.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
```

```
<form name="form" action="">
Nom : <input type="text" size="25" /><br />
Prénom : <input type="text" size="25" /><br />
Adresse : <input type="text" size="25" /><br />
Code Postal : <input type="text" size="25" /><br />
Ville : <input type="text" size="25" />
</form>
<script type="text/JavaScript">
//<![CDATA[
a="Ce document contient " + document.form.length + " éléments de
formulaires.";
document.write(a);
//]]>
</script>
</body>
</html>
```



2. charAt()

La méthode `chartAt()` retourne la lettre ou le signe qui occupe une position déterminée dans une chaîne de caractères. Il faut lui fournir en paramètre la position souhaitée.

Il faut d'abord noter que les caractères sont comptés de gauche à droite et que la position du premier caractère est 0. La position du dernier caractère est donc la longueur totale (*length*) de la chaîne de caractère moins 1;

chaîne : `JavaScript`

position : 0123456789 (dernier caractère = 9 soit longueur totale - 1)

Si la position indiquée en paramètre est inférieure à zéro ou plus grande que la longueur moins 1, JavaScript retourne une chaîne vide.

La syntaxe de `charAt()` est :

```
resultat = chaine_départ.charAt(x);
```

où x est un entier compris entre 0 et la longueur de la chaîne à analyser moins 1.

Notez les exemples suivants :

```
var chaine="Javascript";
resultat=chaine.charAt(0);
resultat="Javascript".charAt(0);
```

La réponse est "J"

```
var str="Javascript";
var chr=str.charAt(9);
var chr=charAt(str,9);
```

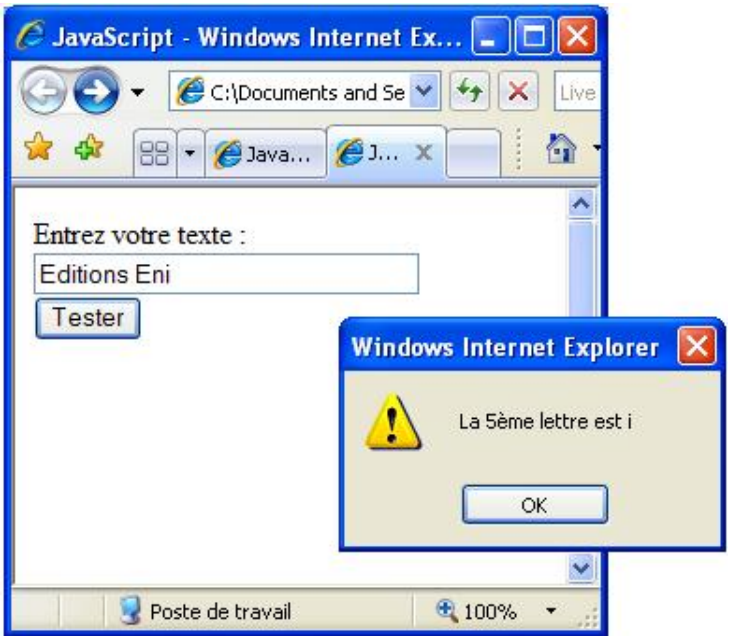
La réponse est "t"

Exemple

Ce script retourne la lettre en 5ème position.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function valider() {
var x=document.form.texte.value;
var a=x.charAt(4);
alert("La 5ème lettre est " +a);
}
//]]>
</script>
</head>
<body>
<form name="form" action="">
Entrez votre texte :<br />
<input type="text" name="texte" size="30" /><br />
<input type="button" value="Tester" onclick="valider()" />
</form>
</body>
</html>
```

Le texte encodé dans la zone de texte est sauvegardé dans la variable `x`. La méthode charAt est appliquée à cette dernière (`x.charAt(4)`). Notez le paramètre 4 qui correspond en JavaScript à la 5ème position.



3. indexOf()

Cette méthode `indexOf()` retourne la position d'une chaîne partielle (lettre unique, groupe de lettres ou mot) dans une chaîne de caractères. Cette chaîne partielle est transmise en paramètre.

Il est possible, mais facultatif et peu retenu dans la pratique, de transmettre comme second paramètre la position à partir de laquelle la recherche doit commencer. S'il n'est pas spécifié, la recherche commence à la position 0.

Pour chercher l'arobase dans une chaîne de caractères, la syntaxe est :

```
variable="chaîne_de_caractères";
x=variable.indexOf("@");
```

Commentaires :

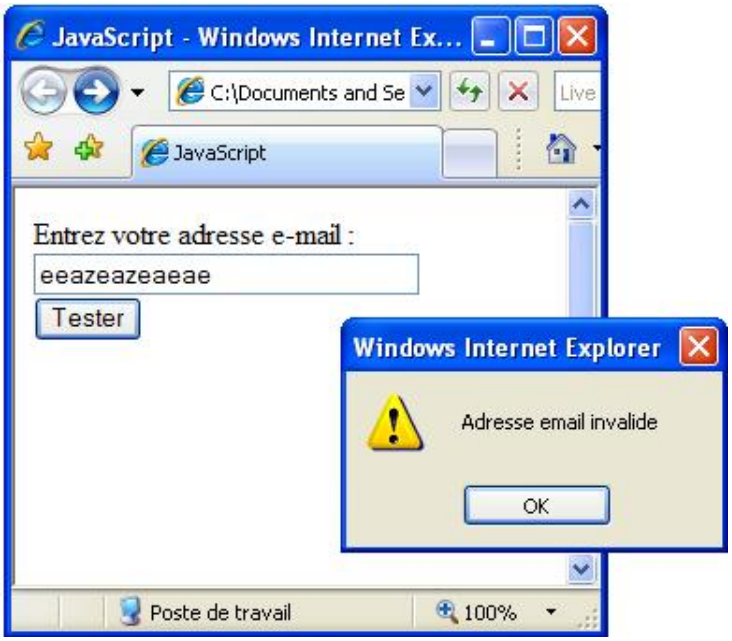
- la position retournée est celle de la première occurrence de la chaîne partielle dans la chaîne de caractères.
- si la chaîne partielle n'est pas trouvée dans la chaîne de caractères à analyser, la valeur retournée est égale à -1.

Exemple

Ce script teste si une adresse e-mail contient le signe @. Il notifiera un message d'alerte si le signe @ n'est pas trouvé (soit valeur -1).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function valider() {
var a=document.form.email.value.indexOf("@");
if (a == -1) {
alert("Adresse email invalide");
}
else {
alert("OK");
}
}
//]]>
</script>
</head>
<body>
<form name="form" action="">
Entrez votre adresse e-mail :<br />
<input type="text" name="email" size="30" /><br />
<input type="button" value="Tester" onclick="valider()" />
</form>
</body>
</html>
```

L'adresse e-mail est fournie par `document.form.email.value`. La méthode `indexOf` est appliquée avec le signe @ à rechercher transmis en paramètre. Si l'arobase a été trouvée, la méthode retourne sa position qui ne peut être qu'un chiffre positif. Si la position retournée vaut -1, cela indique que l'arobase n'a pas été trouvée et que l'adresse e-mail n'est donc pas valide.



Ce test est assez sommaire et peut être affiné. Outre l'arobase, une adresse e-mail contient également au moins un point. Le script suivant va tester la présence du point et ne déclarer l'e-mail valide que si les deux conditions sont remplies.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function valider() {
var a=document.form.email.value.indexOf("@");
if (a == -1) {
alert("Adresse email invalide");
}
else {
alert("OK");
}
}
//]]>
</script>
</head>
<body>
<form name="form" action="">
Entrez votre adresse e-mail :<br />
<input type="text" name="email" size="30" /><br />
<input type="button" value="Tester" onclick="valider()" />
</form>
</body>
</html>
```

Dans la variable a, on obtient la position du signe @ ou son absence (-1). Idem pour la variable b qui fournit la position du point ou son absence (-1). L'absence de l'arobase ou du point détermine l'adresse e-mail non valide.

Pour des procédures de validation de formulaires plus sophistiquées, reportez-vous au point Les expressions de cette section.

4. LastIndexOf()

Cette méthode ressemble fortement à `indexOf()` sauf que la recherche s'effectue cette fois de droite à gauche (en

commençant donc par la fin).

Notons que même lorsque la recherche s'effectue à partir de la fin de la chaîne, la position retournée est numérotée depuis le début de la chaîne avec la numérotation commençant à zéro.

```
x=variable.lastIndexOf(chaîne_partielle);
```

Les exemples suivants montrent la différence entre indexOf() et lastIndexOf() :

Soit `variable="Javascript"`. On recherche la position de la lettre a.

`x = variable.indexOf(a)` retourne la position 1.

`x=variable.lastIndexOf(a)` retourne la position 3 (pour le second a).

Cette méthode est peu utilisée dans la pratique.

5. Substring()

La méthode `substring()` sera particulièrement utile pour extraire des données d'une chaîne de caractères.

La syntaxe est `substring(x,y)` où `x` désigne la position du premier signe à extraire dans la chaîne de caractères et y la position du premier signe ne devant plus être extrait de la chaîne de caractères. Pour rappel, la numérotation commence à 0.

Si x est égal à y, substring() retourne (logiquement) une chaîne vide.

Voici quelques exemples :

```
Javascript
||||||||||
0123456789

str="Javascript";
str1=str.substring(0,4);
str2=str.substring(6,9);
str3=str.substring(7,10);
```

Les résultats sont :

```
str1="Java"; soit les positions 0,1, 2 et 3.
str2="rip"; soit les positions 6, 7 et 8.
str3="ipt" soit les positions 7, 8 et 9.
```

Exemple

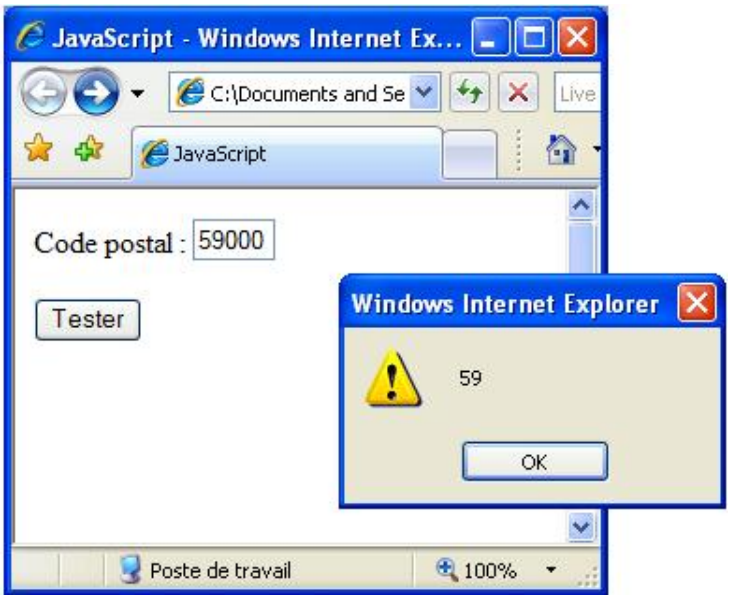
Retrouver les deux premiers chiffres d'un code postal.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function extraire() {
dep = document.form.cp.value;
dep = dep.substring(0,2);
alert(dep);
document.form.cp.value = "";
}
//]]>
</script>
</head>
<body>
<form name="form" action="">
Code postal :
<input type="text" name="cp" size="3" maxlength="5" /><br />
```

```
<p><input type="button" name="bouton" value="Tester"
onclick="extraire()" /></p>
</form>
</body>
</html>
```

La fonction `extraire()` récupère les 5 chiffres du code postal encodé dans la zone de texte (`document.form.cp.value`). Les deux premiers chiffres sont extraits (`dep.substring(0,2)`) et sont affichés dans une boîte d'alerte.

La zone de texte est réinitialisée à 0 par `document.form.cp.value = ""`.



6. toLowerCase()

Cette méthode transforme tous les caractères (majuscules ou minuscules) d'une chaîne de caractères en minuscules.

```
variable="chaîne de caractères";
x=variable.toLowerCase();
```

Exemple

```
str="JavaScript";
str1=str.toLowerCase();
```

Le résultat sera :

```
str1="javascript";
```

7. toUpperCase()

Cette méthode transforme les caractères (majuscules ou minuscules) d'une chaîne de caractères en majuscules.

```
variable="chaîne de caractères";
x=variable.toUpperCase();
```

Exemple

```
str="JavaScript";
str2=str.toUpperCase();
```

Le résultat sera :

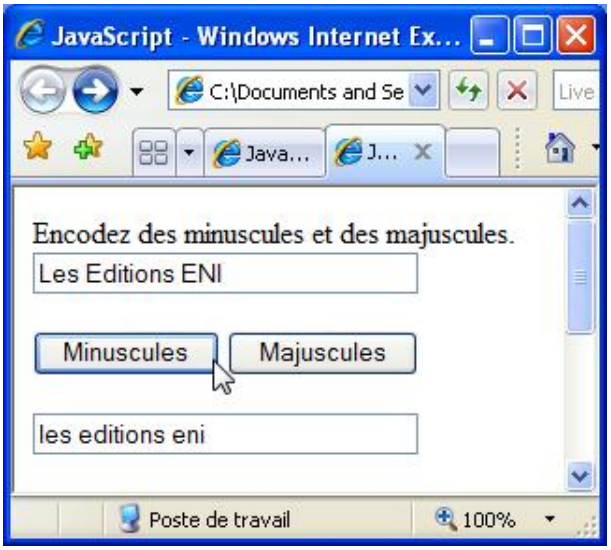
```
str2="JAVASCRIPT";
```

Exemple

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function minuscules() {
a = document.formulaire.entree.value.toLowerCase();
document.formulaire.sortie.value = a;
}
function majuscules() {
a = document.formulaire.entree.value.toUpperCase();
document.formulaire.sortie.value = a;
}
//]]>
</script>
</head>
<body>
<form name="formulaire" action="">
Encodez des minuscules et des majuscules.<br />
<input name="entree" size="30" /><br />
<input type="button" value="Minuscules" onclick="minuscules()" />
<input type="button" value="Majuscules" onclick="majuscules()" />
<br />
<input name="sortie" size="30" />
</form>
</body>
</html>
```

Pour la fonction `minuscules()`, la variable a repris le contenu de la zone de texte d'entrée (`document.formulaire.entree.value`) auquel la méthode `toLowerCase()` est appliquée. Le résultat est renvoyé dans la zone de texte de sortie (`document.formulaire.sortie.value`).

Pour la fonction `majuscules()`, la variable a repris le contenu de la zone de texte d'entrée (`document.formulaire.entree.value`) auquel la méthode `toUpperCase()` est appliquée. Le résultat est renvoyé dans la zone de texte de sortie (`document.formulaire.sortie.value`).



8. replace()

La méthode replace(x,y) remplace les caractères x dans la chaîne de caractères par y.

Il est important de noter que le premier paramètre est défini entre deux barres obliques /.

```
str="Je programme en PHP";
str=str.replace(/PHP/,"JavaScript")
```

Le résultat est : "Je programme en JavaScript".

Il faut noter que la méthode replace(x,y) ne va remplacer que la première occurrence des caractères correspondant à x dans la chaîne de caractères.

Pour remplacer toutes les occurrences dans la chaîne de caractères, il faut adopter la notation suivante :

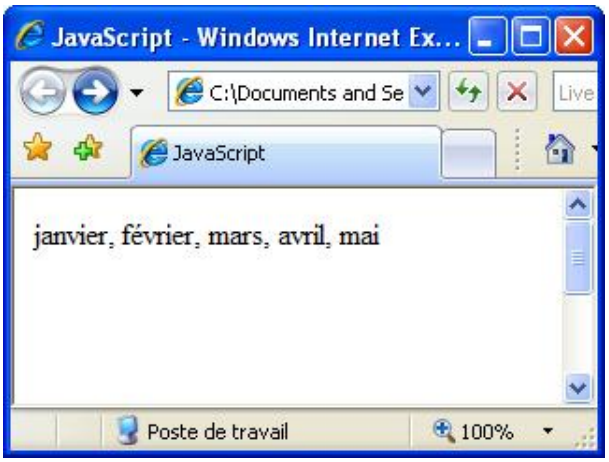
```
str=str.replace(/PHP/g,"JavaScript");
```

Exemple

JavaScript retourne une chaîne de caractères initiale "janvier,février, mars,avril".

Pour afficher une présentation plus conviviale avec les mêmes termes séparés par des espaces "janvier, février, mars, avril", il suffit de remplacer toutes les chaînes virgules par les chaînes virgule espace.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
chaine="janvier,février,mars,avril,mai"
chaine=chaine.replace(/,/g,", ");
document.write(chaine);
//]]>
</script>
</head>
<body>
</body>
</html>
```



9. Les expressions régulières

Les méthodes de manipulation de caractères de l'objet String (JavaScript 1.1) décrites plus haut se sont rapidement révélées trop limitées.

On a introduit alors en JavaScript les expressions régulières et l'objet RegExp (JavaScript 1.2) qui comportent des critères de recherche plus sophistiqués et qui, en outre, permettent de modifier de façon dynamique des chaînes de caractères pendant l'exécution du script.

Pour cela, il vous faut définir une instance de l'objet **RegExp**. C'est ce qu'on appellera une **exp**ression **rég**ulière. Les

éléments de cette expression régulière sont inclus dans des barres obliques. Les différents éléments sont séparés par une virgule.

a. Déclaration

La création d'un objet RegExp se crée à l'aide de la syntaxe suivante :

```
expression = /critère/;
```

Il est également possible de créer un tel objet de manière plus classique à l'aide de son constructeur :

```
expression = new RegExp("critère");
```

On peut de plus, préciser le comportement de l'expression régulière de façon optionnelle par :

- `g` qui indique une recherche globale de toutes les occurrences.
- `i` pour une recherche non sensible à la casse, soit indépendamment de la présence de majuscule ou minuscule dans la chaîne.
- `gi` qui combine les deux.

```
expression = /critère/g;
expression = new RegExp("critère","i");
```

b. Syntaxe

Les expressions régulières, issues du langage de programmation Perl, en reprennent la concision mais aussi la complexité...

Le tableau suivant présente une liste de composants pour former une expression régulière.

Caractère	Description
xyz	Trouve xyz n'importe où dans la chaîne de caractères.
^xyz	Trouve xyz au début de la chaîne de caractères.
xyz$	Trouve xyz à la fin de la chaîne de caractères.
xyz*	Trouve xy suivi de zéro ou plusieurs z.
xyz+	Trouve xy suivi de un ou plusieurs z.
xyz?	Trouve xy suivi de zéro ou un seul z.
.xyz	Trouve xyz précédé d'un caractère joker.
.+xyz	Trouve xyz précédé de plusieurs caractères joker.
\bxyz\b	Trouve xyz en tant que mot distinct.
\Bxyz\B	Trouve uniquement xyz à l'intérieur de mots.
x|y	Trouve x ou y.
[xyz]	Trouve x ou y ou z.
[a-z]	Trouve n'importe quel caractère de a à z (alphabet ASCII, soit sans caractères accentués).

[0-9]	Trouve n'importe quel chiffre de 0 à 9.
x{n}	Trouve x exactement n fois.
x{n,}	Trouve x au moins n fois.
x{n,m}	Trouve x entre n et m fois.

Pour rechercher un caractère faisant partie des caractères spéciaux, il suffit de le faire précéder d'une barre oblique inverse.

Caractère spécial	Notation
\	\\
.	\.
$	\$
[	\[
]	\]
(	\(
)	\)
{	\{
}	\}
^^	\^
?	\?
*	*
++	\+
-	\-

On note également :

Caractère	Description
\f	Trouve un signe de saut de page.
\n	Trouve un saut de ligne.
\r	Trouve un retour chariot.
\t	Trouve une tabulation horizontale.
\v	Trouve une tabulation verticale.
\s	Trouve toute sorte d'espace vide donc espace, retour chariot, tabulation, saut de ligne, saut de page.
\S	Trouve un caractère quelconque qui ne soit pas un espace vide.

\w	Trouve tous les caractères alphanumériques (minuscules ou majuscules) ainsi que le tiret de soulignement.
\W	Trouve tous les caractères non alphanumériques.
\d	Trouve tous les caractères numériques soit chiffre de 0 à 9.
\D	Trouve tous les caractères non numériques.

Exemple 1

Un nombre (entier ou décimal).

```
var reg = /^\d+[.]?\d*$/;
```

- le signe ^ signale que l'on commence au début de la chaîne de caractères.
- \d+ pour plusieurs caractères numériques.
- [.]? pour zéro ou une fois un point décimal.
- \d* pour zéro ou plusieurs caractères numériques.
- $ pour fin du nombre.

Exemple 2

Login valide de 3 à 8 caractères (numériques ou alphanumériques, majuscules ou minuscules) sans caractères spéciaux.

```
var exp=new RegExp("^[a-zA-Z0-9]{3,8}$");
```

- le signe ^ signale que l'on commence au début de la chaîne de caractères.
- [a-zA-Z0-9] note que les caractères alphanumériques peuvent être en minuscule de a à z, en majuscules de A à Z et des chiffres de 0 à 9.
- {3,8} vérifie si la chaîne comporte entre 3 et 8 caractères.
- $ signale la fin de la chaîne de caractères et interdit tout caractères supplémentaires.

c. Méthodes

Certaines méthodes de l'objet String peuvent utiliser les expressions régulières.

Méthode	Description
match()	Effectue une recherche de correspondance dans une chaîne. Elle renvoie un tableau de valeurs ou *null* en cas d'échec.
replace()	Effectue une recherche de correspondance dans une chaîne et remplace la sous-chaîne capturée par la chaîne de remplacement.
split()	Utilise une expression rationnelle ou une chaîne pour diviser une chaîne en un tableau de sous-chaînes.

Par ailleurs l'objet RegExp possède certaines méthodes qui lui sont propres.

Méthode	Description
exec()	Effectue une recherche de correspondance dans une chaîne. Elle renvoie un tableau de valeurs.
test()	Teste la correspondance d'une chaîne. Elle renvoie la valeur *true* ou *false*.
search()	Teste la correspondance d'une chaîne. Elle renvoie l'index de la capture, ou -1 en cas d'échec.

Exemple avec split()

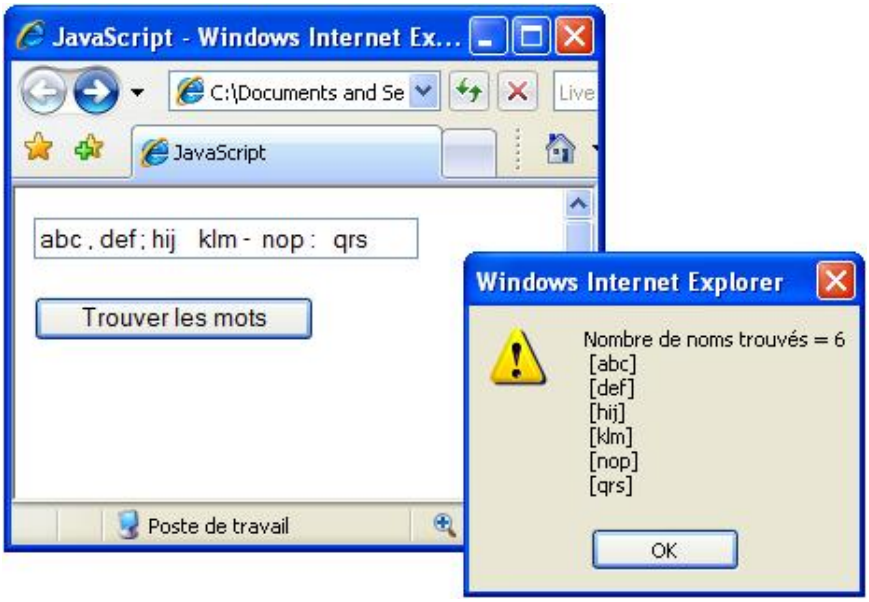
Soit une liste de noms dont les séparateurs ne sont pas uniformes : " abc ,def ; hij klm - nop". Isolons tous les noms de cette liste.

L'expression régulière va prévoir des séparateurs sous forme de virgule, point-virgule, double point, tiret, un ou des espaces. Celle-ci devient :

```
var expression=new RegExp("[,;:- ]+","g");
```

Le code complet devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function extraire(form) {
var test=document.form.exemple.value;
var expression = new RegExp("[,:;-]+","g");
var tableau = test.split(expression);
var affichage = "Nombre de noms trouvés = " + tableau.length + "\n";
for (var i=0;i<tableau.length;i++){
affichage=affichage + " ["+ tableau[i] + "]\n";
}
alert(affichage);
}
//]]>
</script>
</head>
<body>
<form name="form" action="">
<input type="text" name="exemple" size="30" value="abc ,def ; hij
klm - nop"/>
<p><input type="button" value="Trouver les mots"
onclick="extraire(form)" /></p>
</form>
</body>
</html>
```



Exemple avec match()

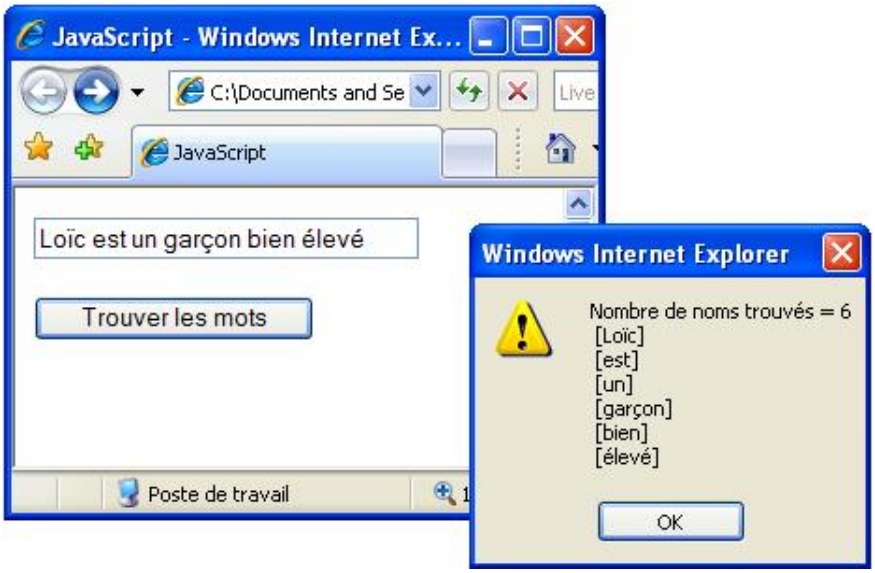
Continuons sur le même thème en utilisant la méthode `match()` pour isoler les mots d'une phrase.

L'expression régulière doit reconnaître tous les caractères alphanumériques, en majuscules ou minuscules, de la chaîne de caractères. Il est prudent de prévoir quelques caractères accentués.

```
var expression=new RegExp("[a-zA-Zéèàçûï\-]+","gi");
```

Le code complet devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function extraire(form) {
var test=document.form.exemple.value;
var expression=new RegExp("[a-zA-Zéèàçûï\-]+","gi");
var tableau=test.match(expression);
var affichage="Nombre de noms trouvés = " + tableau.length + "\n";
for (var i=0;i<tableau.length;i++){
affichage=affichage + " ["+ tableau[i] + "]\n";
}
alert(affichage);
}
//]]>
</script>
</head>
<body>
<form name="form" action="">
<input type="text" name="exemple" size="30" value="Loïc est un garçon
bien élevé"/>
<p><input type="button" value="Trouver les mots"
onclick="extraire(form)" /></p>
</form>
</body>
</html>
```



Exemple avec test()

Parmi les méthodes de l'objet RegExp, la méthode expression.test("chaîne") est assurément la plus utilisée. Elle teste une chaîne de caractères selon les critères de l'expression régulière. Cette méthode retourne *true* si la recherche est fructueuse et *false* dans le cas contraire.

Appliquons-la à la vérification de la validité d'une adresse email.

Une adresse e-mail comporte toujours le signe @ et un point pour le nom du serveur.

```
exp = new RegExp("@.");
```

C'est un peu simpliste. Entre l'arobase et le point, il doit y avoir quelques caractères (au moins 2). De plus, après le point, il doit avoir deux, trois ou quatre caractères pour le nom de domaine.

```
exp = new RegExp("@[a-z]{2,}[.][a-z]{2,4}$");
```

Le signe $ signifie la fin de l'e-mail.

Il y a bien entendu des caractères (ici en minuscules) avant l'arobase. La présence du point, du tiret ou du soulignement est également prévue.

```
exp = new RegExp("^[a-z0-9.-_]+@[a-z]{2,}[.][a-z]{2,4}$");
```

Le signe ^ fait démarrer le test depuis le début de la chaîne.

Certains ont encore des majuscules dans leur adresse.

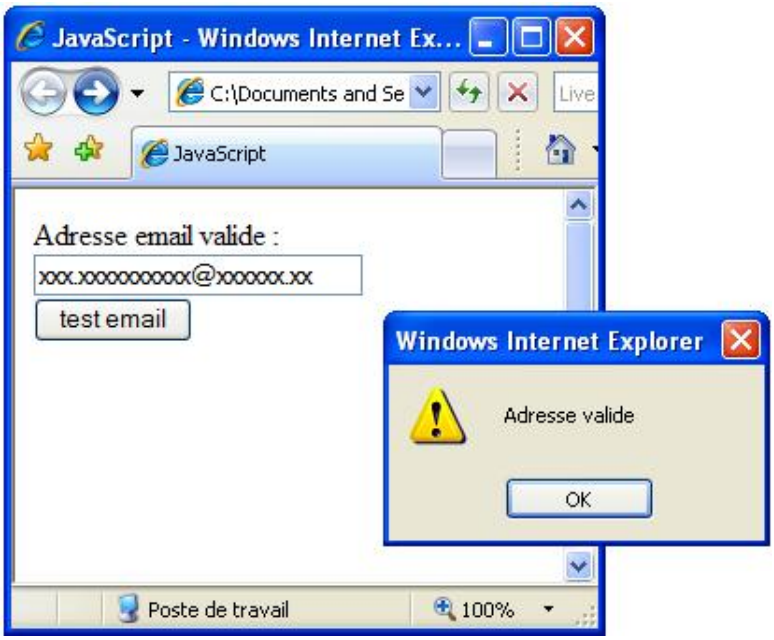
```
exp = new RegExp("^[a-z0-9.-_]+@[a-z]{2,}[.][a-z]{2,4}$","i");
```

L'équivalent de cette expression régulière en JavaScript "classique" aurait réclamé de nombreuses lignes de code.

Le code devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function trouver() {
var exp = new RegExp("^[a-z0-9.-_]+@[a-z]{2,}[.][a-z]{2,4}$","i");
var email = document.form.mail.value;
if (exp.test(email)) {
alert("Adresse valide");
}
```

```
else {
alert("Non valide");
document.form.mail.value="";
}
}
//]]>
</script>
</head>
<body>
<form action="" name="form">
Adresse email valide <br />
<input type="text" name="mail" size="25" /><br />
<input type="button" value="test email" onclick="trouver()" />
</form>
</body>
```



La vérification de la validité des adresses de courrier électronique peut également se réaliser avec l'expression régulière suivante, aussi efficace mais à priori assez hermétique.

```
RegExp("^\\w[\\w\-\_\.]*\\w@\\w[\\w\-\_\.]*\\w\\.\\w{2,4}$");
```

`^\\w`

un caractère au début de l'e-mail.

`[\\w\-\_\.]*`

ensuite une série de caractères avec éventuellement un tiret, un soulignement et un point.

`\\w`

un caractère seul devant l'arobase.

`@@`

le signe arobase.

`\\w`

un caractère seul.

`[\\w\-\_\.]*`

divers caractères avec éventuellement un tiret, un soulignement et un point.

`\\w`

un caractère pour finir cette série.

`\\.`

un point.

`\\w{2,4}`

de deux à quatre caractères pour le nom de domaine.

`$`

fin de la chaîne.

Une expression régulière peut même être prévue pour s'assurer qu'il n'y a pas de signes interdits dans l'adresse.

```
var illegal = new RegExp("[\(\),;: àéè\']+","g");
```

On interdit ici les caractères les parenthèses ouvrante et fermantes, la virgule, le point-virgule, le double point, l'espace des caractères accentués et l'apostrophe.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function trouver() {
var illegal = new RegExp("[\(\),;: àéè\']+","g");
var exp = new RegExp("^\\w[\\w\-\_\.]*\\w@\\w[\\w\-\_\.]*\\w\\.\\w{2,4}
$");
var email = document.form.mail.value;
if (illegal.test(email)==false) {
if (exp.test(email)==true) {
alert("Adresse valide");
}
}
else {
alert("Recommencez");
document.form.mail.value="";
}
}
//]]>
</script>
</head>
<body>
<form action="" name="form">
Adresse email valide <br />
<input type="text" name="mail" size="25" /><br />
<input type="button" value="test email" onclick="trouver()" />
</form>
</body>
</html>
```

Tableaux en JavaScript (objet Array)

L'objet `Array` permet de lister une série de valeurs dans une seule variable. Les différents éléments sont identifiés par un nombre entier représentant leurs positions. Ce nombre est appelé un index. L'objet Array permet ainsi de créer des variables dites indicées ou indexées.

L'index commence (comme souvent en JavaScript) à la valeur 0.

Exemple

`semaine`	
0	Lundi
1	Mardi
2	Mercredi
3	Jeudi
4	Vendredi
5	Samedi
6	Dimanche

La variable indexée `semaine` contient comme valeurs, les différents jours de la `semaine`.

Ces variables indexées peuvent être identifiées en JavaScript (comme dans d'autres langages de programmation) sous le vocable de tableau.

Les tableaux en JavaScript n'ont rien en commun avec les tableaux du langage Html ou Xhtml. À ne pas confondre !

---

1. La définition d'un tableau

La création d'un objet de type Array s'opère à l'aide de l'opérateur new suivi de Array().

```
var tableau = new Array();
```

Commentaires :

- en JavaScript, il faut d'abord définir le tableau avant de pouvoir en préciser son contenu.
- il n'est pas obligatoire de définir la taille du tableau dans sa déclaration (par exemple `var tableau = new Array (7)`. La taille des tableaux est prise en charge de façon dynamique par JavaScript.
- il est possible, à tout moment, de connaître la taille d'un tableau, en invoquant la propriété *length* de l'objet correspondant à ce tableau.

Notre tableau avec les jours de la semaine, peut se définir comme suit :

```
var semaine = new Array();
```

2. L'initialisation d'un tableau

Pour initialiser le tableau, la syntaxe suivante est utilisée :

`tableau[i] = elément;` où i est un nombre compris entre 0 et la longueur du tableau -1.

Soit pour le tableau semaine :

```
var semaine = new Array();
semaine[0] = "Lundi";
semaine[1] = "Mardi";
semaine[2] = "Mercredi";
semaine[3] = "Jeudi";
semaine[4] = "Vendredi";
semaine[5] = "Samedi";
semaine[6] = "Dimanche";
```

La définition et l'initialisation peuvent se faire lors de la même instruction sous la forme :

```
var semaine = new Array("Lundi" , "Mardi" , "Mercredi" , "Jeudi" ,
"Vendredi" , "Samedi" , "Dimanche");
```

3. L'accès aux données du tableau

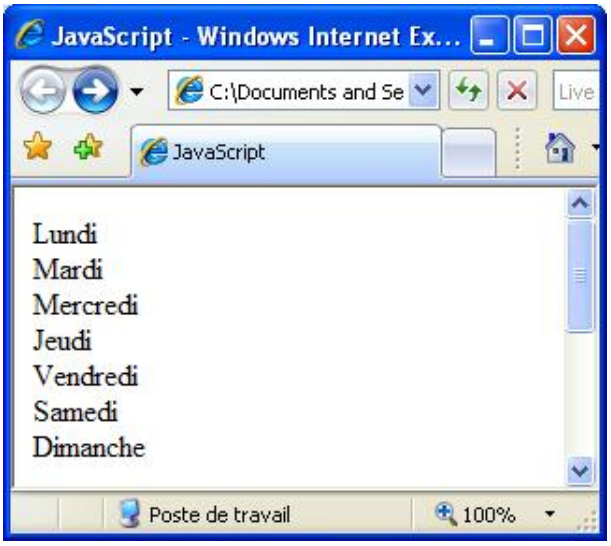
On accède à un élément du tableau, par le nom du tableau et le numéro de l'indice.

Soit, `semaine[0]` pour "Lundi".

Exemple

Afficher les éléments du tableau semaine.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
var semaine = new Array();
semaine[0] = "Lundi";
semaine[1] = "Mardi";
semaine[2] = "Mercredi";
semaine[3] = "Jeudi";
semaine[4] = "Vendredi";
semaine[5] = "Samedi";
semaine[6] = "Dimanche";
for (i=0; i<7; i++) {
document.write(semaine[i] + "<br />");
}
//]]>
</script>
</body>
</html>
```



L'affichage des jours de la semaine est réalisé par une boucle for (cf ce chapitre - Conditions et boucles - La boucle for) avec i initialisé à 0 et comme condition i inférieur à 7.

Lorsque la longueur du tableau est inconnue, on peut utiliser la propriété length. Ainsi, l'instruction for de l'exemple deviendrait : `for (i=0; i<semaine.length; i++);`

---

4. Les tableaux associatifs

Les tableaux associatifs sont des tableaux dont l'indice est une chaîne de caractères (et non plus un nombre). La chaîne est considérée comme l'index pour l'accès aux éléments du tableau.

```
var moteur = new Array("Google","Yahoo","Voila");
moteur["Google"] = "http://www.google.fr";
moteur["Yahoo"]="http://www.yahoo.fr";
moteur["Voila"]="http://www.voila.fr";
```

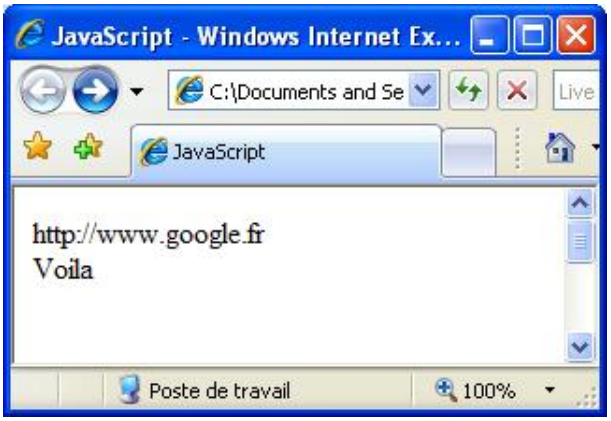
Notez la présence des guillemets entre les crochets de rigueur pour utiliser les chaînes de caractères.

Pour accéder aux données d'un tableau associatif :

```
moteur["Google"] ou moteur[1];
```

Exemple

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
var moteur = new Array("Google","Yahoo","Voila");
moteur["Google"] = "http://www.google.fr";
moteur["Yahoo"]="http://www.yahoo.fr";
moteur["Voila"]="http://www.voila.fr";
document.write(moteur["Google"] + "<br />");
document.write(moteur[2]);
//]]>
</script>
</body>
</html>
```



5. Les méthodes spécifiques aux tableaux

L'objet `Array()` présente de nombreuses méthodes. En voici quelques-unes.

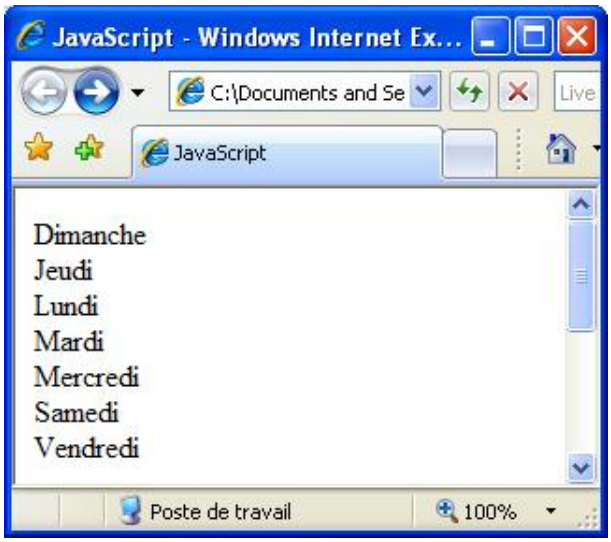
Elément	Description
`concat()`	Assemble deux ou plusieurs tableaux pour n'en faire qu'un seul. Cette méthode réclame en argument le tableau (ou plusieurs) qui est rajouté à la fin du tableau référencé.
`join()`	Regroupe tous les éléments du tableau dans une seule chaîne de caractères. Les différents éléments sont séparés par un caractère séparateur spécifié en argument. Par défaut, ce séparateur est une virgule.
`pop()`	Supprime le dernier élément du tableau et retourne sa valeur.
`reverse()`	Inverse l'ordre des éléments (mais ne les trie pas).
`shift()`	Supprime le dernier élément du tableau et retourne sa valeur.
`slice()`	Crée un nouveau tableau à partir d'une partie d'un tableau existant. S'écrit slice[début,fin].
`sort()`	Trie les éléments par ordre alphabétique (à condition qu'ils soient de même nature).

a. Tri alphabétique d'un tableau

On applique la méthode sort() au tableau *semaine*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
var semaine = new Array();
```

```
semaine[0] = "Lundi";
semaine[1] = "Mardi";
semaine[2] = "Mercredi";
semaine[3] = "Jeudi";
semaine[4] = "Vendredi";
semaine[5] = "Samedi";
semaine[6] = "Dimanche";
semaine.sort();
for (i=0; i<7; i++) {
document.write(semaine[i] + "<br />");
}
//]]>
</script>
</body>
</html>
```



b. Assemblage de tableaux

Soit deux tableaux : `semestre1` et `semestre2`. Ces deux tableaux sont regroupés dans un nouveau tableau `annee` par la méthode concat()<$I[]>.

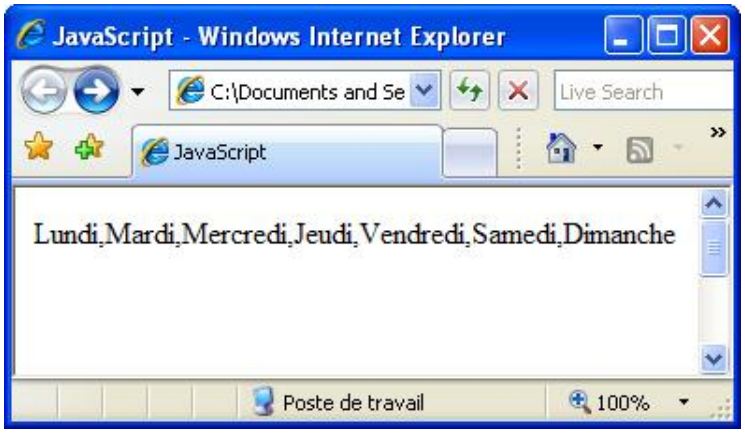
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
var semestre1 = new Array();
semestre1[0] = "Janvier";
semestre1[1] = "Février";
semestre1[2] = "Mars";
semestre1[3] = "Avril";
semestre1[4] = "Mai";
semestre1[5] = "Juin";
var semestre2 = new Array();
semestre2[0] = "Juillet";
semestre2[1] = "Août";
semestre2[2] = "Septembre";
semestre2[3] = "Octobre";
semestre2[4] = "Novembre";
semestre2[5] = "Décembre";
var annee= new Array();
```

```
annee = semestre1.concat(semestre2);
for (i=0; i<annee.length; i++){
document.write(annee[i] + " - ");
}
//]]>
</script>
</body>
</html>
```

c. Regrouper en une chaîne

Avec la méthode join(), regroupons le tableau `semaine` dans une chaîne de caractères.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
var semaine = new Array();
semaine[0] = "Lundi";
semaine[1] = "Mardi";
semaine[2] = "Mercredi";
semaine[3] = "Jeudi";
semaine[4] = "Vendredi";
semaine[5] = "Samedi";
semaine[6] = "Dimanche";
var join = semaine.join();
document.write(join);
//]]>
</script>
</body>
</html>
```

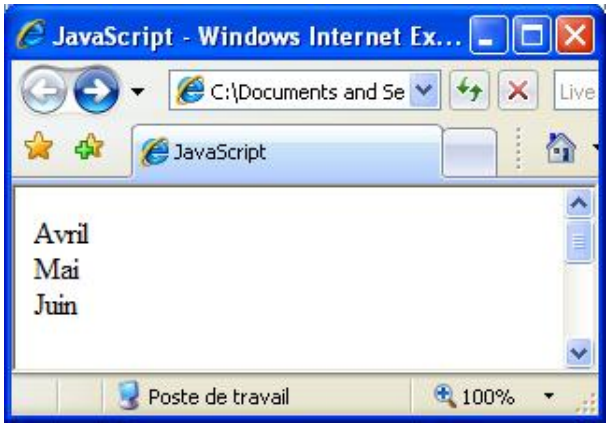


Par défaut, le séparateur est une virgule. Un autre séparateur peut être spécifié en argument de la méthode join(). Soit par exemple, join("-") pour un tiret de séparation.

d. Partie d'un tableau

Utilisation de la méthode slice() pour créer un nouveau tableau comportant les mois du printemps.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
var annee = new Array();
annee[0] = "Janvier";
annee[1] = "Février";
annee[2] = "Mars";
annee[3] = "Avril";
annee[4] = "Mai";
annee[5] = "Juin";
annee[6] = "Juillet";
annee[7] = "Août";
annee[8] = "Septembre";
annee[9] = "Octobre";
annee[10] = "Novembre";
annee[11] = "Décembre";
var printemps = new Array();
printemps = annee.slice(3,6);
for (i=0; i<printemps.length; i++){
document.write(printemps[i] + "<br /> ");
}
//]]>
</script>
</body>
</html>
```



Quelques autres objets JavaScript

Nous n'avons repris jusqu'à présent que les éléments essentiels du JavaScript. Il existe néanmoins d'autres objets, qu'il est difficile de ne pas présenter mais dont l'étude est moins détaillée.

1. L'objet Date

L'objet Date permet de gérer les informations relatives au temps, soit la date et l'heure.

a. new Date();

L'objet Date en JavaScript présente une particularité dans la mesure où il faut d'abord le créer, avant de pouvoir l'utiliser.

En effet, avant de pouvoir appliquer les multiples fonctions dédiées à la gestion du temps, il faut créer une instance de la date et l'heure de l'ordinateur de l'utilisateur.

Cette instance est créée par new Date().

```
variable=new Date();
```

Ainsi `variable` sera une chaîne de caractères au format :

```
Fri Feb 23 10:42:23 UTC+0100 2007
```

Soit vendredi (`Fri` pour `Friday`) février (`Feb` pour `February`) `23` pour le 23ème jour du mois 10:42:23 pour 10 heures 42 minutes et 23 secondes et 2007 pour l'année. Cette heure a été notée à l'heure UTC (temps universel coordonné) + 1 heure.

Commentaires :

- la date et l'heure en JavaScript commencent au 1er janvier 1970. Toute référence à une date antérieure donne un résultat aléatoire.
- l'unité pour le calcul interne du temps est le millième de seconde.

b. Méthodes

```
getYear()
```

Retourne en principe les deux derniers chiffres de l'année dans l'objet Date. Microsoft Internet Explorer retourne cependant les quatre chiffres soit ici 2007. Des problèmes de compatibilité sont donc possibles.

```
objetdate=new Date();

annee=objetdate.getYear();
```

```
getFullYear()
```

Retourne les quatre chiffres de l'année. La compatibilité est totale.

```
objetdate=new Date();

annee=objetdate.getFullYear();
```

```
getMonth()
```

Retourne le mois sous forme d'un entier compris entre 0 et 11 (0 pour janvier, 1 pour février, 2 pour mars, etc.).

```
objetdate=new Date();

mois=objetdate.getMonth();
```

```
getDate();
```

Retourne le jour du mois sous forme d'un entier compris entre 1 et 31.

```
objetdate=new Date();
```

```
jour=objetdate.getDate();
```

```
getDay();
```

Retourne le jour de la semaine sous forme d'un entier compris entre 0 et 6 (0 pour dimanche, 1 pour lundi, 2 pour mardi, etc.).

```
objetdate=new Date();
semaine= objetdate.getDay();
```

```
getHours();
```

Retourne l'heure sous forme d'un entier compris entre 0 et 23.

```
objetdate=new Date();
heure=objetdate.getHours();
```

```
getMinutes();
```

Retourne les minutes sous forme d'un entier compris entre 0 et 59.

```
objetdate=new Date();
minute=objetdate.getMinutes();
```

```
getSeconds();
```

Retourne les secondes sous forme d'un entier compris entre 0 et 59.

```
objetdate=new Date();
seconde=objetdate.getSeconds();
```

Il existe de nombreuses autres méthodes mais elles n'entrent pas dans le cas de cet ouvrage et de notre étude de JavaScript.

c. Exemple

Afficher la date et l'heure de façon usuelle en français.

Remplacez le numéro du mois retourné avec la méthode getMonth() par son nom (soit 4 par avril).

Pour ce faire, nous allons créer un tableau indicé (*Array*) en prenant soin de commencer à l'indice 0 pour le mois de janvier.

```
var tableau_mois = Array();
tableau_mois[0] = "janvier";
tableau_mois[1] = "février";
tableau_mois[2] = "mars";
tableau_mois[3] = "avril";
tableau_mois[4] = "mai";
tableau_mois[5] = "juin";
tableau_mois[6] = "juillet";
tableau_mois[7] = "août";
tableau_mois[8] = "septembre";
tableau_mois[9] = "octobre";
tableau_mois[10] = "novembre";
tableau_mois[11] = "décembre";
```

Pour afficher le jour de la semaine en toutes lettres, nous procédons de la même façon par un tableau.

```
var tableau_jours = Array();
tableau_jours[0] = "dimanche";
tableau_jours[1] = "lundi";
tableau_jours[2] = "mardi";
tableau_jours[3] = "mercredi";
tableau_jours[4] = "jeudi";
tableau_jours[5] = "vendredi";
tableau_jours[6] = "samedi";
```

On souhaite harmoniser l'affichage des heures, minutes et secondes avec deux chiffres. Ainsi au lieu d'afficher 7 h, ce sera 07 h.

Cette manipulation peut se réaliser avec un test conditionnel. Si le chiffre est inférieur à 10, on ajoutera un 0. Sinon, on garde le chiffre.

Ce test peut s'écrire :

```
if (if (min<10) {
```

```
min="0" + min;
}
else {
min="" + min;
}
```

Ce qui peut se noter de façon abrégée (voir dans ce chapitre - Conditions et boucles - Les conditions if...else) :

```
min = ((min<10) ? "0" : "") + min;
```

Le script devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
var objetdate= new Date();
var mois=objetdate.getMonth();
var tableau_mois = Array();
tableau_mois[0] = "janvier";
tableau_mois[1] = "février";
tableau_mois[2] = "mars";
tableau_mois[3] = "avril";
tableau_mois[4] = "mai";
tableau_mois[5] = "juin";
tableau_mois[6] = "juillet";
tableau_mois[7] = "août";
tableau_mois[8] = "septembre";
tableau_mois[9] = "octobre";
tableau_mois[10] = "novembre";
tableau_mois[11] = "décembre";
var nommois=tableau_mois[mois];
var jour=objetdate.getDay();
var tableau_jours = Array();
tableau_jours[0] = "dimanche";
tableau_jours[1] = "lundi";
tableau_jours[2] = "mardi";
tableau_jours[3] = "mercredi";
tableau_jours[4] = "jeudi";
tableau_jours[5] = "vendredi";
tableau_jours[6] = "samedi";
var nomjour=tableau_jours[jour];
document.write("Nous sommes le ");
document.write(nomjour + " ");
document.write(objetdate.getDate());
document.write(" ");
document.write(nommois);
document.write(" ");
document.write(objetdate.getFullYear());
document.write(" ");
heure= objetdate.getHours();
heure = ((heure<10) ? "0" : "") + heure;
document.write(heure + " heure ");
min=objetdate.getMinutes();
min = ((min<10) ? "0" : "") + min;
document.write(min + " minutes ");
sec=objetdate.getSeconds();
sec = ((sec<10) ? "0" : "") + sec;
document.write(sec + " secondes ");
//]]>
</script>
</body>
</html>
```

2. L'objet Math

AJAX se focalise surtout sur la manipulation des chaînes de caractères mais il n'est pas exclu qu'il faille effectuer des opérations sur des chiffres. L'objet Math met à la disposition du programmeur JavaScript une panoplie de méthodes permettant d'effectuer des calculs complexes.

La syntaxe est :

```
x = Math.méthode(paramètre);
```

Les méthodes les plus courantes de l'objet Math sont :

`abs(y)`

La méthode abs(y) renvoie la valeur absolue (valeur positive) de y. Elle supprime, en quelque sorte, le signe négatif d'un nombre.

```
Y = -4;
x = math.abs(y); a comme résultat 4.
```

`ceil(y)`

La méthode ceil(y) renvoie l'entier supérieur ou égal à y.

Attention ! Cette fonction n'arrondit pas le nombre. Comme l'illustre l'exemple suivant, si y = 1.01, la valeur de x vaut 2.

```
y=1.01;
x=Math.ceil(y); a comme résultat 2.
```

`floor(y)`

La méthode floor(y) renvoie l'entier inférieur ou égal à y.

Attention ! Cette fonction n'arrondit pas le nombre. Si y = 1.99, la valeur de x vaut 1.

```
y=1.99;
x=Math.floor(y); a comme résultat 1.
```

`round(y)`

La méthode round(y) arrondit le nombre à l'entier le plus proche.

```
y=20.355;
x=Math.round(y); a comme résultat 20.
```

Attention ! Certains calculs réclament une plus grande précision. Ainsi, pour avoir deux décimales après la virgule, la formule suivante est utilisée :

`x=Math.round(y*100))/100;` et dans ce cas, x vaut 20.36.

`max(y,z)`

La méthode max(y,z) renvoie le plus grand des deux nombres y et z.

```
y=20; z=10;
x=Math.max(y,z); a comme résultat 20.
```

```
min(y,z)
```

La méthode min(y,z) renvoie le plus petit des 2 nombres y et z.

```
y=20; z=10;
x=Math.min(y,z); a comme résultat x=10.
```

```
pow(y,z)
```

La méthode pow() calcule la valeur d'un nombre y à la puissance z.

```
y=2; z=8
x=Math.pow(y,z); a comme résultat 2^8 soit 256.
```

```
random()
```

La méthode random() renvoie la valeur d'un nombre aléatoire choisi entre 0 et 1.

Ce nombre aléatoire pourrait être 0.042105102268759464.

```
sqrt(y)
```

La méthode sqrt(y) renvoie la racine carrée de y.

```
y=25;
x=Math.sqrt(y); a comme résultat 5.
```

```
parseFloat(var)
```

Cette fonction convertit une chaîne en un nombre à virgule flottante. Particulièrement utile pour transformer le contenu d'une zone de texte (qui est considéré comme une chaîne de caractères) en un nombre en vue d'un traitement de calcul.

```
str='-.12345';
x=parseFloat(str); aura comme résultat le nombre -.12345.
```

Attention ! Le résultat risque d'être surprenant si JavaScript rencontre autre chose dans la chaîne que des chiffres, les signes + et -, le point décimal ou un exposant. S'il trouve un caractère "étranger", la fonction ne prend en compte que les caractères avant le caractère "étranger".

Si le premier caractère n'est pas un caractère admis, x est égal à 0 ou à NaN (*Not a Number*).

```
str=' €  5.50';
x=parseFloat(str); a comme résultat NaN.
```

```
parseInt(var)
```

Retourne la partie entière d'un nombre avec une virgule.

```
str='1.2345';
x=parseInt(str); a comme résultat 1.
```

```
eval(var)
```

Cette fonction évalue une chaîne de caractères sous forme de valeur numérique. Dans la chaîne peuvent être stockées des opérations numériques, des opérations de comparaison, des instructions et même des fonctions.

```
Str='5 + 10';
x=eval(str); a comme résultat 15.
```

Exemple

Élaborons un convertisseur FF vers l'euro.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
```

```
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
var montant_francs;
var montant_euro;
function convertir() {
montant_francs = document.form.entree.value;
montant_francs=montant_francs.replace(/,/,".");
montant=parseFloat(montant_francs);
montant_euro=montant/6.55957;
montant_euro=(Math.round(montant_euro*100))/100;
document.form.sortie.value=montant_euro;
if (isNaN(montant_francs))
{
alert("Entrez un nombre");
document.form.entree.value = "";
document.form.sortie.value = "";
}
}
//]]>
</script>
</head>
<body>
<form name="form" action="">
<input type="text" size="10" name="entree" /> FF
<input type="button" value="Convertir" onclick="convertir()" />
<input type="text" size="10" name="sortie" /> _
</form>
</body>
</html>
```

Pour commencer, les variables `montant_francs` et `montant_euro` sont définies.

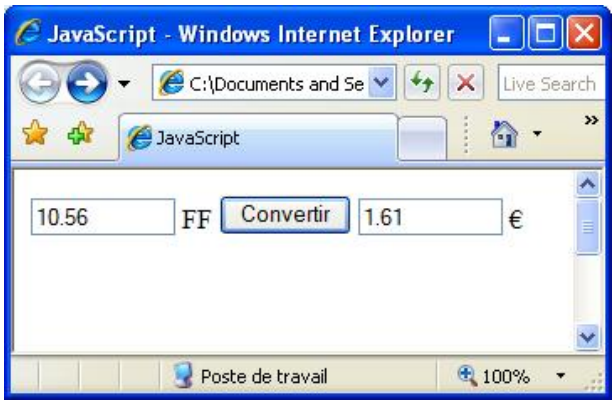
Dans `montant_francs`, la valeur de la zone de texte d'entrée est récupérée soit la valeur de `document.form.entree.value`.

Le visiteur risque d'encoder le montant avec une virgule. La fonction replace() est utilisée pour convertir l'éventuelle virgule en point (voir ce chapitre Manipulation des chaînes des caractères - replace()).

Pour transformer en nombre la chaîne de caractères contenue dans la variable `formule`, la méthode parseFloat() présentée ci-avant est utilisée.

Ce nombre est divisé par 6.55957, puis est arrondit (`Math.round()`).

Enfin si l'utilisateur n'a pas encodé un nombre (NaN), une boîte d'alerte apparaît et les zones de texte sont réinitialisées.



3. L'objet navigator

L'objet navigator est particulièrement utile pour adapter les modalités d'exécution des scripts lors des divergences d'implémentation de certaines instructions JavaScript selon les navigateurs ou selon les versions d'un même navigateur.

a. Propriétés

L'objet navigator de JavaScript fournit un ensemble d'informations sur l'environnement de travail du visiteur et tout particulièrement sur le navigateur qu'il utilise.

Passons en revue quelques-unes de ses propriétés et méthodes.

Propriété	Description
	Retourne le "surnom" (*CodeName*) du navigateur.

`appCodeName`	Historiquement, cette propriété a été inventée par Netscape : en effet le navigateur Netscape portait (déjà) le surnom de "Mozilla".
`appName`	Retourne le nom du navigateur.
`appVersion`	Retourne la version du navigateur.
`language`	Spécifie la langue utilisée par le navigateur. Cette propriété n'existe pas sous Internet Explorer.
`userLanguage`	Spécifie la langue utilisée par le navigateur. Cette propriété n'est valable que sous Internet Explorer.
`platform`	Retourne le système d'exploitation du navigateur de l'internaute.
`userAgent`	Spécifie des informations relatives au navigateur (*user agent* signifie navigateur).

Appliquées à la machine de l'auteur, ces propriétés fournissent les informations suivantes :

navigator.appCodeName

Internet Explorer 6 et 7	Mozilla (étonnant non ?)
Firefox	Mozilla

Comme Internet Explorer, Netscape, Mozilla, Opera ou Firefox renvoient la même valeur, cette propriété n'est pas très utile dans la pratique.

navigator.appName

Internet Explorer 6 et 7	Microsoft Internet Explorer
Firefox	Netscape

navigator.appVersion

Internet Explorer 7	4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30)
Internet Explorer 6	4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30)
Firefox	5.0 (Windows; fr-FR)

Les informations, autant pour Internet Explorer que pour Firefox sont assez neutres. Seul la mention MSIE (**M**icro**S**oft **I**nternet **E**xplorer) permet vraiment d'identifier Internet Explorer.

navigator.language

Internet Explorer 6 et 7	undefined
Firefox	fr-FR

navigator.userLanguage

Internet Explorer 6 et 7	fr
Firefox	undefined

navigator.platform

Internet Explorer 6 et 7	Win32
Firefox	Win32

navigator.userAgent

Internet Explorer 7	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30)
Internet Explorer 6	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30)

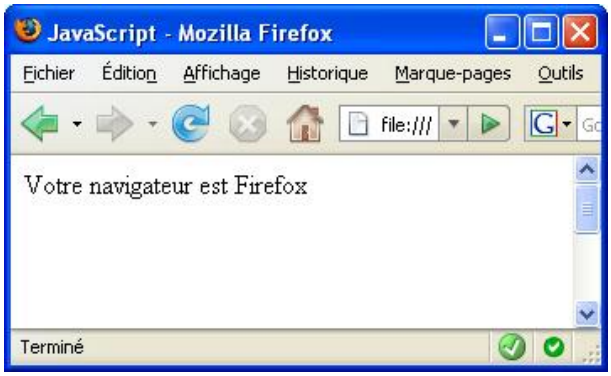
Firefox 2	Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1

b. Distinguer Internet Explorer de Firefox

La propriété *userAgent* permet d'identifier le nom du navigateur. Grâce à la méthode indexOf (ce chapitre Manipulation des chaînes de caractères - indexOf()), la présence de la chaîne partielle "MSIE" ou "Firefox" va être vérifiée.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
if (navigator.userAgent.indexOf("MSIE") != -1) {
navigateur = "Internet Explorer";
}
else {
if (navigator.userAgent.indexOf("Firefox") != -1) {
navigateur = "Firefox";
}
else {
navigateur ="Autre";
}
}
document.write("Votre navigateur est " + navigateur);
//]]>
</script>
</body>
</html>
```

Voici le résultat sous Firefox :



c. Identifier les versions d'Internet Explorer

Nous utilisons les informations transmises par `navigator.userAgent`.

Internet Explorer 7	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET)
Internet Explorer 6	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Internet Explorer 5.5	Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.1; SV1)
Internet Explorer 5	Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0; SV1)

Il est évident que nous devons rechercher la présence des chaînes partielles MSIE 7, MSIE 6, MSIE 5.5 et MSIE 5.0 afin d'identifier les versions d'Internet Explorer.

Le script devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
if (navigator.userAgent.indexOf("MSIE 7") != -1) {
document.write("<h2> Internet Explorer 7</h2>");
}
if (navigator.userAgent.indexOf("MSIE 6") != -1) {
document.write("<h2> Internet Explorer 6</h2>");
}
if (navigator.userAgent.indexOf("MSIE 5.5") != -1) {
document.write("<h2> Internet Explorer 5.5</h2>");
}
if (navigator.userAgent.indexOf("MSIE 5.0") != -1) {
document.write("<h2> Internet Explorer 5</h2>");
}
//]]>
</script>
</body>
</html>
```

Et le résultat sous Internet Explorer 6 :

4. L'objet window

Certaines méthodes de l'objet window ne vous sont pas inconnues. Ainsi les méthodes alert(), confirm(), prompt() et setTimeout ont déjà été abordées dans ce chapitre - Fonctions et méthodes.

a. La fenêtre popup

Les méthodes open() et close() de l'objet window permettent d'ouvrir (*open*) ou de fermer (*close*) une nouvelle fenêtre du navigateur. Il sera en outre possible de définir de nombreuses caractéristiques de cette nouvelle fenêtre.

L'usage des nouvelles fenêtres, appelées aussi fenêtres pop-up, est souvent utile pour afficher des informations complémentaires sans surcharger la page (ou la fenêtre) de départ. Elles comportent cependant deux désagréments notoires. En effet, si le concepteur n'a pas prévu un moyen de fermer celles-ci, un nombre important de fenêtres pop-up peut apparaître. En outre, il faut admettre que la toile est "polluée" par ces fenêtres pop-up, le plus souvent utilisées à des fins publicitaires. Nous vous conseillons de bien fermer les fenêtres pop-up et à les utiliser avec modération ou discernement.

La syntaxe de open() est :

```
window.open("URL","nom_fenêtre","caractéristiques_fenêtre");
```

Commentaires :

- la mention de `window` est facultative car window étant l'objet avec la plus haute priorité, celui-ci est repris par défaut par le JavaScript.
- `URL` est l'adresse de la page que l'on désire afficher dans la nouvelle fenêtre. Si on ne désire pas afficher un fichier htm existant, on indiquera simplement " ".

- `nom_fenêtre` désigne le nom de la nouvelle fenêtre : cette variable peut être reprise en référence dans le code JavaScript.
- `caractérstiques_fenêtre` est une liste (voir ci-après) de certaines ou de toutes les caractéristiques de la fenêtre. Celles-ci se notent à la suite l'une de l'autre, séparées par des virgules, sans retour à la ligne. Ces caractéristiques doivent être impérativement écrites sur une seule et même ligne.

Pour le moteur JavaScript, les caractéristiques de la fenêtre doivent être impérativement écrites sur une seule et même ligne.

Ces caractéristiques sont :

`width=x`

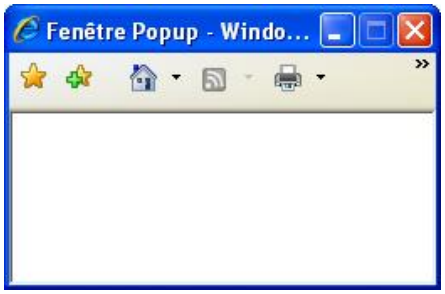
Largeur de la nouvelle fenêtre en pixels.

`height=x`

Hauteur de la nouvelle fenêtre en pixels. La valeur minimale est de 100 pixels.

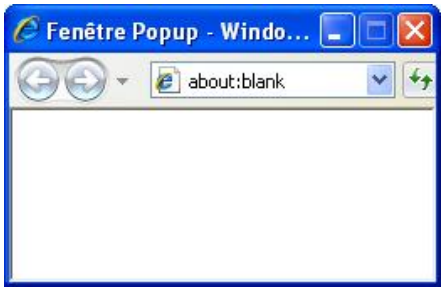
`toolbar=yes ou no`

Affichage de la barre d'outils. La valeur par défaut est yes



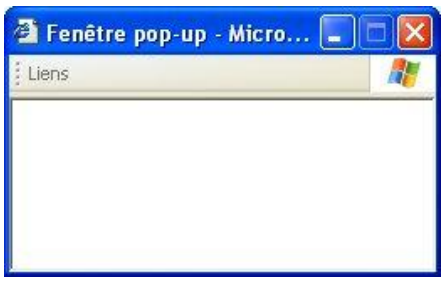
`location=yes ou no`

Affichage de la barre d'adresse.



`directories=yes ou no`

Affichage des boutons d'accès rapide (barre de liens). Pas d'affichage sous IE 7.



`status=yes ou no`

Affichage de la barre d'état.

`menubar=yes ou no`

Affichage de la barre de menus.

`scrollbars=yes ou no`

Affichage des barres de défilement.

`resizable=yes ou no`

Redimensionnement de la fenêtre.

`fullscreen=yes ou no`

Affichage de la nouvelle fenêtre en plein écran. Internet Explorer uniquement.

`top=x`

Position en pixels par rapport au bord supérieur de l'écran.

`left=x`

Position en pixels par rapport au bord gauche de l'écran.

La notation 1 ou 0 à la place de yes ou no est également possible.

La syntaxe de close() est la suivante :

`window.close("nom_de_la_fenêtre")`où window est facultatif.

Dans la pratique, pour éviter toute confusion possible, le raccourci *self* est utilisé pour indiquer la fenêtre en cours.

```
window.self.close();
```

La méthode close() entre dans le cadre du concept de sécurité de JavaScript. Cela signifie qu'une fenêtre, à partir du moment où elle possède un historique (parce que l'utilisateur a déjà appelé plusieurs pages), ne se laisse plus fermer sans demande de confirmation de la part du navigateur. Il n'est pas possible d'empêcher cette demande.

Exemple 1

Plutôt que de charger un fichier htm, il est possible d'écrire en JavaScript dans la nouvelle fenêtre.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function popup() {
nouvelle = open(’’, ’’ , ’width=200, height=100, toolbar=no, location=no,
directories=no, status=no, menubar=no, scrollbars=no, resizable=no’);
nouvelle.document.write(’<html><head><title>Popup 1</title></head>
<body bgcolor=\"#99ccff\"><h2>JavaScript</h2></body></html>’);
}
//]]>
</script>
</head>
<body>
<form action="">
<input type ="button" value="Ouvrir la fenêtre" onclick="popup()" />
</form>
</body>
</html>
```

Pour rappel, les caractéristiques de la nouvelle fenêtre et le code de l'instruction `write` doivent être impérativement écrits sur une seule et même ligne dans le code.

Exemple 2

Fermer une nouvelle fenêtre avec un bouton.

La méthode close() requiert le nom de la fenêtre comme argument. Pour éviter toute confusion possible, on utilise self pour indiquer la fenêtre en cours.

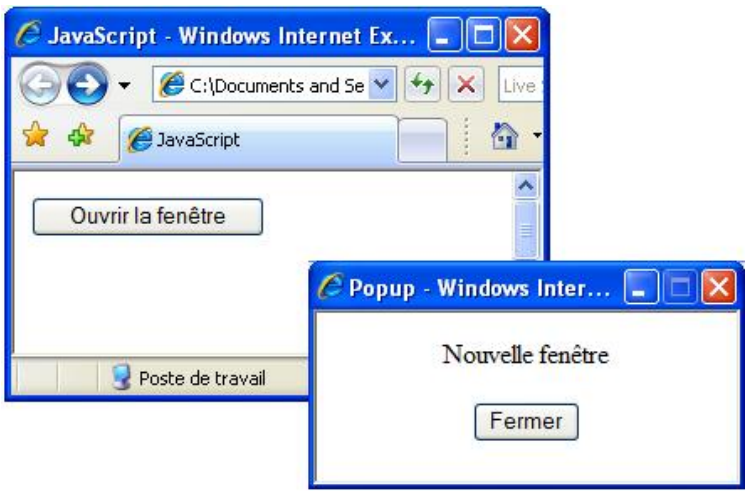
Voici le fichier qui lance la fenêtre popup :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function popup() {
open('popup.htm','nouvelle', 'width=200, height=100, toolbar=no, locati
on=no, directories=no, status=no, menubar=no, scrollbars=no, resizable=
no');
}
//]]>
</script>
</head>
<body>
<form action="">
<input type ="button" value="Ouvrir la fenêtre" onclick="popup()" />
</form>
</body>
</html>
```

Pour rappel, les caractéristiques de la nouvelle fenêtre doivent être impérativement écrites sur une seule et même ligne dans le code.

Le fichier popup.htm de la nouvelle fenêtre :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Popup</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body style="text-align: center;">
Nouvelle fenêtre
<form  action="">
<input type="button" value= "Fermer" onclick="self.close()" />
</form>
</body>
</html>
```



b. La zone utile du navigateur

La zone disponible pour la page dans la fenêtre du navigateur peut être déterminée par les propriétés innerWidth et innerHeight de l'objet window pour Firefox et autres navigateurs de la famille Mozilla.

`innerHeight`

Hauteur utile de la fenêtre (hors cadre et barre de défilement). En pixels.

`innerWidth`

Largeur utile de la fenêtre (hors cadre et barre de défilement). En pixels.

Internet Explorer ne reconnaît pas ces propriétés mais ces dimensions sont disponibles par les propriétés *body.clientHeight* et *body.clientWidth* de l'objet document.

Voici un script compatible pour les deux versions du navigateur :

```
<html>
<head>
<title>JavaScript</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<script type="text/JavaScript">
//<![CDATA[
var largeur;
var hauteur;
if (window.innerWidth) {
largeur = window.innerWidth;
}
else {
largeur = document.body.clientWidth;
}
if (window.innerHeight) {
hauteur = window.innerHeight;
}
else {
hauteur = document.body.clientHeight;
}
document.write("La largeur disponible est de " + largeur +"
pixels.<br />");
document.write("La hauteur disponible est de " + hauteur +" pixels.");
//]]>
</script>
</body>
</html>
```

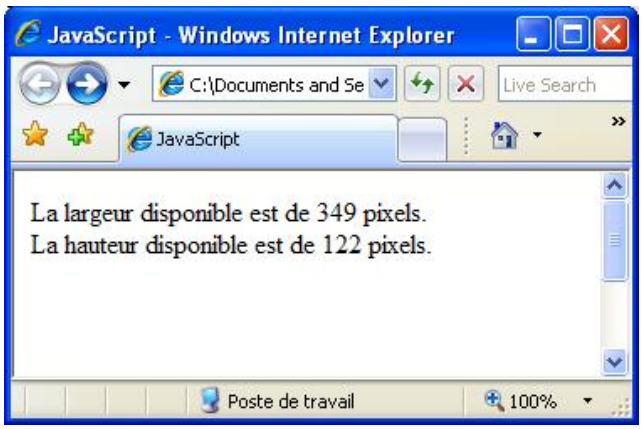
Commentaires

- Après avoir défini les variables `largeur` et `hauteur`, si la propriété `window.innerWidth` est reconnue dans ce cas nous avons à faire à Firefox. Ceci est une autre façon de tester le navigateur de l'utilisateur. Plutôt que de passer par les informations de l'objet navigator vues au point Quelques autres objets JavaScript - L'objet Navigator de ce chapitre, on teste simplement si l'objet ou la propriété existe pour gagner des lignes de code.!mb o,

- Pour être tout à fait complet, on signale dans les sites spécialisés dans la compatibilité des navigateurs (par exemple, www.quirksmode.org), que dans le cas d'un fichier Html ou Xhtml avec une déclaration de DTD, il est nécessaire, dans le cas d'Internet Explorer, de passer par : `document.documentElement.clientWidth` et `document.documentElement.clientHeight`.Ceci illustre bien que dans certaines situations, JavaScript peut se révéler hautement incompatible.

Capture d'écran pour Firefox :

Capture d'écran pour Internet Explorer :



Le XML

Dans le terme AJAX, le X signifie XML (*Asynchronous JavaScript and **XML***). Ainsi, il convient d'introduire ou de rappeler quelques éléments de ce langage.

Le XML est un métalangage, soit un langage pour écrire d'autres langages.

Même s'il n'y a que peu de chances que vous conceviez un jour votre propre langage, le XML est une véritable révolution dans le panorama de la publication sur le Web. Il apparaît aujourd'hui comme incontournable car il est déjà à la base de toute une série de nouveaux langages qui sont, ou qui seront, utilisés dans la conception des pages Web. Ces nouveaux langages générés à partir du XML en reprennent l'esprit, les règles et la syntaxe que vous découvrirez ci-après.

Au tout début de l'informatique et de l'Internet, existait déjà le SGML (*Standard Generalized Markup Language*), un langage normalisé pour la conception de langages de balises. Cette norme internationale (ISO8879) a comme objectif de décrire la structure et le contenu de différents types de documents électroniques. Le SGML a la particularité d'être très concis mais aussi très abstrait. En conséquence, il n'est que très difficilement abordable par les non-initiés. Les langages qui en découlent sont pourtant assez nombreux et vous ne pouvez pas ignorer un de ses fils qui est un langage de balises utilisé pour la publication sur le Web : le HTML (*HyperText Markup Language*).

Le Html ayant mal vieilli au fil des versions, le W3C Consortium, qui régit les règles de la publication sur le Web, a décidé de repartir d'une feuille blanche en revenant en quelques sortes aux sources. D'où le XML (*eXtensible Markup Language*) qui présente de fortes similitudes avec le SGML dont il est issu. Ainsi, le XML peut être considéré comme un SGML simplifié ou un SGML qui serait plus abordable.

Le XML (*eXtensible Markup Language*) est donc un langage de balises comme le Html mais il est extensible ou autrement dit, évolutif. En XML, les balises ne sont pas prédéfinies. C'est à vous de définir vos propres balises.

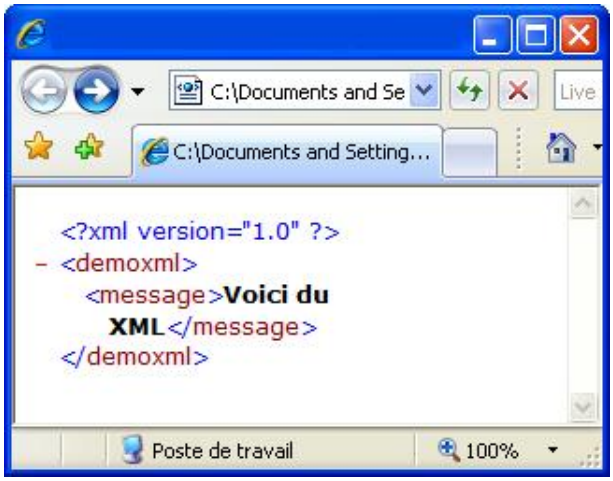
Si les navigateurs affichent sans problème les balises prédéfinies du Html comme `<h1>`, `<br>` ou autres `<table>`, que doivent-ils faire avec vos balises du style `<membre>` ou `<nouveau>` ? Le XML a comme vocation de décrire de l'information et non pas de l'afficher. Ainsi le XML pourtant créé en 1999, est resté durant quelques années un concept plutôt théorique faute de navigateurs capable d'afficher ce format. Avec le développement de nouvelles techniques comme le XSL (*Extended Stylesheet Language* ou langage de feuilles de style extensible), il est devenu maintenant possible de percevoir concrètement les énormes potentialités de ce nouveau langage.

En conclusion, si le Html a été conçu pour afficher de l'information, le XML a été créé pour structurer de l'information. À lui seul, il n'est censé rien faire d'autre !

Voici un premier exemple de XML.

```
<?xml version="1.0"?>
<demoxml>
<message>Voici du XML</message>
</demoxml>
```

Ce qui affiché dans le navigateur Internet Explorer donne le résultat suivant.



Il n'y a pas de quoi se réjouir sur le plan esthétique... Mais le XML n'est finalement que de l'information structurée, encodée entre des balises. Il faudra d'autres éléments comme une déclaration de feuilles de style CSS ou un fichier XSL, pour que le navigateur puisse "comprendre" vos balises et afficher ce fichier sous une forme plus conviviale.

Si le Html a régné en maître sur le Web durant la dernière décennie du 20ème siècle (1990 à 2000), le XML est sans

aucun doute possible, le standard omniprésent pour tout ce qui concerne la structuration et l'échange de données durant les premières décennies du 21ème siècle.

Le XML, le Html et le Xhtml

Lorsque les techniques de publication sur le Web sont abordées, une comparaison entre le Html, le Xhtml et le XML s'impose. Au contraire de ce qui a déjà été écrit par ailleurs, le XML n'est pas le successeur du Html. Le XML, le Xhtml et le Html sont des langages distincts !

1. Une seule similitude : le SGML

Le seul point commun entre le Html et le XML est qu'ils sont issus tous deux de la même "mère" soit le SGML (*Standardized Generalised Markup Language*) qui est le langage de référence en milieu professionnel pour tout ce qui concerne la gestion électronique des documents. Ils sont donc, tous deux, des langages de balises (voir le *Markup Language*). Ils ont également des caractéristiques communes héritées du SGML qui consistent à transporter sur le Web des données en mode texte (*plain text*), compatibles avec n'importe quelle plate-forme logicielle.

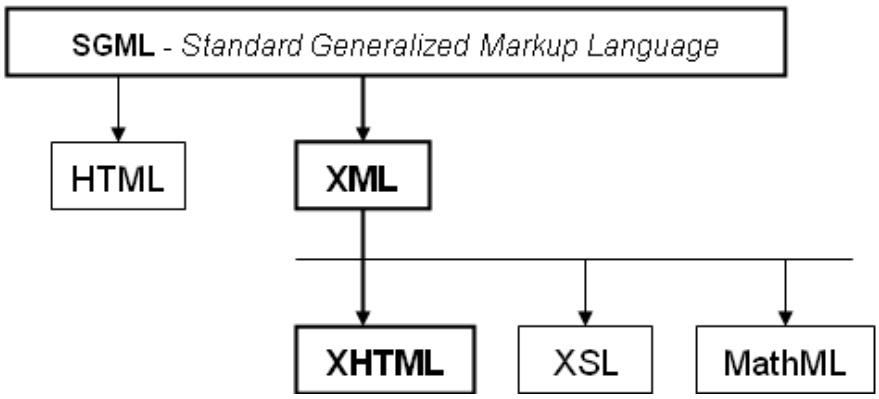
2. Les différences entre le Html et le XML

Le Html et le XML diffèrent sur de très nombreux points dont certains ont trait à l'essence même du langage.

- Le XML décrit, structure, échange des données tandis que le Html a pour vocation d'afficher des données.
- Le XML est un générateur de langages (métalangage). Le Html est un langage normalisé de publication sur le Web.
- Le XML est extensible. Il permet de créer ses propres balises en fonction des données traitées. En Html, les balises sont prédéfinies et elles sont donc figées.
- Le XML est un langage strict dont l'écriture doit être rigoureuse. La syntaxe du Html est devenue très (trop ?) permissive à cause des navigateurs récents.

3. Le Xhtml

Le Xhtml est présenté comme le successeur du Html. Mais il est surtout un des "enfants" engendrés par le XML. Pour contourner le problème des dérives du Html au fil des différentes versions, le W3C a conçu le Xhtml comme étant une reformulation du Html 4.0 selon la syntaxe et les règles du XML.



Le Xhtml a ainsi, entre autres choses, hérité de la structure rigoureuse du XML et de sa rigueur d'écriture.

La syntaxe du XML

Le XML impose des règles de syntaxe très strictes et très spécifiques par rapport au Html. On retrouvera ces mêmes règles de syntaxe dans tous les langages dérivés du XML, dont le Xhtml.

Passons ces règles en revue :

- Le XML est un langage de balises. Mais au contraire du Html où les balises sont définies, vous devez ou pouvez inventer vos propres balises. Rappelez-vous, le XML est eXtensible. Il faut donc définir soi-même le nom des balises utilisées.

  Il y a quelques règles pour la composition des noms (mais elles ne dérouteront pas les habitués du JavaScript) :

- Les noms peuvent contenir des lettres, des chiffres ou d'autres caractères.

- Les noms ne peuvent débuter par un nombre ou un signe de ponctuation.

- Les noms ne peuvent commencer par les lettres xml (ou XML ou Xml...).

- Les noms ne peuvent contenir des espaces.

- La longueur des noms est libre mais on conseille de rester raisonnable.

- On évitera certains signes qui pourraient prêter à confusion comme le tiret, le point virgule, le point, la virgule, le signe plus grand que (>) et le signe plus petit que (<).

- Les balises sont sensibles aux majuscules et minuscules (case sensitive). Ainsi, la balise `<Message>` est différente de la balise `<message>`. La balise d'ouverture et la balise de fermeture doivent donc être identiques. Ainsi par exemple `<Message>` ... `</message>` est incorrect et `<message>` ... `</message>` est correct.

- Un consensus se dégage pour n'écrire les balises qu'en minuscules, limitant ainsi les erreurs possibles.

- Toute balise ouverte doit être, impérativement, fermée.

- Les mauvaises pratiques du Html avec lesquelles on pouvait dans certains cas omettre la balise de fin comme pour le paragraphe `<p>` ou l'élément de liste `<li>` sont révolues.

Ainsi en Html, ce qui suit est affiché correctement :

```
<p>
<ul>
<li>Point 1
<li>Point 2
```

Le XML est beaucoup plus strict. On devrait avoir :

```
<p>
<ul>
<li>Point 1</li>
<li>Point 2</li>
</ul>
<p>
```

Les éventuelles balises uniques, appelées aussi balises vides, comme `<br>`, `<meta>` ou `<img>` en Html, doivent également comporter un signe de fermeture soit balise/. Ainsi une balise `<meta/>` serait correcte en XML.

- Les balises doivent être correctement imbriquées. Le XML attachant beaucoup d'importance à la structure des données, des balises mal imbriquées sont des fautes graves de sens.

  Ainsi l'écriture suivante est incorrecte car les balises ne sont pas bien imbriquées :

`<parent><enfant>Marine</parent></enfant>`

L'écriture correcte avec une bonne imbrication des éléments est : `<parent><enfant>Marine</enfant></parent>`

- Tout document XML doit comporter une racine.

  En fait, la première paire de balises d'un document XML sera considérée comme la balise de racine (*root*).

Par exemple :

```
<racine>
... suite du document XML ...
</racine>
```

En comparaison avec le Html ou le Xhtml, l'élément racine est `<html> ... </html>`.

Tous les autres éléments sont imbriqués entre ces balises de racine.

Par exemple :

```
<racine>
<parents>
<enfants>
<petits_enfants> ... </petits_enfants>
</enfants>
</parents>
</racine>
```

- Les valeurs des attributs doivent toujours être mises entre des guillemets. Le XML peut (comme le Html) avoir des attributs comportant des valeurs. En XML, les valeurs des attributs doivent obligatoirement être reprises entre des guillemets, à l'inverse du Html où leur présence ou leur absence n'a plus beaucoup d'importance pour les navigateurs.

  Ainsi, l'écriture suivante est incorrecte car il manque les guillemets.

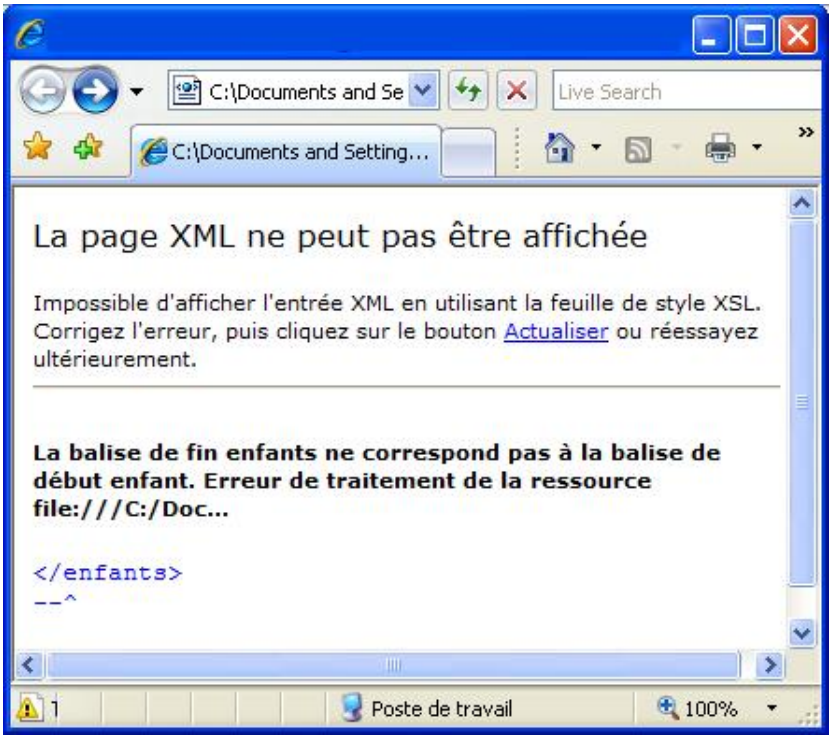
  ```
  <enfant date_anniversaire=071185>
  ```

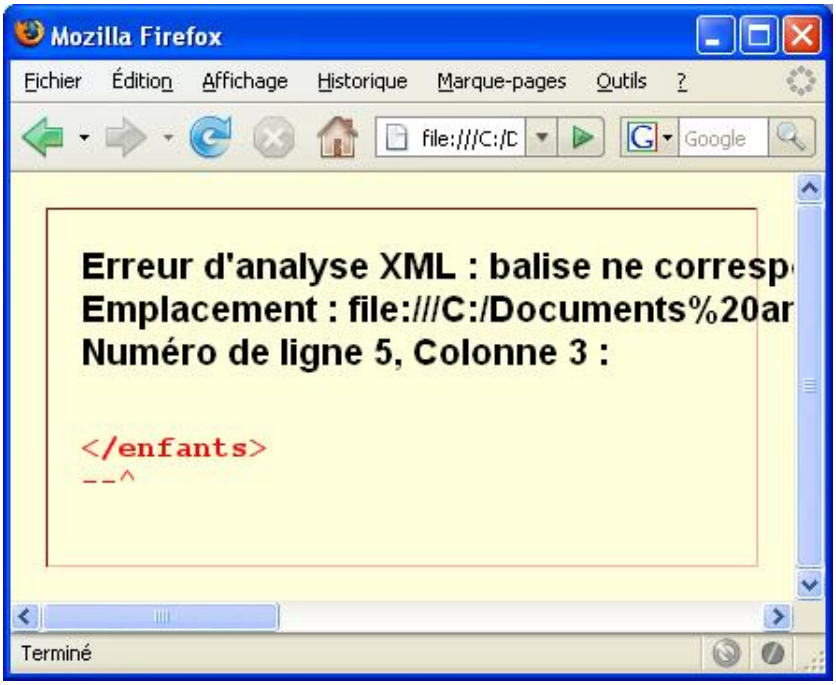
  La bonne écriture est :

  ```
  <enfant date_anniversaire="071185">
  ```

- Le XML est un langage strict. Votre document doit impérativement respecter la syntaxe du XML. On dit alors que le document est **bien formé** *(Well-formed).* Seuls les documents "bien formés" sont affichés correctement. À la moindre erreur de syntaxe, le document n'est pas affiché !

Voici ce qui est affiché dans Internet Explorer et Firefox lorsque le document est incorrect.





Les navigateurs sont aussi un outil de débogage rudimentaire du code XML.

Un premier document XML

Voici un premier document XML qui sera étoffé en cours de chapitre.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
```

La déclaration `<?xml version="1.0"?>` indique au navigateur que ce qui suit est un document XML selon sa version 1.0. Vous remarquerez que cette balise ne comporte pas de signe de fermeture car cette balise n'est pas encore du XML.

Le jeu de caractères à utiliser est généralement notifié à l'intention de l'interpréteur XML (*Parser*). Le jeu de caractères ISO-8859-15 a, pour nous francophones, l'avantage d'accepter la plupart des lettres avec des accents et le signe euro. De façon native, le XML est conçu par le jeu de caractères UTF-8.

```
<racine>
```

L'élément racine indispensable au XML. Vous pouvez utiliser, à votre convenance, n'importe quel nom pour cette balise de racine.

```
... suite du document XML ...
```

Votre document XML proprement dit, qui respecte bien entendu scrupuleusement la syntaxe du XML (bien formé).

```
</racine>
```

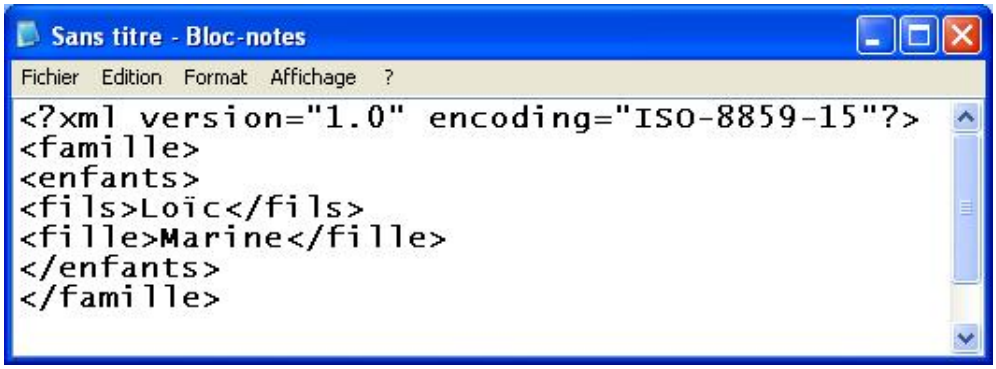
Le document XML se termine obligatoirement par la fermeture de la balise de racine.

Exemple

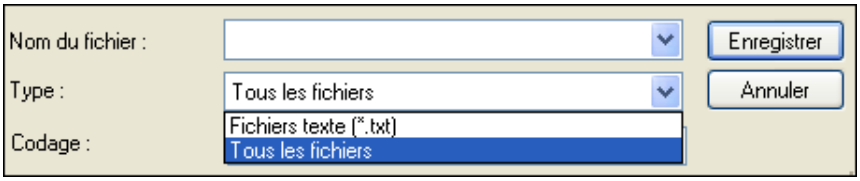
Voici un petit fichier XML.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<famille>
<enfants>
<fils>Loïc</fils>
<fille>Marine</fille>
</enfants>
</famille>
```

On le reproduit dans le programme Bloc-notes (*Notepad*) pour les utilisateurs de Windows.

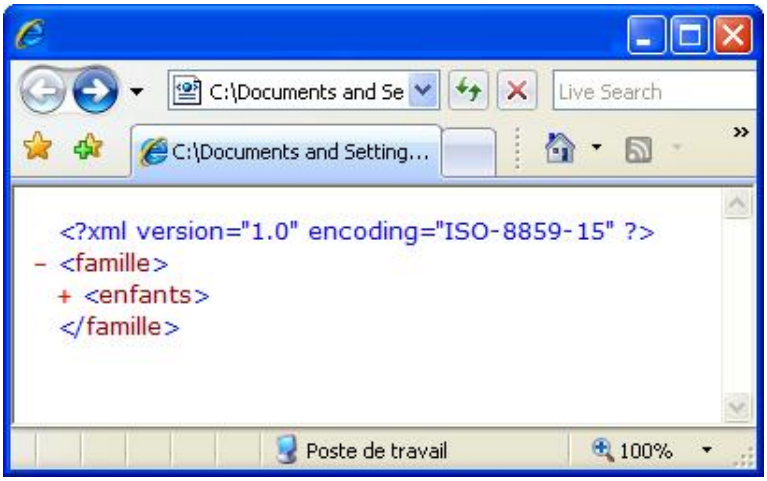


Et on l'enregistre en **Type : Tous les fichiers** sous un nom avec une extension .xml.

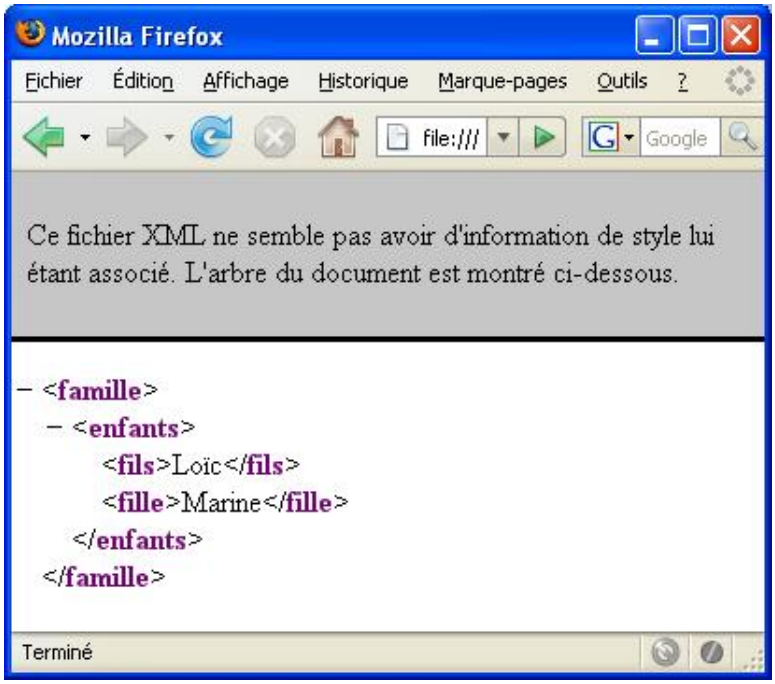


Résultat dans Microsoft Internet Explorer 7 :

Vous remarquerez qu'il y a un petit signe **moins** affiché devant des balises englobantes. Il suffit de cliquer sur le signe pour masquer celles-ci, et bien entendu, de cliquer sur le signe **plus** pour les faire réapparaître.



Résultat sous Firefox :



Firefox fait remarquer qu'il n'y a aucune information de style associée au fichier XML afin de l'afficher valablement. Ainsi

seule la structure du document (appelée aussi l’arbre) peut être affichée.

Le DOCTYPE

Le DOCTYPE ou DTD (*Document Type Definition*) est l'ensemble des règles et des propriétés que doit suivre le document XML produit. Ces règles définissent généralement le nom et le contenu de chaque balise ainsi que le contexte dans lequel elles doivent exister. Cette formalisation des éléments est particulièrement utile lorsqu'on utilise de façon récurrente des balises dans un document XML ou que plusieurs personnes sont appelées à travailler sur le même document XML.

Les DTD sont écrites selon les prescriptions du SGML.

L'étude détaillée des DTDs dépasse de loin le cadre de cet ouvrage mais un aperçu est cependant utile, surtout pour comprendre le fonctionnement des langages dérivés du XML (comme le Xhtml) qui ne manquent pas d'utiliser ces fameux DTDs.

Dans certains cas, plusieurs concepteurs peuvent se mettre d'accord pour utiliser un DTD commun pour échanger leurs données. C'est le cas du Xhtml, où le DOCTYPE est signalé dans l'en-tête du document (strict, transitional ou frameset).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Cependant, dans la plupart des applications XML, le concepteur doit écrire sa propre définition de document.

1. DTD interne

On peut inclure le DOCTYPE au code du fichier XML. On parlera alors d'un DTD interne.

Un DTD interne suit la syntaxe suivante :

```
<!DOCTYPE élément-racine [
déclaration des composants
]>
```

Du point de vue des DTDs, tous les documents XML sont élaborés avec les composants suivants :

- les éléments ou plus communément les balises. Exemple : la balise `<body>` en Xhtml.
- les attributs qui viennent compléter les balises. Les attributs sont toujours inclus dans la balise ouvrante.

  Exemple : l'attribut **border** de la balise `<table>` en Xhtml.
- Les PCDATA (*parsed character data*), chaînes de caractères qui seront affichées par l'interpréteur XML (Parser). Exemple : le contenu de la balise `<p>Chapitre 1` : Le Xhtml`</p>` en Xhtml.
- Les CDATA (*character data*), chaînes de caractères qui ne sont pas prises en compte et qui ne sont donc pas affichées par le navigateur XML.

  Exemple, du code JavaScript dans un document Xhtml.
- Les entités, sorte de raccourci qui permet de déclarer plusieurs paramètres utilisables en appelant simplement l'entité plus loin dans le document.

Détaillons ci-après quelques règles pour définir ces différents composants.

a. La déclaration d'un élément

Un élément se déclare dans la DTD selon la syntaxe suivante :

```
<!ELEMENT nom_de_l'élément mot-clé>
ou
<!ELEMENT nom_de_l'élément (contenu)>
```

Exemple

`<!ELEMENT h1>` pour la balise `<h1>` en Xhtml.

b. Les éléments vides

Un élément vide (*empty*) se déclare :

```
<!ELEMENT nom_de_l'élément EMPTY>
```

Exemple

`<!ELEMENT img EMPTY>` pour la balise `<img />` en Xhtml.

c. Les éléments comprenant des caractères à afficher

Les éléments comprenant des caractères à afficher, généralement le contenu des balises, se notent :

```
<!ELEMENT nom_de_l'élément (#PCDATA)>
```

Exemple

`<!ELEMENT p (#PCDATA)>` pour la balise de paragraphe `<p>` en Xhtml.

d. Les éléments avec des éléments enfant

```
<!ELEMENT nom_de_l'élément (elément_enfant,élément_enfant,...)>
```

Exemple

```
<!ELEMENT <enfants (fils,fille)>
```

Lorsque des éléments enfant sont déclarés, ces éléments enfant doivent apparaître dans le même ordre dans le document XML.

En outre, les éléments enfant doivent être déclarés.

```
<!ELEMENT <enfants (fils,fille)>
<!ELEMENT fils (#PCDATA)>
<!ELEMENT fille (#PCDATA)>
```

e. Les éléments avec une seule occurrence

Les éléments enfant qui doivent apparaître seulement une fois dans le code de l'élément parent se notent :

```
<!ELEMENT nom_de_l'élément (elément_enfant)>
```

Exemple

```
<!ELEMENT adhérent (nom)>
```

L'élément enfant nom doit apparaître une fois et seulement une fois dans l'élément parent `adhérent`.

f. Les éléments avec au minimum une occurrence

Les élément qui peuvent apparaître plusieurs fois se déclarent :

```
<!ELEMENT nom_de_l'élément (elément_enfant+)>
```

Exemple

```
<!ELEMENT adhérent (telephone+)>
```

Le signe + déclare que l'élément enfant `telephone` doit apparaître une fois mais peut aussi apparaître plusieurs fois dans l'élément parent `adhérent`.

g. Les éléments avec au minimum zéro occurrence

Les éléments facultatifs qui peuvent ne pas apparaître (zéro occurrence) ou apparaître plusieurs fois se notent :

```
<!ELEMENT nom_de_l'élément (elément_enfant*)>
```

Exemple

```
<!ELEMENT adhérent (telephonefixe*)>
```

Le signe * déclare que l'élément enfant `telephonefixe` peut ne pas apparaître, tout comme il peut apparaître plusieurs fois dans l'élément `adhérent`.

h. Les éléments avec zéro ou une occurrence

Les éléments qui peuvent apparaître qu'une fois ou être absents se définissent :

```
<!ELEMENT nom_de_l'élément (elément_enfant?)>
```

Exemple

```
<!ELEMENT adhérent (adresseIP?)>
```

Le signe ? déclare que l'élément enfant `adresseIP` peut être absent ou apparaître une seule fois dans l'élément parent `adhérent`.

i. Les éléments alternatifs

Lorsque des options (le ou logique) sont autorisées dans les éléments enfants, celles-ci se déclarent :

```
<!ELEMENT nom_de_l'élément ((elément_enfant|élément_enfant))>
```

Exemple

```
<!ELEMENT telephone ((national|international))>
```

L'exemple déclare que l'élément parent `telephone` peut contenir l'élément enfant `national` ou l'élément enfant `international`.

Appliquons une DTD interne à notre fichier XML.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<famille>
<enfants>
<fils>Loïc</fils>
<fille>Marine</fille>
</enfant>
</famille>
```

Celui-ci devient :

```
<?xml version="1.0" "standalone="yes"?>
<!DOCTYPE famille [
<!ELEMENT famille (enfants?)>
<!ELEMENT enfants (fils*, fille*)>
<!ELEMENT fille (#PCDATA)>
<!ELEMENT fils (#PCDATA)>
]>
<famille>
<enfants>
```

```
<fils>Loïc</fils>
<fille>Marine</fille>
</enfants>
</famille>
```

Commentaires :

- Lorsqu'un doctype interne est défini, il faut déclarer que le fichier est indépendant (`standalone="yes"`) dans le prologue XML.
- L'élément racine est déclaré dans le début du DOCTYPE (`<!DOCTYPE famille`).
- Le contenu du DOCTYPE est encodé entre des crochets ouvrants et fermants.
- La ligne `<!ELEMENT famille (enfants?)>` déclare que la balise `<famille>` contient la balise `<enfants>`. Le signe ? prévoit le cas des familles sans enfants.
- La ligne `<!ELEMENT enfants (fils*, fille*)>` déclare que la balise <enfants> contient les balises `<fils>` et `<fille>`. Le signe * est prévu pour le cas où une descendance n'est composé que d'un ou plusieurs fils ainsi que d'une ou plusieurs fille(s).
- Le DOCTYPE se termine par le signe >.

2. DTD externe

Le DTD externe suit la syntaxe suivante :

```
<!DOCTYPE élément-racine SYSTEM "nom_du_fichier.dtd">
```

Le même fichier que ci-dessus est alors :

```
<!DOCTYPE famille SYSTEM "parent.dtd">
<?xml version="1.0" standalone="no"?>
<famille>
<enfants>
<fils>Loïc</fils>
<fille>Marine</fille>
</enfants>
</famille>
```

Le fichier de DTD externe (ici dans le même répertoire) `parent.dtd` contient :

```
<!ELEMENT famille (enfants?)>
<!ELEMENT enfants (fils*, fille*)>
<!ELEMENT fille (#PCDATA)>
<!ELEMENT fils (#PCDATA)>
```

Mais il est aussi possible de faire référence à un DTD externe situé sur un autre site comme pour, par exemple, le XHTML :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Un document est dit valide s'il respecte les règles spécifiques de son DOCTYPE.

En aparté

Il existe une autre méthode pour définir les DTDs, non plus en SGML mais en XML, c'est le XML Schema.

Le XML Schema est un langage de description de format de document (*XML Schema Description*), encodé en XML, permettant de définir la structure d'un document XML. Ce fichier de description de structure tient le rôle de DTD et permet également de valider le document XML.

Afficher le XML avec CSS

Pour afficher les balises XML, on peut faire appel aux feuilles de style CSS, maintenant classiques dans le paysage de la publication sur le Web. Pour chaque balise "inventée" dans le fichier XML, un élément de style va être intégré pour l'affichage par le navigateur.

Voici un exemple des possibilités des feuilles de style CSS associées à un document XML.

Soit notre document XML de départ :

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
<editeur>Columbia Tristar</editeur>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
<editeur>Virgin France</editeur>
</mp3>
<mp3>
<titre>>Sheller en solitaire</titre>
<artiste>William Sheller</artiste>
<date>1992</date>
<editeur>Mercury</editeur>
</mp3>
<mp3>
<titre>Caché derrière</titre>
<artiste>Laurent Voulzy</artiste>
<date>1993</date>
<editeur>Ariola</editeur>
</mp3>
</compilation>
```

Affiché dans le navigateur, ce code XML s'affiche comme suit :

Un fichier .css est ajouté, voici son contenu (xmlcss.css) :

```
compilation , mp3 {}
titre { display: block;
        width: 250px;
        font-size: 12pt;
        font-family: Arial;
        font-weight: bold;
        background-color: #9cf;
        color: black;
        padding-left: 10px;}
artiste { display: block;
          font-size: 12pt;
          padding-left: 10px;}
date { display: block;
       font-size: 12pt;
       color: red ;
       font-weight: bold;
       padding-left: 10px;}
editeur { display: block;
          font-size: 11pt ;
          font-style: italic;
          font-family: Arial ;
          padding-left: 10px;}
```

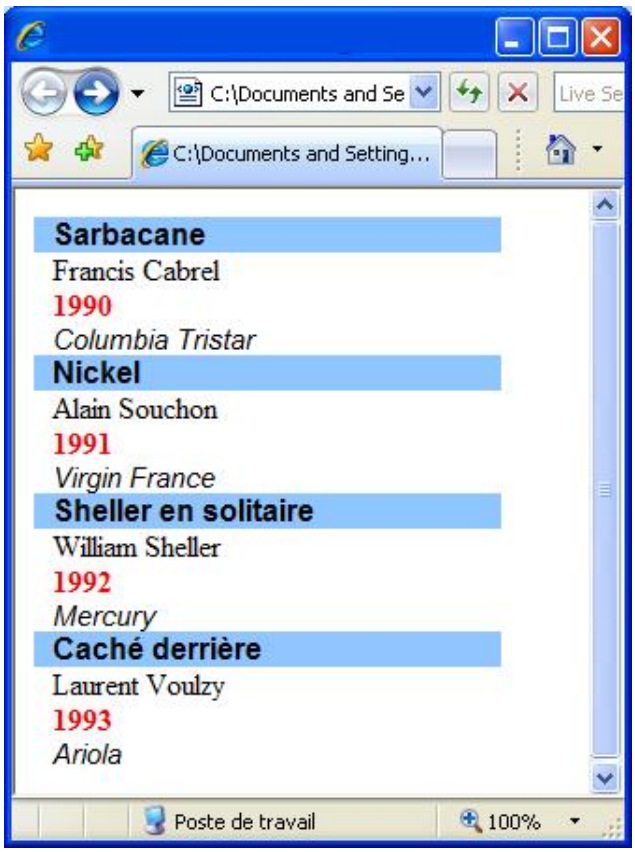
Après avoir ajouté un lien vers le fichier css dans le fichier XML :

```
<?xml-stylesheet href="xmlcss.css" type="text/css"?>
```

Le code complet devient :

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<?xml-stylesheet href="xmlcss.css" type="text/css"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
<editeur>Columbia Tristar</editeur>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
<editeur>Virgin France</editeur>
</mp3>
<mp3>
<titre>Sheller en solitaire</titre>
<artiste>William Sheller</artiste>
<date>1992</date>
<editeur>Mercury</editeur>
</mp3>
<mp3>
<titre>Caché derrière</titre>
<artiste>Laurent Voulzy</artiste>
<date>1993</date>
<editeur>Ariola</editeur>
</mp3>
</compilation>
```

On obtient finalement dans le navigateur :



Assez efficace, non ?... Mais il y a encore un autre moyen, plus performant et ayant des possibilités plus étendues : afficher du XML avec le XSL soit le langage de feuilles de style eXtensible : le pendant du XML aux feuilles de style CSS.

Afficher le XML avec XSL

Pour afficher des documents XML, il est nécessaire de disposer d'un mécanisme qui permettrait de décrire l'affichage du document dans le navigateur. Les feuilles de style CSS constituent un de ces mécanismes mais le XSL (*eXtensible Stylesheet Language*) est de loin un langage de feuille de style plus adapté au XML et donc plus performant.

Nous y reviendrons en détail au chapitre Introduction au XSL.

À seule fin de démonstration, voici un exemple des possibilités du XSL associé à un document XML.

Reprenons notre document XML de départ :

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
<editeur>Columbia Tristar</editeur>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
<editeur>Virgin France</editeur>
</mp3>
<mp3>
<titre>Sheller en solitaire</titre>
<artiste>William Sheller</artiste>
<date>1992</date>
<editeur>Mercury</editeur>
</mp3>
<mp3>
<titre>Caché derrière</titre>
<artiste>Laurent Voulzy</artiste>
<date>1993</date>
<editeur>Ariola</editeur>
</mp3>
</compilation>
```

On ajoute un fichier XSL (simple.xsl) dont voici le contenu :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body style="font-family: Arial; font-size: 12pt;">
<xsl:for-each select="compilation/mp3">
<div style="background-color: #9cf; color: black;">
<span style="font-weight: bold; padding-left: 4px">
<xsl:value-of select="titre"/></span>
- <xsl:value-of select="artiste"/>
</div>
<div style="margin-left: 20px; font-size: 11pt">
<span><xsl:value-of select="date"/>
</span>
<span style="font-style: italic;">
<br/>
<xsl:value-of select="editeur"/>
</span>
</div>
</xsl:for-each>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

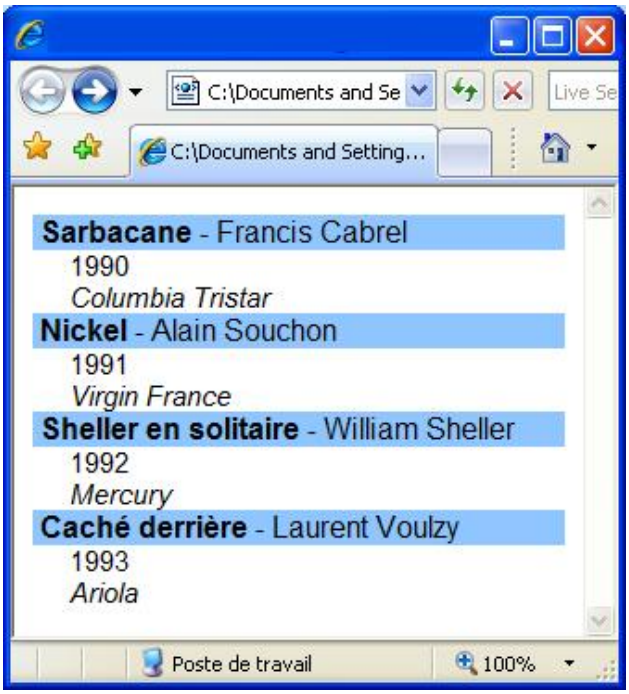
Après avoir ajouté un lien vers le fichier XSL dans le fichier XML :

```
<?xml-stylesheet type="text/xsl" href="simple.xsl"?>
```

Le code complet devient :

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<?xml-stylesheet type="text/xsl" href="simple.xsl"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
<editeur>Columbia Tristar</editeur>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
<editeur>Virgin France</editeur>
</mp3>
<mp3>
<titre>Sheller en solitaire</titre>
<artiste>William Sheller</artiste>
<date>1992</date>
<editeur>Mercury</editeur>
</mp3>
<mp3>
<titre>Caché derrière</titre>
<artiste>Laurent Voulzy</artiste>
<date>1993</date>
<editeur>Ariola</editeur>
</mp3>
</compilation>
```

On obtient finalement :



Afficher du XML dans du Xhtml

On peut toujours incorporer du XML dans un fichier Xhtml ou Html avec la balise `<xml> ... </xml>`. Mais, en toute logique, quand les navigateurs rencontrent des balises incorrectes ou inconnues, rien n'est affiché. Ce sera le cas avec vos balises XML incorporées dans un fichier Xhtml (ou Html). Heureusement, nous allons utiliser une technique intitulée "îlots de données" (*Data Islands*).

Cette technique offre une possibilité assez intéressante. Dans un fichier Xhtml, on peut créer un "îlot de données" XML, se trouvant dans un fichier XML distinct, et en extraire des données que l'on pourra alors inclure dans le document Xhtml.

Dans le fichier Xhtml, on va désigner le fichier XML extérieur par un identifiant *id* :

```
<xml id="fichierxml" src="exemple.xml"></xml>
```

Ensuite, dans un tableau Xhtml, que l'on relie par un attribut à la source des données au moyen de l'identifiant id désigné plus haut (`datasrc="#id"`), des données du fichier XML peuvent être reprises. Pour ce faire, un attribut de champ de données ayant pour valeur le nom de la balise XML est utilisé (`datafld= "balise_xml"`).

Voilà le fichier XML extérieur (exemple.xml) :

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
<editeur>Columbia Tristar</editeur>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
<editeur>Virgin France</editeur>
</mp3>
<mp3>
<titre>Sheller en solitaire</titre>
<artiste>William Sheller</artiste>
<date>1992</date>
<editeur>Mercury</editeur>
</mp3>
<mp3>
<titre>Caché derrière</titre>
<artiste>Laurent Voulzy</artiste>
<date>1993</date>
<editeur>Ariola</editeur>
</mp3>
</compilation>
```

Un fichier Xhtml (ou Html) doit être créé dans lequel on reprend des données du fichier XML et plus précisément le contenu des balises `<titre>`, `<artiste>` et `<date>`.

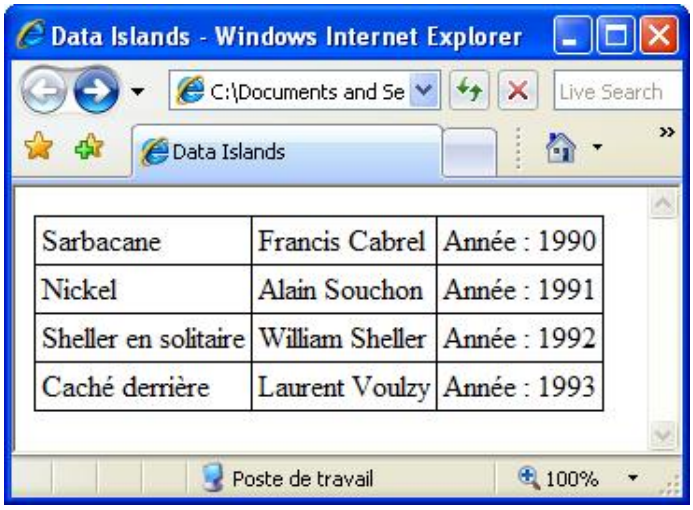
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Data Islands</title>
<meta http-equiv="Content-Type" content="application/xhtml+xml;
charset=iso-8859-15" />
</head>
<body>
<xml id="fichierxml" src="exemple.xml"></xml>
<table datasrc="#fichierxml">
<tr>
<td><span datafld="titre"></span></td>
<td><span datafld="artiste"></span></td>
<td>Année : <span datafld="date"></td>
</tr>
```

```
</table>
</body>
</html>
```

Ajoutons à ce fichier quelques déclarations de feuilles de style CSS afin d'améliorer sa présentation.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Data Islands</title>
<meta http-equiv="Content-Type" content="application/xhtml+xml;
charset=iso-8859-15" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<style type="text/css">
table { border-collapse: collapse;
        border: solid black 1px;}
td { padding: 3px;
     border: solid black 1px;}
</style>
</head>
<body>
<xml id="fichierxml" src="exemple.xml"></xml>
<table datasrc="#fichierxml" style="">
<tr>
<td><span datafld="titre"></span></td>
<td><span datafld="artiste"></span></td>
<td>Année : <span datafld="date"></td>
</tr>
</table>
</body>
</html>
```

On obtient finalement :



Des éditeurs XML

N'attendez pas de miracle des éditeurs XML ! Étant donné qu'en XML vous concevez vous-même vos balises, les éditeurs XML ne sont que des outils d'aide à l'encodage et à la structuration de votre document.

Les éditeurs XML sont cependant d'une grande utilité si de nombreuses balises récurrentes se présentent dans votre document XML. En outre, s'il est nécessaire d'actualiser souvent les données du fichier XML, il est plus facile de retrouver une information dans l'interface d'un éditeur XML que dans une quantité innombrable de balises du code source.

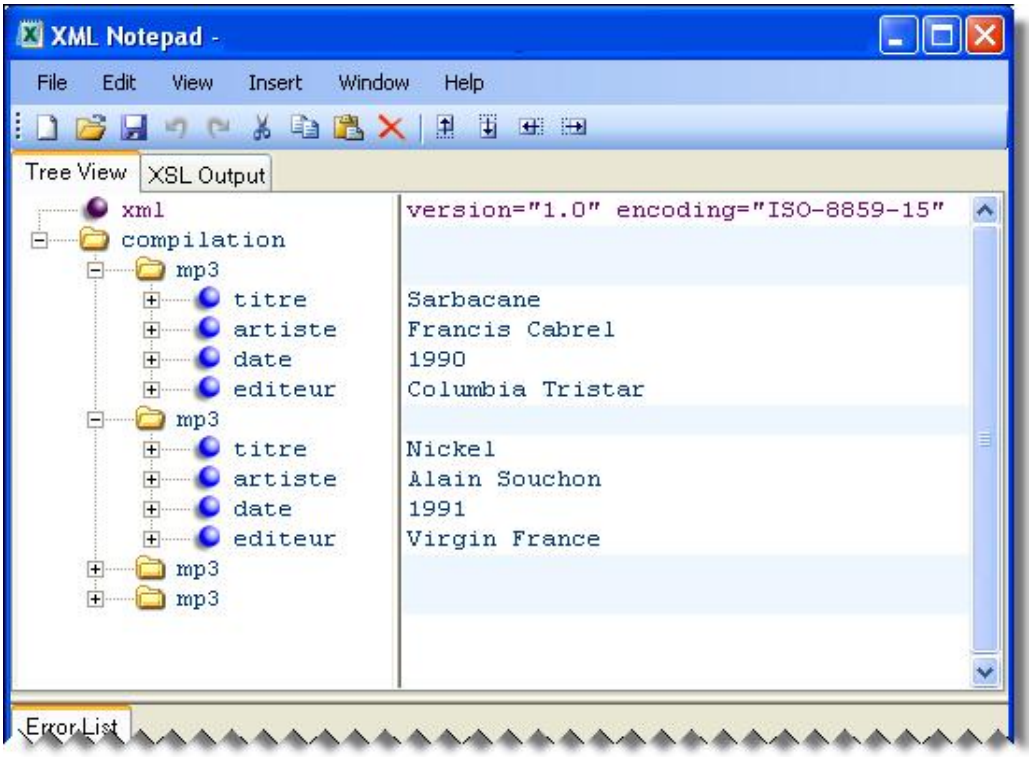
1. Microsoft XML Notepad 2007

À la fin 2006, Microsoft a fortement remanié son logiciel XML Notepad 1.0, devenu complètement désuet car il datait de 1998. Ainsi XML Notepad 2007 (logiciel gratuit) se positionne comme un éditeur XML simple et parfaitement adapté à l'apprentissage du XML.

XML Notepad 2007 est une application qui vous permet de créer et de modifier rapidement des documents XML. L'interface présente deux volets : celui de gauche affiche graphiquement la structure arborescente du document et celui de droite les valeurs du fichier. Des éléments, des attributs, des commentaires peuvent être ajoutés dans le volet de gauche et leurs valeurs peuvent être encodées dans la zone de texte du volet droit correspondant.

Il est aussi possible de déplacer des éléments par un simple glisser/déposer (drag/ drop) et de visualiser le document en direct grâce à un document XSL associé.

- Ouvrons notre document XML (**File - Open**) dans XML Notepad 2007.



On remarque directement, dans le panneau de gauche, l'approche visuelle du document XML.

XML Notepad, comme la plupart des éditeurs XML, met l'accent sur la structuration et l'arborescence des données que

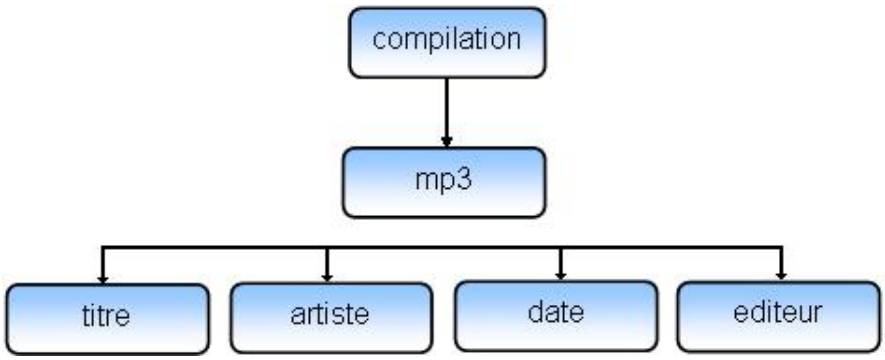
ne révèlent pas directement le code source et l'encodage brut des balises.

La balise <compilation> ... </compilation> est l'élément racine obligatoire, soit celui qui occupe la position la plus haute dans la hiérarchie. Elle est considérée comme l'élément parent de toutes les autres balises du document.

La balise <mp3> ... </mp3> est une descendance de la balise parent <compilation>. Elle fait partie des éléments enfant (*child*) de celle-ci.

Cette balise <mp3> est à son tour parent de différentes balises. Ainsi, les balises <titre>, <artiste>, <date> et <editeur> sont des balises enfant de cette dernière.

L'arborescence ou l'arbre de notre document XML peut donc se représenter comme suit.



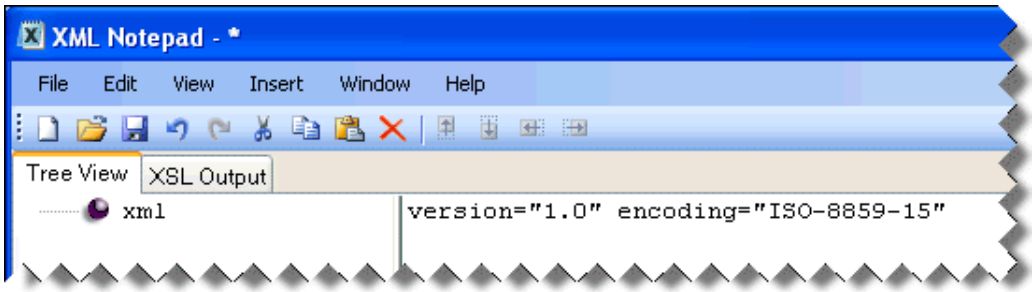
L'affichage de l'arborescence du document est des plus utile pour illustrer le DOM (*Document Object Model*), abordé dans le chapitre Le DOM (Document Object Model).

Pour une rapide prise en main, nous allons reproduire notre document XML avec le logiciel XML Notepad 2007.

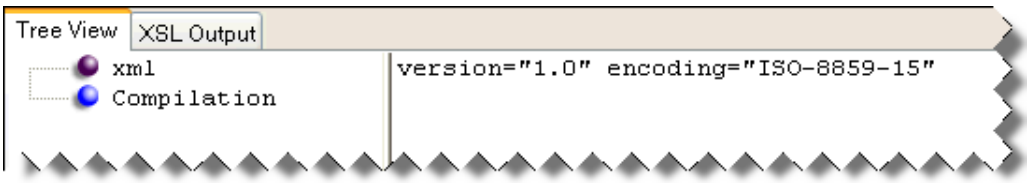
```
<?xml version="1.0" encoding="ISO-8859-15"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
<editeur>Columbia Tristar</editeur>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
<editeur>Virgin France</editeur>
</mp3>
Etc.
```

- Commençons par ouvrir un document vierge, **File - New**.

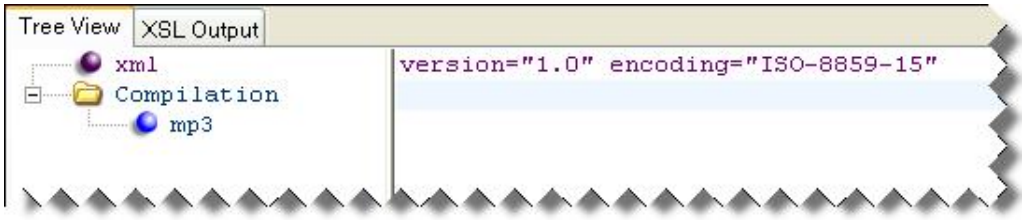
Il faut tout d'abord encoder la déclaration de document XML, **Insert - Processing Instruction** - Before. Une zone de texte apparaît dans le panneau de gauche, il faut introduire au clavier le mot **xml** et dans la zone de texte apparaissant dans le panneau de droite **version="1.0" encoding="ISO-8859-15"**.



- Il faut ensuite veiller à déterminer l'élément racine <compilation>, **Insert - Element - After**, on complète la zone de texte du panneau de gauche par le mot **compilation**.

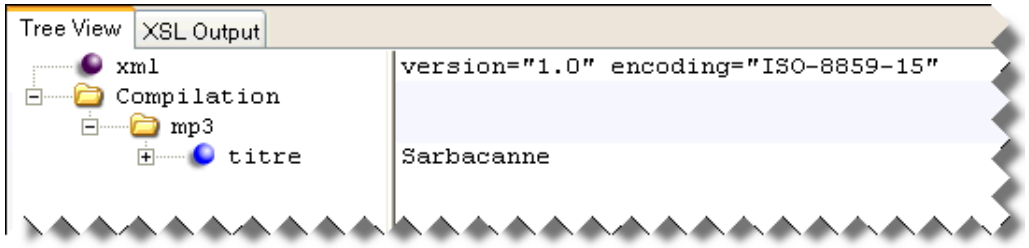


- Maintenant traitons des éléments enfant <mp3>. Après avoir sélectionné la ligne compilation, **Insert - Element - Child** il faut introduire au clavier le mot **mp3**.

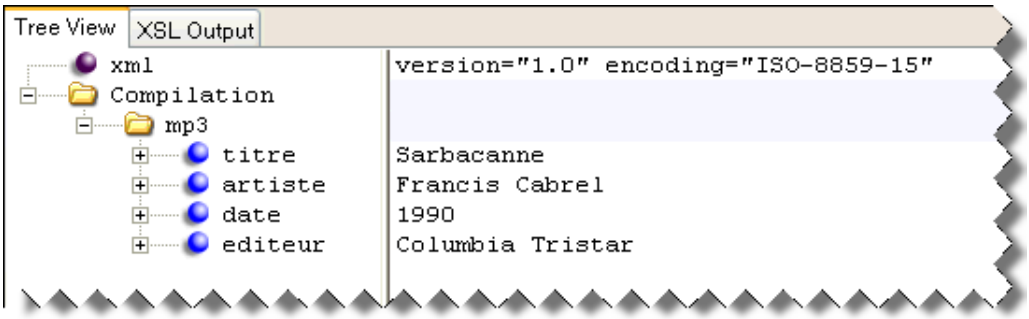


On remarque que l'icône en face de compilation n'est plus une puce mais un dossier, signe que la balise <compilation> est devenu une balise parent.

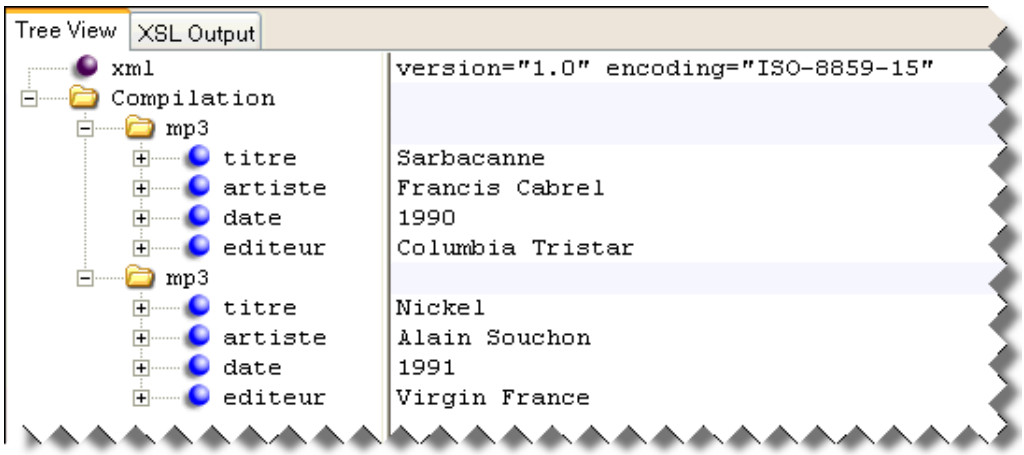
- Après avoir sélectionné le niveau mp3, préoccupons-nous des balises enfant de cette dernière. **Insert - Element - Child** puis introduire au clavier le mot **titre** et son contenu **Sarbacane** dans le panneau de droite.



- On procède de même pour les autres balises (artiste, date et editeur). Après avoir sélectionné l'élément parent mp3, **Insert - Element - Child** puis complété les informations.



- Il faut maintenant procéder de même avec les autres blocs mp3 du document. Après avoir sélectionné mp3, **Edit - Duplicate** puis modifier les informations dans le panneau de droite.



On répètera l'opération autant de fois pour insérer toutes les données du document XML.

- Après avoir enregistré le document (**File - Save As...**), le code source du fichier sera sous sa forme classique.

2. Altova XMLSpy 2007

Dans le paysage des éditeurs XML professionnels, un éditeur se détache du lot. En effet, XMLSpy existe dès les débuts du XML et l'interface et son utilisation sont devenues de plus en plus sophistiquées. XMLSpy d'Altova est très largement adopté par les concepteurs professionnels.

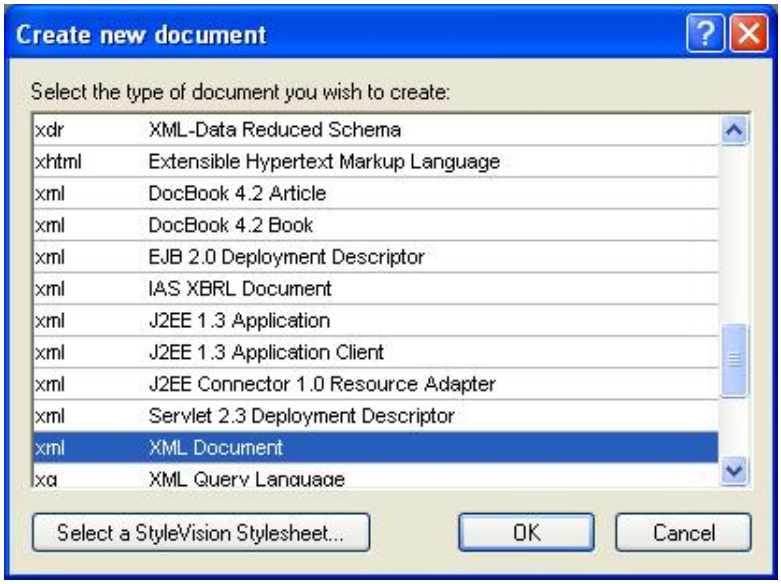
XML Spy est bien plus qu'un éditeur de balises XML car il permet aussi :

- de valider en direct le document XML généré ;
- de saisir un document XSLT ;
- de débugger un document XSLT ;
- de générer les DTD ;
- de convertir les DTD en Schema ;
- etc.

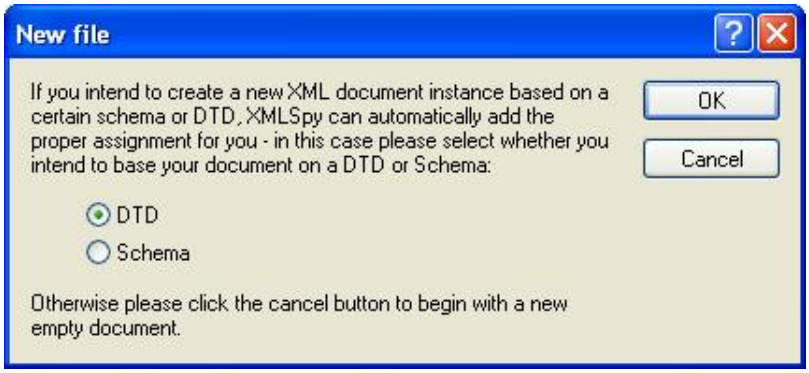
Bien qu'il existe une version freeware (*XMLSpy authentic 2007*), la version shareware (limitée à 30 jours) de la version professionnelle (*XMLSpy Professional Edition 2007*) permet quant à elle, de découvrir réellement les multiples potentialités du logiciel.

Pour donner un exemple de rapide prise en main de l'outil, reproduisons le même fichier XML.

- Ouvrez un fichier XML, soit **File - New** puis sélectionnez le type de document **xml - XML Document** et **OK**.

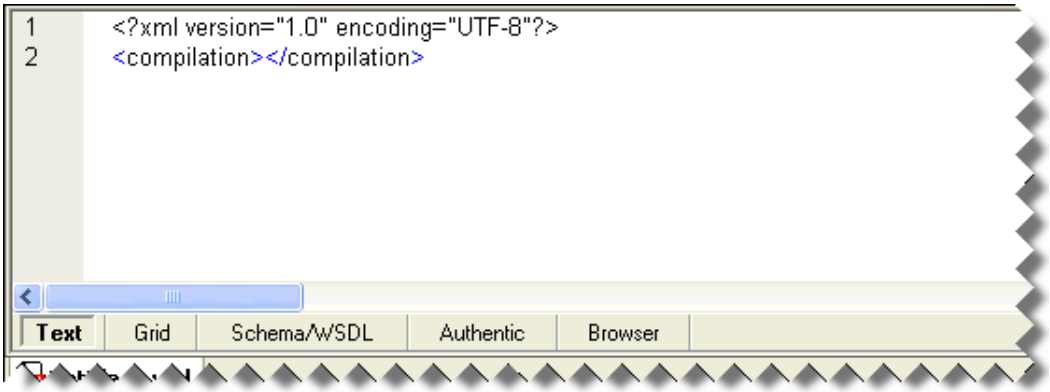


Il faut noter l'éventail des possibilités offertes permettant de traiter différents types de document.

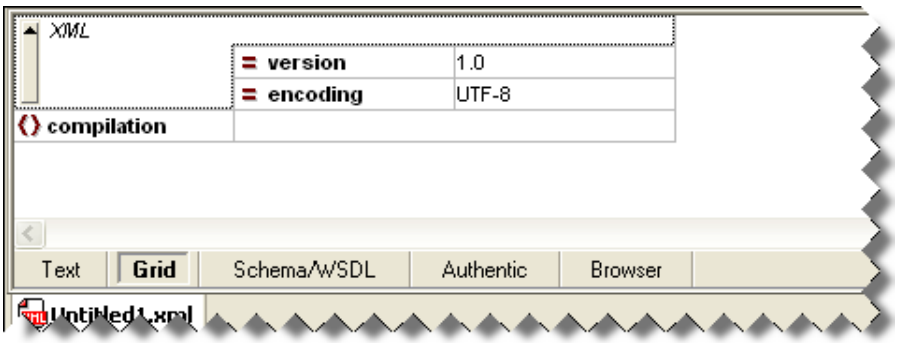


À ce stade de prise en main, déclinez par **Cancel** la boîte de dialogue proposée par XML Spy permettant d'inclure un DTD existant.

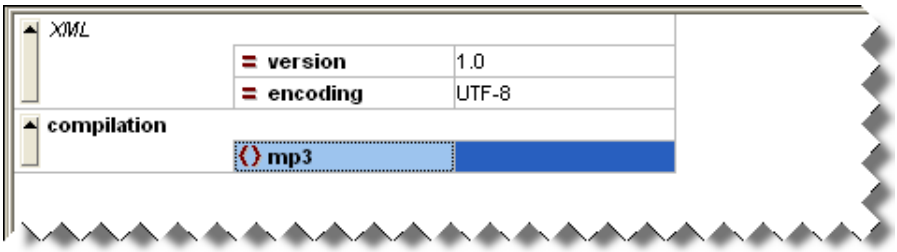
Dans la sous-fenêtre réservée à l'édition, encodez la balise racine `<compilation>`.



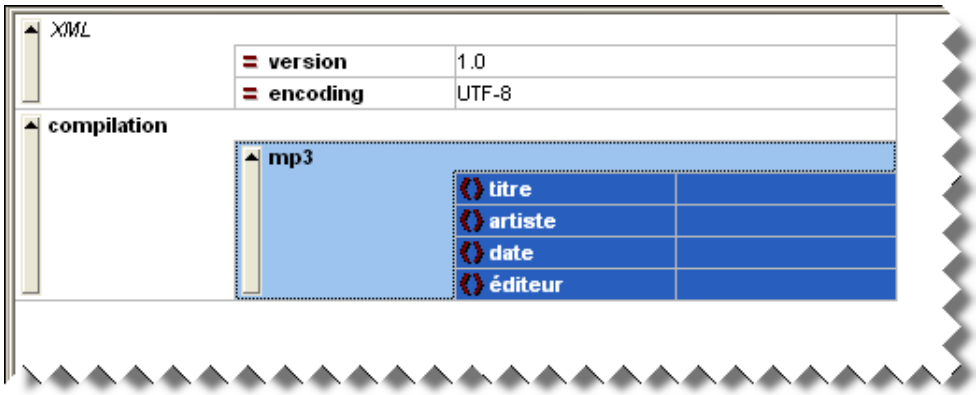
Vous remarquez au bas de la fenêtre d'édition, les boutons **Text, Grid, Schema/ WSDL, Authentic, Browser**. Cliquez sur le bouton **Grid** pour passer en mode de création graphique, ce qui est plus convivial.



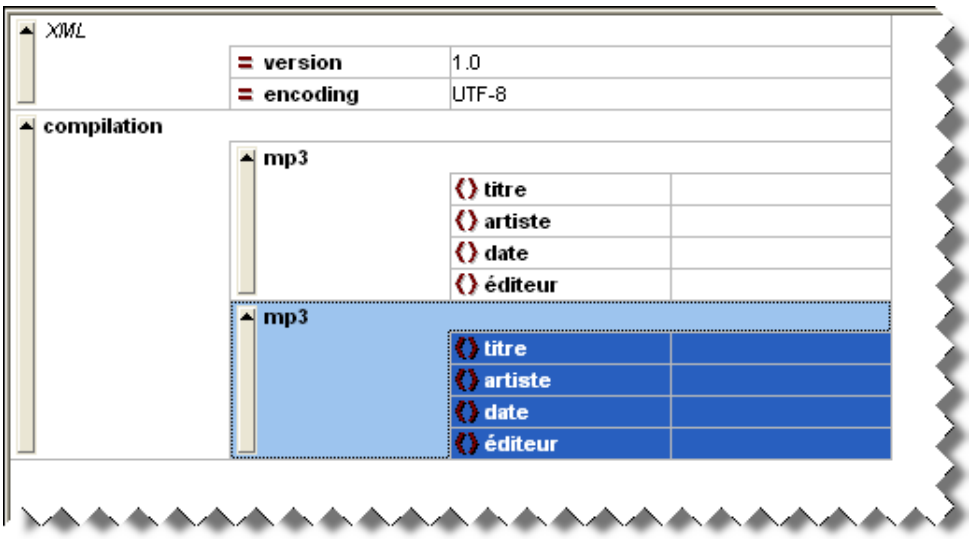
- Encodons à présent, l'élément enfant **mp3**. Après avoir cliqué dans la zone **compilation**, menu **XML - Add child - Element**, encodez la balise **mp3**. La même opération peut être réalisée à partir du bouton droit.



- Les éléments enfant de mp3 peuvent à présent être enregistrés. Après avoir sélectionné la zone **mp3**, construisez quatre éléments soit **XML - Add child - Element** opération que l'on répète quatre fois et le nom des balises **titre, artiste, date et editeur** est encodé.

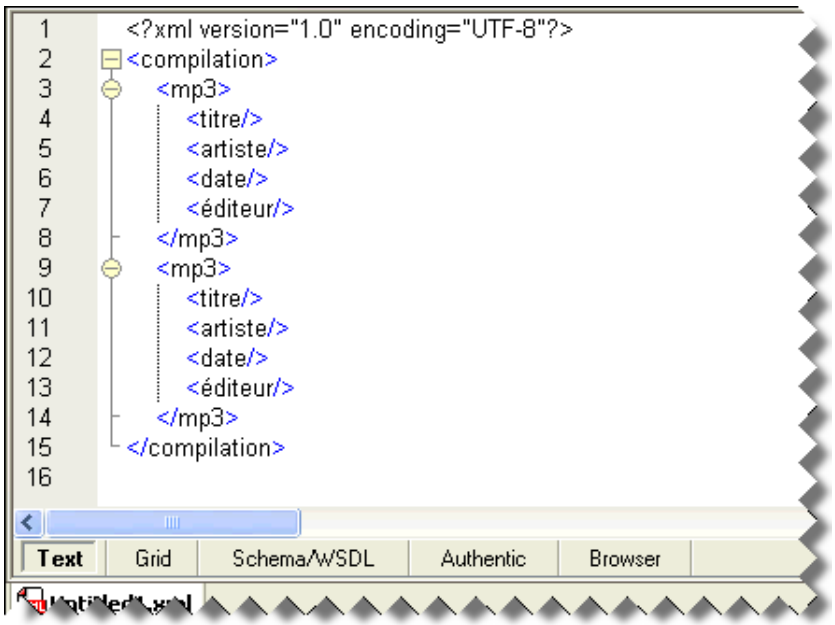


- Pour reproduire un second bloc **mp3**, sélectionnez la zone mp3, **Edit - Copy** puis **Edit - Paste**.

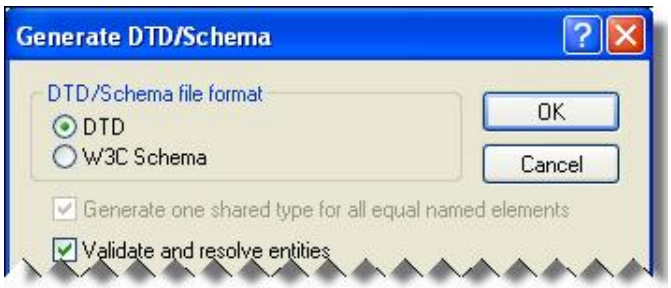


Il ne reste plus qu'à encoder les données.

Il est possible de revenir à tout moment aux fichiers XML en cliquant sur le bouton **Text** pour visualiser le fichier sous la forme suivante :



- Pour élaborer le DTD du document, c'est facile avec XMLSpy, Menu **DTD/ Schema - Generate DTD/Schema**.



Et le DTD apparaît !

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD generated by XMLSpy v2007 sp1 (http://www.altova.com)-->
<!ELEMENT compilation ((mp3+))>
<!ELEMENT mp3 ((titre, artiste, date, éditeur))>
<!ELEMENT titre (#PCDATA)>
```

```
<!ELEMENT artiste (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT éditeur (#PCDATA)>
```

Ceci n’est vraiment qu’un très bref aperçu de XML Spy. Ses fonctionnalités sont en effet plus nombreuses et étendues.

Pour compléter l’étude du XML, l’auteur vous recommande l’ouvrage XML par la pratique - bases indispensables, concepts et cas pratiques de Sébastien Lecomte dans la collection Ressources Informatiques des Editions ENI.

Le XSL

Le XSL pour *eXtensible Stylesheet Language* (langage extensible de feuilles de style) est une recommandation du consortium W3C datant de novembre 1999. Il s'agit d'un standard dans le domaine de la publication sur le Web. Le XSL est en quelque sorte le langage de feuilles de style du XML. Un fichier de feuilles de style XSL reprend des données XML et produit la présentation ou l'affichage de ce contenu XML selon les souhaits du concepteur de la page web.

Le XSL comporte en fait 3 langages :

- Le XSL**T**(*XSL Transform*), langage qui **T**ransforme un document XML en un format, généralement en Html ou Xhtml, reconnu par un navigateur.
- Le Xpath langage qui permet de définir et d'adresser des parties de document XML.
- Le XML Formatter langage permettant de "formater" ou transformer du XML de façon à le rendre compatible avec des organismes PDA ou des unités de reconnaissance vocale.

Pour la suite de ce tutorial, nous nous limitons aux langages XMLT et Xpath. Et comme dans la littérature relative à ce sujet, ils sont identifiés sous le vocable général de XSL.

Le XSL est dérivé du XML, reprenant ainsi toutes les règles de syntaxe du XML (détaillée au chapitre Introduction au XML).

Soit en bref :

- les balises sensibles à la casse, s'écrivent en minuscules.
- toutes les balises ouvertes doivent être impérativement fermées.
- les balises vides auront aussi un signe de fermeture soit `<balise/>`.
- les balises doivent être correctement imbriquées.
- les valeurs des attributs doivent toujours être mises entre des guillemets.
- le document XSL devra être "bien formé".

Le XSL ne permet pas uniquement l'affichage de XML. Il permet aussi :

- de sélectionner une partie des éléments XML.
- de trier des éléments XML.
- de filtrer des éléments XML en fonction de certains critères.
- de choisir des éléments.
- de retenir des éléments par des tests conditionnels.
- de transformer un document.

Un premier document XSL

Avant de débuter, il est utile de préciser que :

- le XSL est dérivé du XML. Le document XSL reprend donc la structure et la syntaxe de n'importe quel document XML.
- le document XSL comporte un document Html ou Xhtml qui sera quant à lui reconnu par le navigateur et qui servira de support à tout ou partie des données du document XML associé.
- le XSL fonctionne avec une ou plusieurs **templates**, sorte de gabarit, pour définir comment afficher des éléments du fichier XML. Les éléments concernés du fichier XML sont déterminés par l'attribut *match*.

Voici un premier document XSL. Sa conception est basique mais n'est pas compliquée ; ce document sera étoffé en cours d'étude.

```
<?xml version="1.0"?>
```

Le XSL est dérivé du XML. Aussi, il est normal que le document XSL commence par la déclaration de document XML, soit <?xml version="1.0"?>.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Cette ligne déclare que le document est au format XSL.

L'attribut *xmlns* fait référence à l'espace de nommage (*namespace*) utilisé. Le *namespace* officiel du W3C est : xmlns:xsl="http://www.w3.org/1999/XSL/ Transform.

```
<xsl:template match="/">
```

Voici une balise *template* et son attribut *match*.

Cette balise *template* va déterminer un gabarit dans lequel des éléments du fichier XML sont transformés sous une forme affichable par le navigateur.

Les éléments du fichier XML sont déterminés par l'attribut `match="/"`. La barre oblique ("/") entre guillemets signale que toutes les balises XML du document associé à partir de la racine sont concernées.

```
<html>
<head>
<title>XSL</title>
</head>
<body>
```

Il s'agit du début de la partie Html ou Xhtml servant de support pour l'affichage du document dans le navigateur. Attention, les balises doivent être écrites en minuscules !

```
Diverses balises Xhtml et XSL...
```

La partie Html ou Xhtml du document.

```
</body>
</html>
```

Fin de la partie en Html ou Xhtml.

```
</xsl:template>
```

La fermeture de la balise de template.

```
</xsl:stylesheet>
```

Le document XSL se termine obligatoirement par la fermeture de la balise de déclaration de document XSL.

Attention ! Pour que ce fichier XSL soit d'une quelconque utilité, il ne faut pas oublier de faire référence à celui-ci dans le fichier XML.

Ainsi cette ligne doit être ajoutée dans le fichier XML :

```
<?xml-stylesheet type="text/xsl" href="nom_du_fichier_xsl.xsl"?>
```

Cette balise indique au navigateur qu'une feuille de style (*stylesheet*) de type XSL est associée au fichier XML et qu'il doit aller chercher le fichier à l'adresse indiquée par l'attribut *href*.

Voici notre premier document XSL :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
Diverses balises Xhtml et XSL...
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

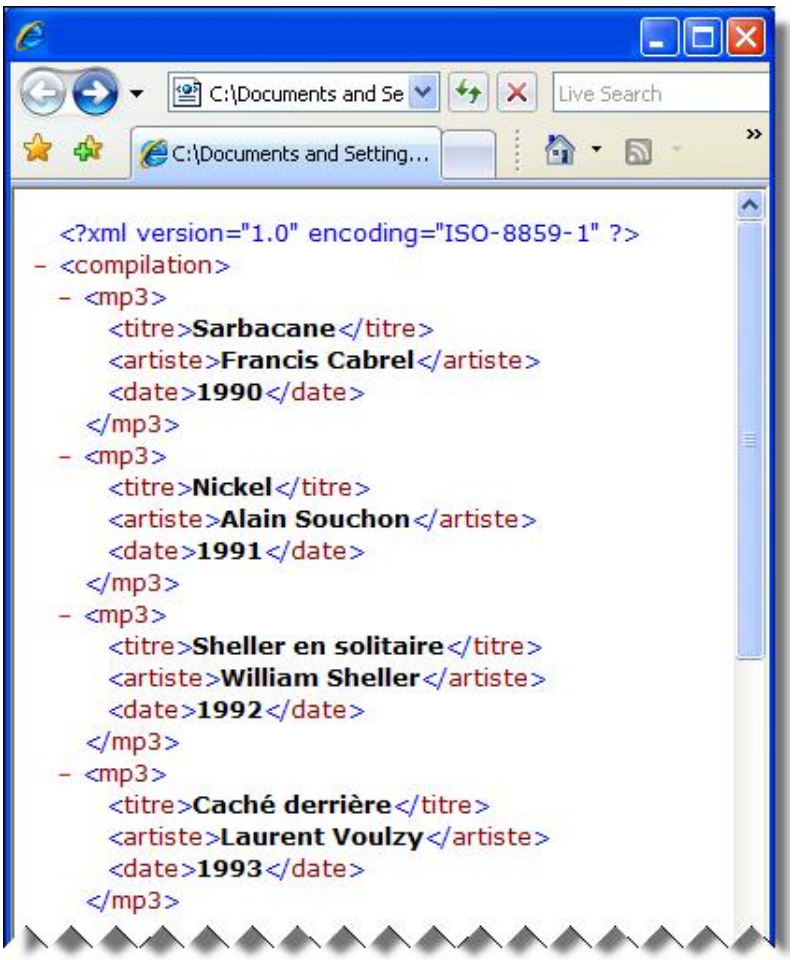
Un premier exemple XSL

Après cet aperçu théorique, étudions un exemple détaillé : une compilation de composition musicale MP3.

Voici un fichier XML que nous allons utiliser tout au long de ce chapitre.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
</mp3>
<mp3>
<titre>Sheller en solitaire</titre>
<artiste>William Sheller</artiste>
<date>1992</date>
</mp3>
<mp3>
<titre>Caché derrière</titre>
<artiste>Laurent Voulzy</artiste>
<date>1993</date>
</mp3>
<mp3>
<titre>Rio Grande</titre>
<artiste>Eddy Mitchell</artiste>
<date>1994</date>
</mp3>
<mp3>
<titre>Samedi soir sur la Terre</titre>
<artiste>Francis Cabrel</artiste>
<date>1995</date>
</mp3>
<mp3>
<titre>Défoule sentimentale</titre>
<artiste>Alain Souchon</artiste>
<date>1996</date>
</mp3>
</compilation>
```

On l'enregistre sous le nom xsldemo avec une extension .xml (soit xsldemo.xml) dont voici un aperçu dans Internet Explorer :



Passons maintenant au fichier XSL.

Le but de l'exemple est de représenter la compilation sous forme d'un tableau de trois colonnes.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
<h3>Les victoires de la musique</h3>
<table border="1" style="border-collapse: collapse"
bordercolor="#000000" cellpadding="3">
<tr style="background: #9cf;">
<td>Année</td>
<td>Album</td>
<td>Artiste</td>
</tr>
<tr>
<td><xsl:value-of select="compilation/mp3/date"/></td>
<td><xsl:value-of select="compilation/mp3/titre"/></td>
<td><xsl:value-of select="compilation/mp3/artiste"/></td>
</tr>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Après les balises de départ d'un fichier XSL, on élabore un tableau tout à fait classique en Xhtml. Dans la première cellule, une information de date est renseignée, soit la balise `<xsl:value-of />` avec

l'attributselect="compilation/mp3/date" qui indique comme chemin d'accès dans le fichier XML la balise racine **compilation**, la balise **mp3** et la balise **date**. Dans la deuxième cellule, une information de titre est renseignée, soit la balise <xsl:value-of /> avec l'attribut select="compilation/mp3/titre" qui indique comme chemin d'accès dans le fichier XML, la balise racine **compilation**, la balise **mp3** et la balise **titre**. Enfin la dernière cellule est consacrée à l'information relative à l'artiste avec l'attribut select="compilation/mp3/artiste".

Le fichier est enregistré sous le nom **xsldemo** avec une extension **.xsl** (xsldemo.xsl).

Afin d'associer ce fichier XSL au fichier XML précédent, la balise suivante est ajoutée :

```
<?xml-stylesheet type="text/xsl" href="xsldemo.xsl"?>
```

Le code complet de ce dernier devient :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="xsldemo.xsl"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
</mp3>
Etc.
```

Et une fois affiché dans un navigateur, notre fichier XML est devenu un tableau plus présentable.

Par contre, une seule référence de la compilation est affichée dans le tableau. Pour afficher toutes les références, procéder comme suit :

Le fichier XML de départ reste inchangé.

Considérons le fichier XSL dans lequel la balise<xsl:for-each /> (for-each signifie pour chaque) avec comme attribut select="compilation/mp3" va être ajoutée. Dans le fichier XML de balises, une ligne de tableau (<*tr*>) est insérée, contenant des cellules <*td*> dont voici le contenu :

- balise **date** soit <xsl:value-of select="date" /> pour la première cellule ;
- balise **titre** soit <xsl:value-of select="titre" /> pour la seconde cellule ;
- balise **artiste** soit <xsl:value-of select="artiste" /> pour la troisième cellule.

Ce qui donne :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
```

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
<h3>Les victoires de la musique</h3>
<table border="1" style="border-collapse: collapse"
bordercolor="#000000" cellpadding="3">
<tr style="background: #9cf;">
<td>Année</td>
<td>Album</td>
<td>Artiste</td>
</tr>
<xsl:for-each select="compilation/mp3">
<tr>
<td><xsl:value-of select="date"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Le fichier est enregistré avec l'extension **.xsl** (xsldemo1.xsl).

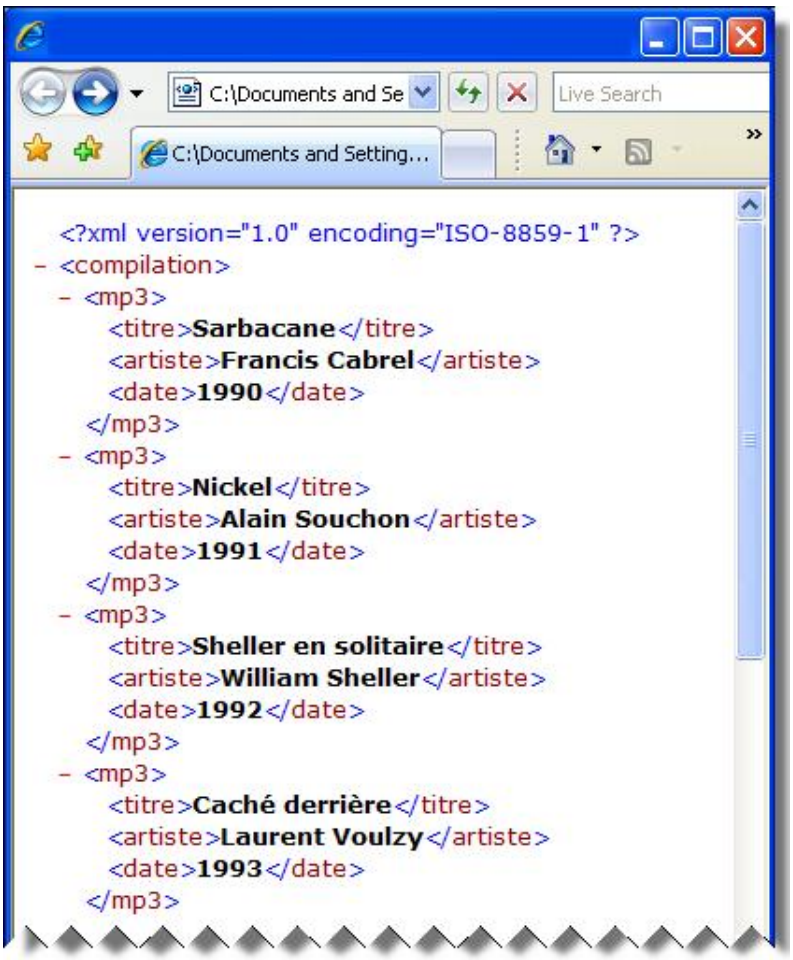
Il ne faut pas oublier de modifier le lien dans le fichier XML. Soit :

```
<?xml-stylesheet type="text/xsl" href="xsldemo1.xsl"?>
```

Le code complet du fichier XML devient :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="xsldemo1.xsl"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
</mp3>
Etc.
```

Pour rappel, le fichier XML sans le fichier XSL prendrait la forme suivante.



Il ne reste plus qu'à vérifier que le fichier affiche bien toutes les références de la compilation.

Trier avec le langage XSL

Le langage XSL permet de trier des données d'un fichier XML associé, en ordre croissant ou décroissant. Ainsi, il suffit d'ajouter la balise `<xsl:sort select="..."/>` et l'attribut `order="ascending"` pour trier des données en ordre croissant ou `order="descending"` pour trier en ordre décroissant.

Cette concision illustre bien la puissance du langage XSL.

Exemple

Reprenons notre fichier XML de départ (inchangé).

Dans le fichier XSL, nous allons trier notre compilation de mp3 en XML en ordre alphabétique croissant du nom des artistes. En outre, nous allons permuter les colonnes "Année" et "Artiste" pour bien montrer que le XSL affiche les données du fichier XML selon le fichier Html ou Xhtml qu'il contient.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
<table border="1" style="border-collapse: collapse"
bordercolor="#000000" cellpadding="3">
<tr style="background: #9cf;">
<td>Artiste</td>
<td>Titre</td>
<td>Année</td>
</tr>
<xsl:for-each select="compilation/mp3">
<xsl:sort select="artiste" order="ascending" />
<tr>
<td><xsl:value-of select="artiste"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="date"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Le fichier est enregistré sous le nom xsl_order avec une extension .xsl (xsl_order. xsl).

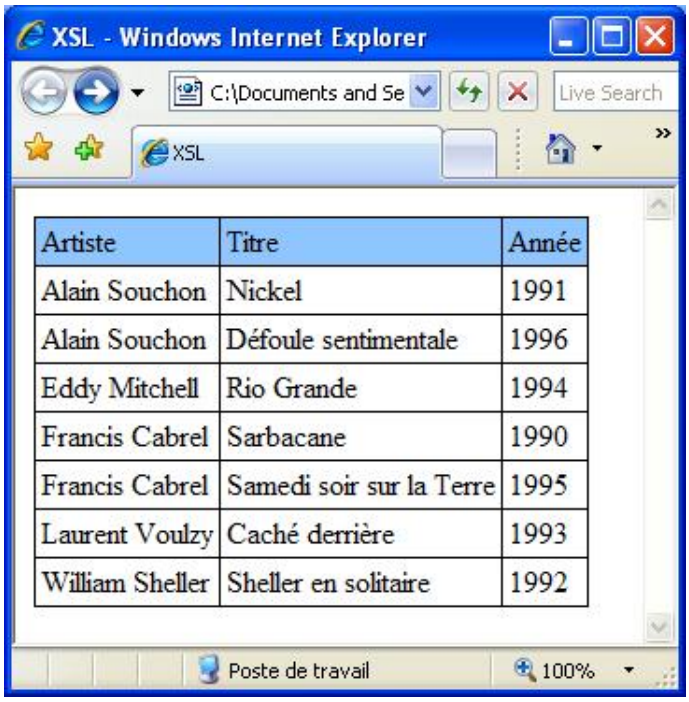
Afin d'associer ce fichier XSL au fichier XML, la balise suivante est ajoutée :

```
<?xml-stylesheet type="text/xsl" href="xsl_order.xsl"?>
```

Soit,

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="xsl_order.xsl"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
</mp3>
```

```
Etc
```



Pour trier la compilation par ordre alphabétique inverse, il suffit de modifier l'attribut `order="descending"`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
<table border="1" style="border-collapse: collapse"
bordercolor="#000000" cellpadding="3">
<tr style="background: #9cf;">
<td>Artiste</td>
<td>Titre</td>
<td>Année</td>
</tr>
<xsl:for-each select="compilation/mp3">
<xsl:sort select="artiste" order="descending" />
<tr>
<td><xsl:value-of select="artiste"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="date"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Le fichier est enregistré sous le nom xsl_orderbis avec une extension .xsl (xsl_orderbis.xsl).

Afin d'associer ce fichier XSL au fichier XML, la balise suivante est ajoutée :

```
<?xml-stylesheet type="text/xsl" href="xsl_orderbis.xsl"?>
```

Soit,

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="xsl_orderbis.xsl"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
</mp3>
Etc.
```

Filtrer avec le XSL

Le langage XSL permet aussi de filtrer les données du fichier XML associé selon des critères tels que égal, différent de, plus grand que, plus petit que.

Il suffit d'utiliser l'attribut `select="chemin_d'accès[balise='xxx']"`.

Les opérateurs possibles sont :

- = pour égal.
- != pour différent (non égal).
- > pour plus grand que.
- < pour plus petit que.

Ceci vous paraît un peu abstrait ? Voyons un exemple de filtre de données...

Dans la compilation mp3, ne reprenons que le (ou les) titre(s) de l'artiste Alain Souchon.

L'attribut select du fichier XSL devient `select="compilation/mp3[artiste= 'Alain Souchon']"`.

Le fichier XML reste toujours inchangé.

Passons maintenant au fichier XSL.

Nous allons reprendre dans notre compilation de mp3 en XML que le (ou les) titre(s) d'Alain Souchon.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
<h3>Les victoires de la musique</h3>
<table border="1" style="border-collapse: collapse"
bordercolor="#000000" cellpadding="3">
<tr style="background: #9cf;">
<td>Année</td>
<td>Album</td>
<td>Artiste</td>
</tr>
<xsl:for-each select="compilation/mp3[artiste='Alain Souchon']">
<tr>
<td><xsl:value-of select="date"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Le fichier est enregistré sous le nom xsl_filter avec l'extension .xsl (xsl_filter.xsl).

Afin d'associer ce fichier XSL au fichier XML, la balise suivante est ajoutée :

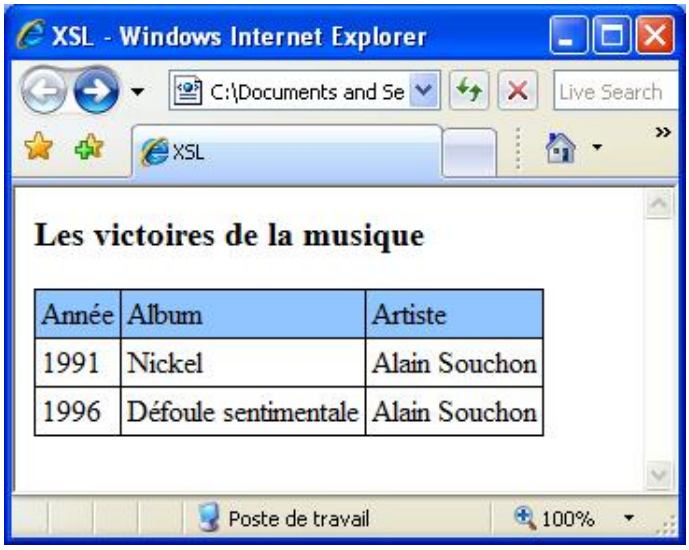
```
<?xml-stylesheet type="text/xsl" href="xsl_filter.xsl"?>
```

Soit,

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<?xml-stylesheet type="text/xsl" href="xsl_filter.xsl"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
</mp3>
Etc.
```

Ce qui nous permet d'afficher uniquement les albums d'Alain Souchon.



Choisir avec le XSL

La balise <xsl:if> ... </xsl:if> permet d'effectuer un choix dans les données du fichier XML. L'attribut match est ajouté pour indiquer l'élément choisi. Ce qui donne :

```
<xsl:if test="balise='xxx'">
balises Html
</xsl:if>
```

Nous allons illustrer ce choix par un exemple : ne faire apparaître que le(s) titre(s) de Voulzy dans le fichier en XML de compilation de mp3.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
<h3>Les victoires de la musique</h3>
<table border="1" style="border-collapse: collapse"
bordercolor="#000000" cellpadding="3">
<tr style="background: #9cf;">
<td>Année</td>
<td>Album</td>
<td>Artiste</td>
</tr>
<xsl:for-each select="compilation/mp3">
<xsl:if test="artiste='Laurent Voulzy'">
<tr>
<td><xsl:value-of select="date"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:if>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Le fichier est enregistré sous le nom xsl_if avec l'extension .xsl (xsl_if.xsl).

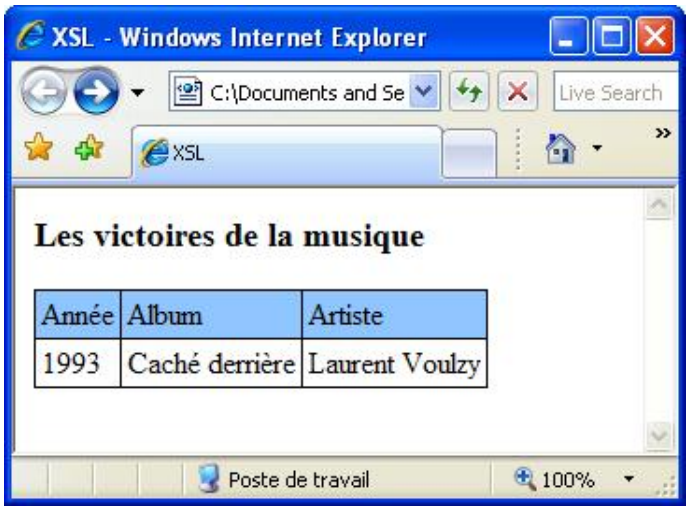
Afin d'associer ce fichier XSL au fichier XML de départ, la balise suivante est ajoutée :

```
<?xml-stylesheet type="text/xsl" href="xsl_if.xsl"?>
```

Le fichier XML complet devient :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="xsl_if.xsl"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
</mp3>
Etc.
```

Ce qui nous permet d'afficher uniquement la ligne du tableau correspondant à Laurent Voulzy.



Voici une autre illustration de cette possibilité de choix : ne retenir dans notre compilation de mp3 en XML que les informations postérieures à 1993.

Il suffit de modifier la balise `<xsl:if>` du fichier XSL précédent. La condition s'exprime par `test="date &gt; 1993"` (> pour le signe > supérieur à).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
<h3>Les victoires de la musique</h3>
<table border="1" style="border-collapse: collapse"
bordercolor="#000000" cellpadding="3">
<tr style="background: #9cf;">
<td>Année</td>
<td>Album</td>
<td>Artiste</td>
</tr>
<xsl:for-each select="compilation/mp3">
<xsl:if test="date &gt; 1993">
<tr>
<td><xsl:value-of select="date"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:if>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Le fichier est enregistré sous le nom xsl_ifbis avec l'extension .xsl (xsl_ifbis.xsl).

Afin d'associer ce fichier XSL au fichier XML de départ, la balise suivante est ajoutée :

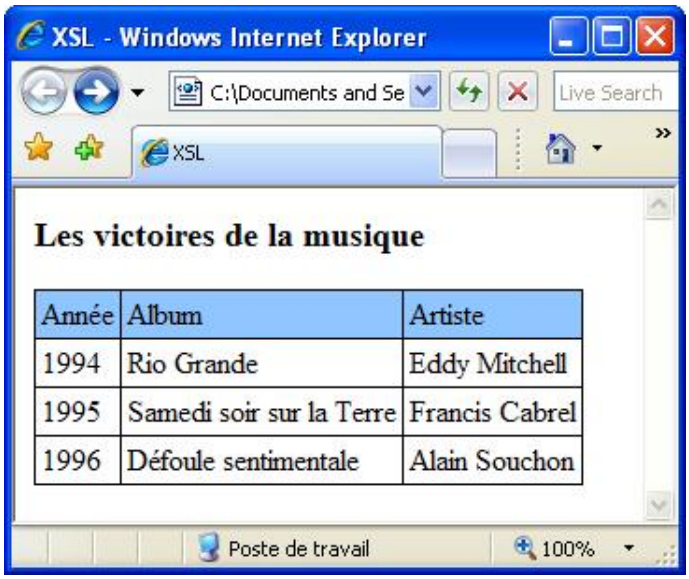
```
<?xml-stylesheet type="text/xsl" href="xsl_ifbis.xsl"?>
```

Soit,

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="xsl_ifbis.xsl"?>
<compilation>
```

```
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
</mp3>
Etc.
```

Ce qui nous permet d'afficher uniquement les lignes du tableau pour les années postérieures à 1993.



Conditions et le XSL

Le XSL permet aussi de faire un choix conditionnel par la balise <xml:choose>. À l'intérieur de cette balise, une action peut être déterminée lorsqu'une condition est vérifiée (balise <xsl:when>) et dans le cas contraire il faut prévoir une autre action (balisexsl:otherwise>).

```
<xsl:choose>
```

Condition vérifiée

```
<xsl:when test="artiste='Francis Cabrel'">
<tr style="background: #9cf;">
<td><xsl:value-of select="date"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:when>
```

sinon

```
<xsl:otherwise>
<tr>
<td><xsl:value-of select="date"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:otherwise>

</xsl:choose>
```

Exemple

Nous allons reprendre dans notre compilation de mp3 en XML tous les titres de Francis Cabrel que nous allons afficher avec un arrière-plan de couleur, les autres titre étant affichés normalement.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
<h3>Les victoires de la musique</h3>
<table border="1" style="border-collapse: collapse"
bordercolor="#000000" cellpadding="3">
<tr>
<td>Année</td>
<td>Album</td>
<td>Artiste</td>
</tr>
<xsl:for-each select="compilation/mp3">
<xsl:choose>
<xsl:when test="artiste='Francis Cabrel'">
<tr style="background: #9cf;">
<td><xsl:value-of select="date"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:when>
<xsl:otherwise>
<tr>
<td><xsl:value-of select="date"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
```

```
</xsl:otherwise>
</xsl:choose>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Le fichier est enregistré sous le nom xsl_choose et l'extension .xsl (xsl_choose.xsl).

Afin d'associer ce fichier XSL au fichier XML de départ, la balise suivante est ajoutée :

```
<?xml-stylesheet type="text/xsl" href="xsl_choose.xsl"?>
```

Soit,

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="xsl_choose.xsl"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
</mp3>
Etc.
```

Ce qui nous permet d'afficher uniquement les lignes concernant Francis Cabrel mises en évidence.

Transformation avec le XSL

L'élément `<xsl:apply-templates />` utilisé au sein de la balise `<xsl: template/>` permet d'appliquer un modèle (*template*) de transformation ou de présentation sur les éléments du document XML.

Sur notre fichier XML, nous allons appliquer des présentations de style différentes pour les informations relatives :

- au titre de l'album `<xsl:template match="titre"> ... </xsl:template>`,
- au nom de l'artiste `<xsl:template match="artiste"> ... </xsl:template>`,
- et à la date `<xsl:template match="date"> ... </xsl:template>`.

Les balises concernées ont été préalablement définies par :

```
<xsl:template match="mp3">
<p>
<xsl:apply-templates select="titre"/>
<xsl:apply-templates select="artiste"/>
<xsl:apply-templates select="date"/>
</p>
</xsl:template>
```

Le fichier XSL devient :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
<h3>Les victoires de la musique</h3>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
<xsl:template match="compilation/mp3">
<p>
<xsl:apply-templates select="titre"/>
<xsl:apply-templates select="artiste"/>
<xsl:apply-templates select="date"/>
</p>
</xsl:template>
<xsl:template match="titre">
Titre : <span style="font-weight:bold;">
<xsl:value-of select="."/></span>
<br />
</xsl:template>
<xsl:template match="artiste">
Artiste : <span style="font-variant:small-caps">
<xsl:value-of select="."/></span>
<br />
</xsl:template>
<xsl:template match="date">
Date: <span style="font-style:italic;">
<xsl:value-of select="."/></span>
<br />
</xsl:template>
</xsl:stylesheet>
```

Le fichier est enregistré sous le nom xsl_apply et l'extension .xsl (xsl_apply.xsl).

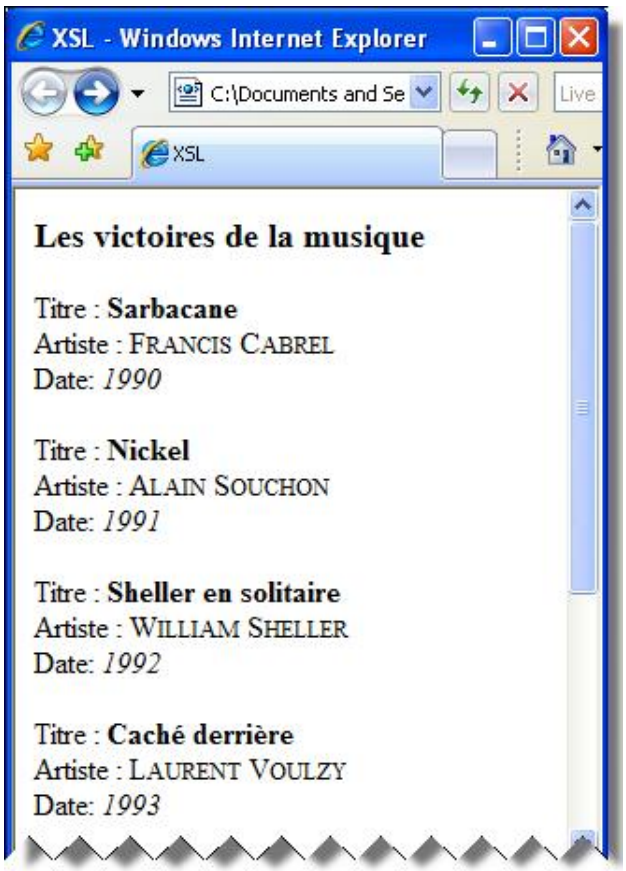
Afin d'associer ce fichier XSL au fichier XML de départ, la balise suivante est ajoutée :

```
<?xml-stylesheet type="text/xsl" href="xsl_apply.xsl"?>
```

Ce dernier devient :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="xsl_apply.xsl"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
</mp3>
Etc.
```

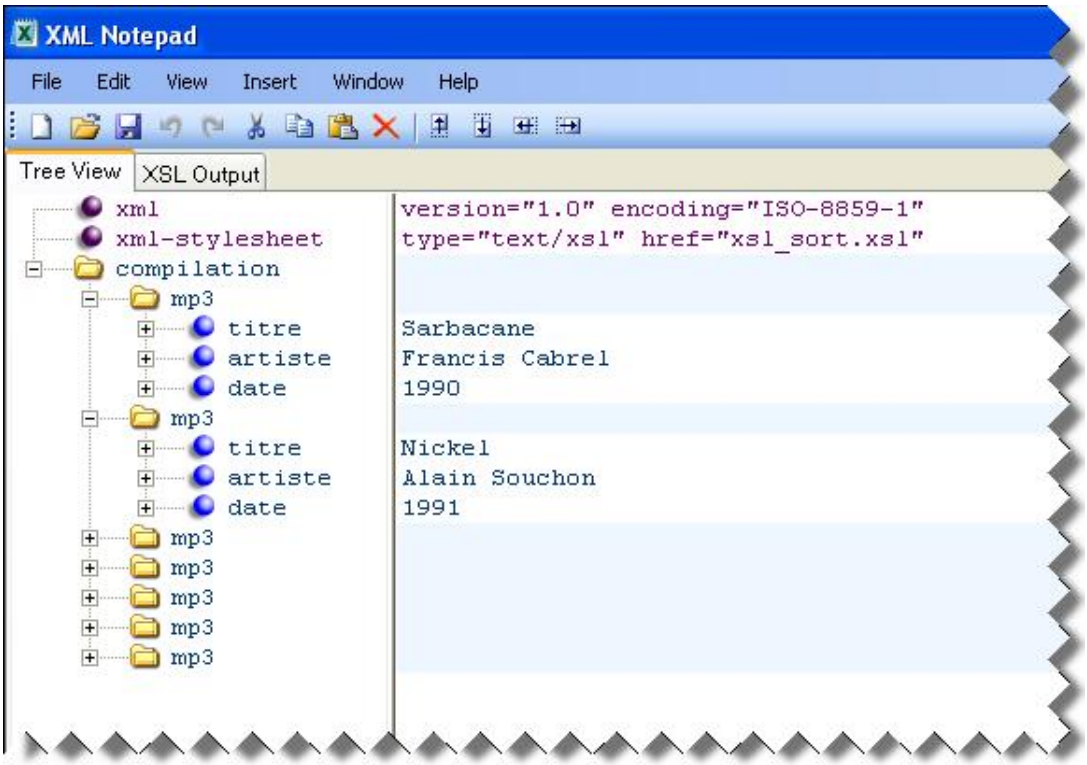
Ce qui permet d'afficher notre fichier avec une mise en forme différente pour le titre (en gras), le nom de l'artiste (petites majuscules) et le date (en italique).



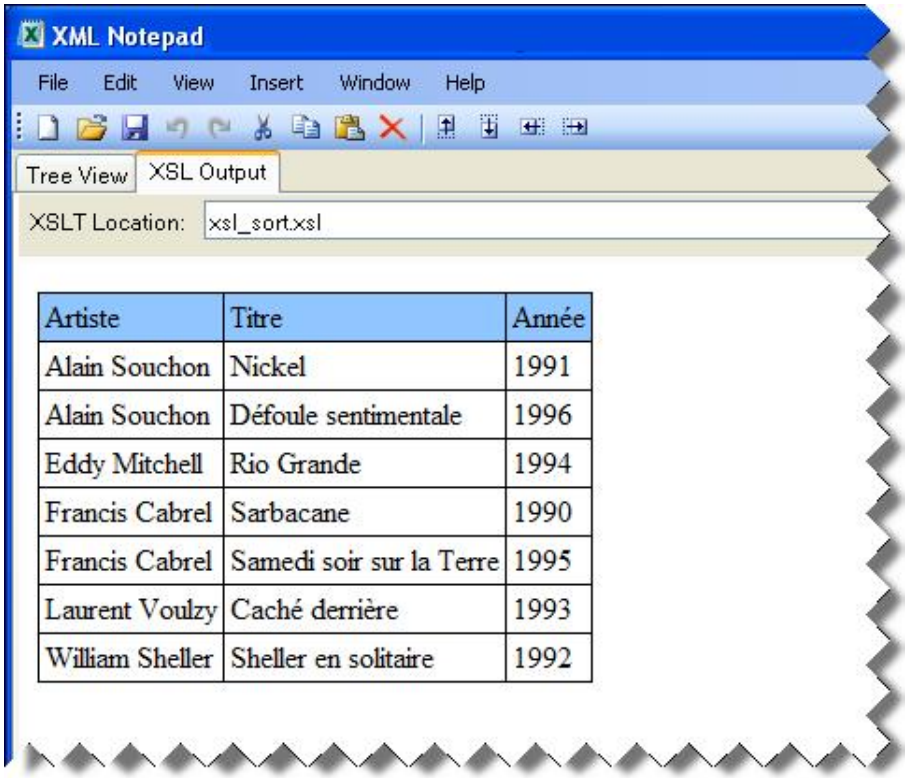
Pour une étude (beaucoup) plus complète du XSL, vous pouvez vous reporter à l'ouvrage **XML et XSL - Les feuilles de style XML** de Cyril Vincent de la collection Ressources Informatiques des Editions ENI.

XML Notepad 2007 et le XSL

Si vous utilisez XML Notepad 2007, lorsque vous ouvrez un fichier (**File - Open**), celui-ci s'affiche dans la fenêtre correspondant à l'onglet **Tree View**.



En choisissant l'onglet **XSL Output**, il est possible d'adjoindre au fichier XML un fichier XSL et de visualiser la transformation graphique instantanément, sans avoir à utiliser un navigateur.



Artiste	Titre	Année
Alain Souchon	Nickel	1991
Alain Souchon	Défoule sentimentale	1996
Eddy Mitchell	Rio Grande	1994
Francis Cabrel	Sarbacane	1990
Francis Cabrel	Samedi soir sur la Terre	1995
Laurent Voulzy	Caché derrière	1993
William Sheller	Sheller en solitaire	1992

Présentation

Le DOM (*pour Document Object Model*)<$SDocument Object Model;DOM, Nœud> définit un mode standardisé pour accéder et mettre à jour tous les éléments d'un document Html, Xhtml ou XML.

Il faut préciser que le DOM n'est pas en soi un langage de balise ou de programmation mais simplement une manière de percevoir, de parcourir et de manipuler un document Html ou XML, en utilisant des méthodes et des propriétés spécifiques.

Lors des balbutiements de la publication sur le Web, Netscape 2 et Internet Explorer 3 avaient déjà introduit un concept objet dans les pages Html. Ce concept rudimentaire permettait d'accéder à certains éléments du document Html, comme les images et les composants de formulaires. Il portera, a posteriori, le nom de DOM niveau 0 (*DOM level 0*). Comme c'était la mauvaise pratique à l'époque, chaque éditeur avait sa propre implémentation des objets, ce qui laissait la porte ouverte à une forte incompatibilité entre les différents navigateurs.

Le consortium W3C s'empressa d'éditer dès 1998 une recommandation pour standardiser les objets et la façon de les appréhender. Celle-ci porte le nom de DOM niveau 1 (*Dom level 1*). Il devenait ainsi possible d'accéder à chaque élément de la page Html, du document entier au simple contenu textuel. Mieux encore, tous les fabricants de navigateurs ont, petit à petit, adopté cette recommandation, avec pour conséquence la disparition quasi-totale des problèmes d'incompatibilité du DOM.

Pour être complet, signalons que le DOM a évolué avec la recommandation DOM niveau 2 en novembre 2000 et la recommandation DOM niveau 3 en janvier 2004. Les spécifications de ces dernières recommandations ne sont, à l'heure actuelle, que partiellement reprises par les navigateurs.

Concept de nœud (node)

Selon le DOM, tout composant ou élément d'un document Html, Xhtml ou XML constitue un nœud (*node*).

Ces différents nœuds sont en relation les uns avec les autres. Il est alors possible de représenter un document selon une structure arborescente qui n'est pas sans rappeler celle d'un arbre généalogique avec ses branches et ses intersections.

Il faut bien admettre que ce terme de nœud ou *node* n'est pas très élégant ni très explicite. On aurait pu parler d'élément mais ce terme désigne également les balises Html et Xhtml. Le terme d'objet est, quant à lui, repris par le JavaScript et les autres langages de programmation. Par contre, il a néanmoins l'avantage de rappeler l'arborescence ou la hiérarchisation d'un document.

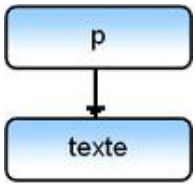
Tout composant étant un nœud, on peut concevoir :

- le document entier comme un nœud, appelé nœud document (*document node*).
- Chaque balise ou élément est un nœud élément (*element node*). Ainsi, des balises comme `<p>`, `<ul>`, `<li>` sont des nœuds élément.
- Le texte contenu entre les balises est un nœud texte (*text node*). Ainsi, pour `<p>Texte du paragraphe</p>`, les mots "Texte du paragraphe" constituent un nœud texte.
- Chaque attribut d'une balise ou élément est un nœud attribut (*attribut node*). Pour `<img href="image.png" alt="" />`, *href* et *alt* sont des nœuds attribut.
- Les commentaires deviennent des nœuds de commentaire (*comment node*).
- Etc.

Hiérarchisation des nœuds

Les nœuds ont une relation hiérarchique entre eux.

Soit, par exemple, `<p>Texte du Paragraphe</p>`. Cet exemple contient deux nœuds. Un nœud élément pour la balise <p> et un nœud texte qui comporte les mots "Texte du paragraphe". Comme le nœud <p> contient un nœud texte, leur relation peut être qualifiée de parent/enfant, la balise <p> jouant le rôle de parent (*parent*) et le texte de celui d'enfant (*child*).

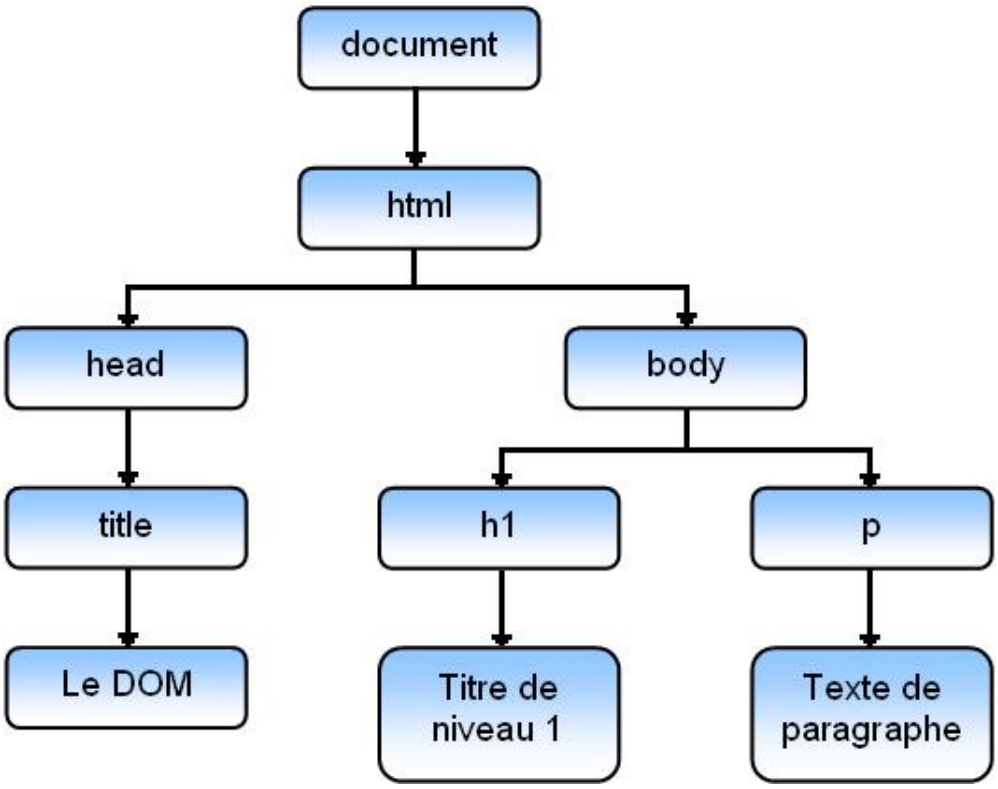


Selon ce mode hiérarchique, tous les nœuds forment l'arborescence ou l'arbre du document. Cet arbre commence par le nœud document et se termine par les nœuds texte pour les niveaux les plus éloignés.

Prenons à titre d'exemple, un document Html rudimentaire.

```
<html>
<head>
<title>Le DOM</title>
</head>
<body>
<h1>Titre de niveau 1</h1>
<p>Texte du paragraphe</p>
</body>
</html>
```

La hiérarchie du document peut se représenter comme suit :



Document, html, head, title, Le DOM, body, h1, Titre de niveau 1, p, Texte du paragraphe sont tous des nœuds. Mais ils ne sont pas tous du même type. Document est un nœud document. Les six balises Html sont des nœuds élément et les trois textes sont des nœuds texte.

Tous ces nœuds sont en relation les uns avec les autres. Pour qualifier ces relations, le DOM emprunte la terminologie utilisée pour les arbres généalogiques.

Chaque nœud, excepté le nœud document, a un nœud parent. Ainsi, par exemple, le nœud parent du nœud élément <body> est le nœud élément <html> et le nœud parent de <h1> est le nœud élément <body>.

La plupart des nœuds élément ont un ou des nœuds enfant. Ainsi, le nœud <head> possède un élément enfant, le nœud <title>. À son tour, le nœud <title> comporte un élément enfant, soit le nœud texte "Le DOM".

Certains nœuds sont frères et sœurs (*siblings*) car ils partagent le même parent. Ainsi, les nœuds <h1> et <p> sont frères et sœurs car ils sont tous deux des éléments enfant du nœud élément <body>.

Des nœuds peuvent avoir des ascendants et des descendants.

Les ascendants sont les nœuds qui sont parents d'un nœud ou parents d'un nœud parent. Dans notre exemple, tous les nœuds de texte ont le nœud <html> comme ascendant et le nœud de texte de <p> a le nœud élément body comme ascendant.

Les descendants sont les nœuds qui sont enfants d'un nœud ou enfants d'un nœud enfant. Dans notre exemple, tous les nœuds texte sont des descendants du nœud <html> et le nœud de texte de <p> est descendant du nœud <body>.

Propriétés de l'objet Node

1. Les propriétés de relation

Il n'est pas étonnant de retrouver les termes de parent (*parent*), enfant (*child*) ou frères/sœurs (*sibling*) dans les propriétés.

`parentNode`	Renvoie le nœud parent d'un nœud.
`firstChild`	Renvoie le premier enfant d'un nœud.
`lastChild`	Renvoie le dernier enfant d'un nœud.
`childNodes`	Stocke une liste de tous les nœuds enfant disponibles à partir d'un nœud.
`previousSibling`	Renvoie le nœud frères/sœurs précédent d'un nœud.
`nextSibling`	Renvoie le nœud frères/sœurs suivant d'un nœud.

2. Les propriétés d'état

Ces propriétés permettent de s'informer sur l'état d'un nœud.

`nodeName`

Indique le nom du nœud sélectionné. Le nom de la balise est toujours retourné en majuscules.

`nodeType`

Indique le type de node rencontré. Ce type prend différentes valeurs :

- 1 si le nœud est un élément.
- 2 si la sélection porte sur un attribut.
- 3 s'il s'agit du nœud de texte.
- 4 pour un nœud de section CDATA.
- 5 pour un nœud de référence à une entité.
- 6 pour un nœud d'entité.
- 7 pour un nœud d'instruction de traitement.
- 8 pour un nœud de commentaire.
- 9 pour un nœud de document.
- 10 pour un nœud de type de document.
- 11 pour un nœud de fragment de document.
- 12 pour un nœud de notation.

`nodeValue`

Permet d'obtenir ou de changer la valeur d'un nœud de type texte.

Exemples :

Selon l'arbre du document étudié au point C. précédent :

Pour le nœud élément <head>

`nodeName`	HEAD
`nodeType`	1
`nodeValue`	null (sans objet)
`parentNode`	Le noeud élément html
`childNodes`	Un enfant, le nœud élément title
`firstChild`	Le nœud élément title
`lastChild`	Le nœud élément title
`nextSibling`	Le nœud élément body
`previousSibling`	null (sans objet)

Pour le nœud élément <body>

nodeName	BODY
nodeType	1
nodeValue	null (sans objet)
parentNode	Le noeud élément html
childNodes	Deux enfants, les nœuds élément h1 et p
firstChild	Le nœud élément h1
lastChild	Le nœud élément p
nextSibling	null (sans objet)
previousSibling	Le node élément head

Pour le nœud élément <h1>

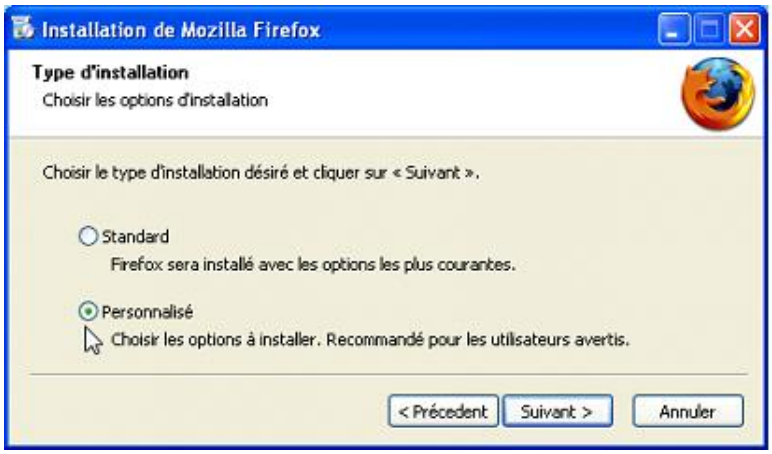
nodeName	H1
nodeType	1
nodeValue	null (sans objet)
parentNode	Le nœud élément body
childNodes	Un enfant, le nœud texte
firstChild	Le nœud texte
lastChild	Le nœud texte
nextSibling	Le nœud élément p
previousSibling	null (sans objet)

Firefox DOM Inspector

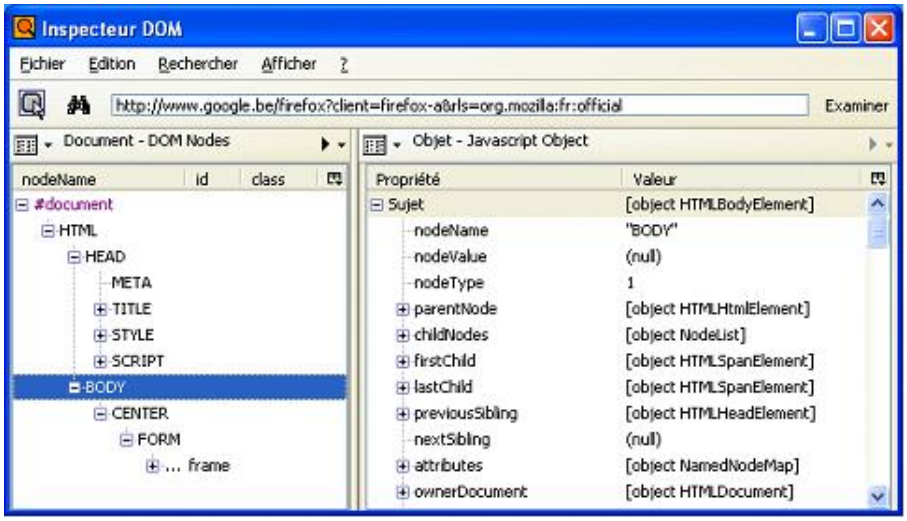
Grâce à ses nombreuses extensions, Firefox offre à la disposition des utilisateurs des fonctionnalités très utiles qui ne cessent de ravir les développeurs.

Parmi ces extensions, il y en a une, assez méconnue et peu documentée, qui se révèle d'une grande utilité pour l'apprentissage du DOM et l'exploration de l'arborescence du document. Il s'agit de Firefox DOM Inspector.

Cet outil est installé automatiquement avec Firefox, à condition que vous ayez choisi l'installation personnalisée. Si, comme la plupart des utilisateurs, vous avez choisi l'installation standard, la façon la plus simple pour disposer de cet outil consiste à désinstaller le navigateur et à le réinstaller en choisissant la bonne option (voir captures d'écran ci-après). Il n'y a aucune crainte à avoir pour votre profil, votre configuration, vos marque-pages ou autres extensions rien ne sera modifié.



- Une fois la page affichée, pour démarrer DOM Inspector, on peut utiliser le raccourci-clavier **[Ctrl] [Maj] i** ou passer par le menu **Outils - DOM Inspector**.

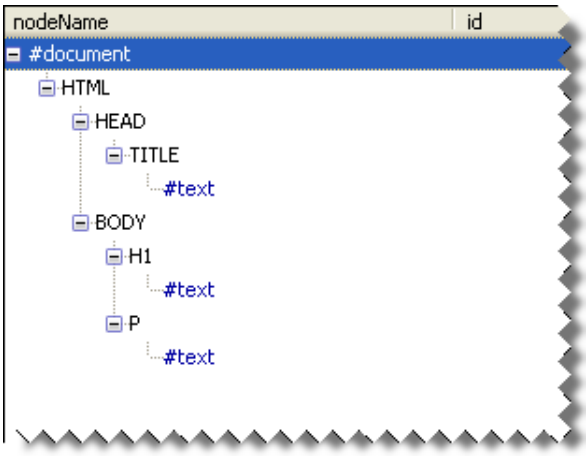


Une fois DOM Inspector ouvert, l'arborescence du document se trouve à gauche. Chaque ligne représente un nœud. On peut naviguer dans l'arbre du DOM en cliquant les petites icônes + ou - pour déployer ou refermer l'arborescence de chaque nœud. Dans la partie de droite, avec l'option **Javascript Object** du menu déroulant, s'affichent toutes les propriétés de nodeName à previousSibling (et plus encore...) du nœud sélectionné dans la partie gauche.

Reprenons le fichier Html du point Hierarchisation des noeuds de ce chapitre.

```
<html>
<head>
<title>Le DOM</title>
</head>
<body>
<h1>Titre de niveau 1</h1>
<p>Texte du paragraphe</p>
</body>
</html>
```

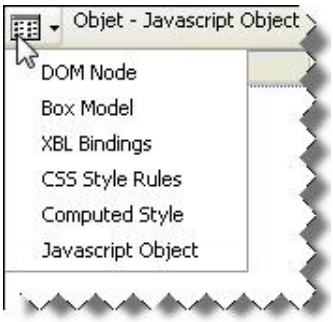
Celui-ci s'affiche dans DOM Inspector comme suit :



Pour des documents plus conséquents, on peut aussi utiliser la fonction de recherche par id, balise ou attribut en passant par menu **Rechercher - Rechercher un nœud** et **[F3]** pour trouver le nœud suivant.

Dans la fenêtre de gauche, vous pouvez choisir d'afficher différents types d'informations :

- **DOM Nodes** est la vue par défaut, elle affiche la structure de la page.
- **Box Model** permet de connaître la position, les dimensions, les marges, la bordure ou le remplissage d'une boîte.
- **XBL Bindings** est destiné aux développeurs qui utilisent le XUL (*XML-based User interface Language*).
- **CSS Style Rules** montre toutes les règles CSS appliquées au nœud sélectionné.
- **Computed Style** indique toutes les règles appliquées au nœud sélectionné, quelle que soit leur origine (déclaration explicite, script ou héritage).
- **JavaScript Object** montre tous les objets JavaScript utilisés.



Ainsi, DOM Inspector est bien plus qu'un outil pour explorer l'arborescence des nœuds. Il permet, par exemple, de débuguer et modifier en direct un document mais toutes ses fonctions dépassent le cadre de cet ouvrage.

Particularité de Firefox

Une utilisation plus intensive de DOM Inspector fait apparaître une interprétation particulière de l'arborescence du DOM de la part de Firefox. En effet, Firefox et les autres navigateurs de la famille Mozilla considèrent les sauts de ligne du code comme des nœuds de texte vides, chose que ne fait pas Microsoft Internet Explorer.

Notre document Html du point Hierarchisation des noeuds de ce chapitre est interprété comme tel par Internet Explorer.

Soit,

```
<html>
<head>
<title>Le DOM</title>
</head>
<body>
<h1>Titre de niveau 1</h1>
<p>Texte du paragraphe</p>
</body>
</html>
```

Par contre Firefox l'interprètera ainsi :

`<html>`nœud de texte

`<head>`nœud de texte

`<title>Le DOM</title>`nœud de texte

`</head>`nœud de texte

`<body>`nœud de texte

`<h1>Titre de niveau 1</h1>`nœud de texte

`<p>Texte du paragraphe</p>`nœud de texte

`</body>`nœud de texte

`</html>`

Le nombre d'enfants d'un élément est donc différent selon que le document est affiché par Internet Explorer ou Firefox. Ce que confirme l'expérience suivante.

Ainsi le même code `alert(x.childnodes.length)` révèle un résultat différent pour un document identique :

Dans Internet Explorer :

Dans Firefox

Cette différence d'interprétation du DOM peut entraîner des variations notables entre les deux navigateurs. Les développeurs s'en accommodent en limitant l'accès par les propriétés de l'objet nœud, à des éléments proches d'un objet déterminé. C'est ce que nous allons faire plus loin dans notre étude d'AJAX aux chapitres L'approche AJAX et AJAX par l'exemple.

Accéder aux nœuds

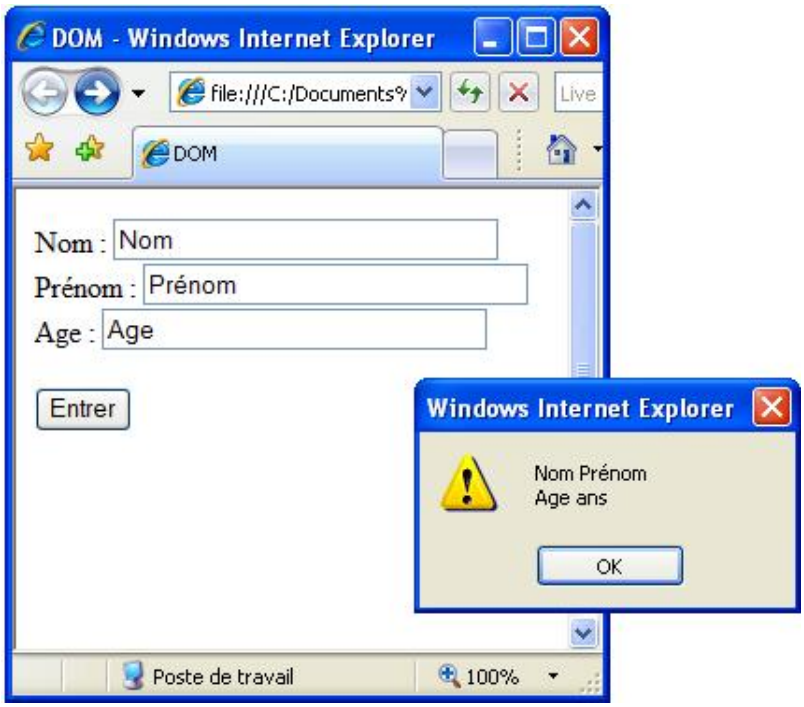
1. Par la procédure classique

C'est la méthode que nous avons étudiée et utilisée dans le chapitre partie Le JavaScript de cet ouvrage. Elle consiste à définir le chemin, d'objet en objet, vers l'objet concerné.

Cette méthode ne met pas encore en œuvre les spécifications du DOM et a été appelée a posteriori DOM niveau 0. Elle est encore tout à fait valable et reconnue par les différents navigateurs du marché, et même si son implémentation n'est pas toute récente elle est assez pratique dans le cas de formulaires par exemple.

Exemple

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function test() {
var nom = document.form.votrenom.value;
var prenom = document.form.votreprenom.value;
var age = document.form.votreage.value;
alert(nom + " " + prenom + "\n" + age + " ans");
}
//]]>
</script>
</head>
<body>
<form name="form" action="">
Nom : <input type="text" name="votrenom" value="Nom" /><br />
Prénom : <input type="text" name="votreprenom" value="Prénom" /><br />
Age : <input type="text" name="votreage" value="Age" />
<p><input type="submit" value="Entrer" onclick="test()" /></p>
</form>
</body>
</html>
```



2. Par la méthode getElementById

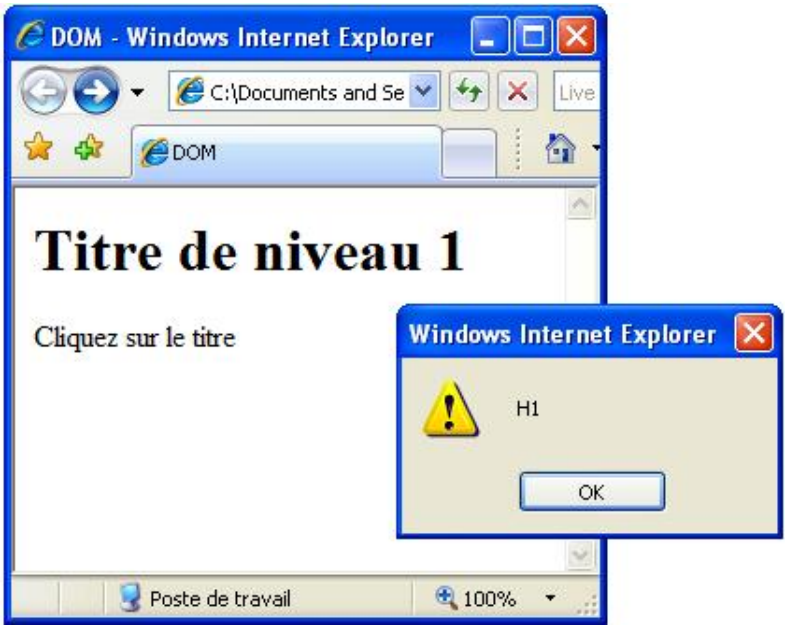
La méthode JavaScript `getElementById` parcourt le document Html ou Xhtml à la recherche d'un nœud unique qui a été spécifié par l'attribut *id*. Cet identifiant *id* doit être unique dans le document.

Le terme *Element* de getElementByID est bien au singulier car il ne peut y avoir qu'un seul identifiant portant ce nom.

Exemple

Au clic sur un titre de niveau 1, retourner dans une boîte d'alerte le nom du nœud (*nodeName*).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/javascript">
// <![CDATA[
function valeur()  {
var x = document.getElementById("titre").nodeName;
alert(x);
}
// ]]>
</script>
</head>
<body>
<h1 id="titre" onclick="valeur()">Titre de niveau 1</h1>
<p>Cliquez sur le titre</p>
</body>
</html>
```



La méthode `getElementById`, appliquée au document, accède à l'identifiant signalé en argument ("titre") et la propriété de nœud *nodeName* lui est appliquée. Cette valeur, stockée dans la variable x, est affichée dans la boîte d'alerte.

Cette méthode `getElementById` n'est que rarement utilisée dans un document XML car elle recherche les attributs de type id qui doivent alors être définis dans un DTD (*Document Type Definition*) particulier.

3. Par la méthode getElementsByName

La méthode `getElementsByName` permet de sélectionner les éléments portant un nom donné, spécifié par l'attribut name. Les éléments portant le même nom sont stockés dans une liste de nœuds. Cette liste se gère comme un tableau Array.

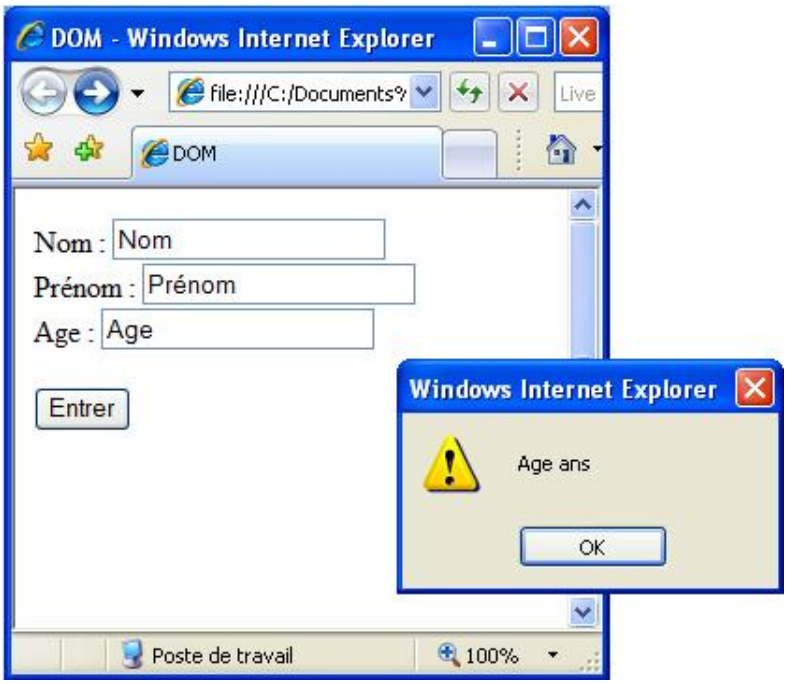
Le terme *Elements* de `getElementsByName` est bien au pluriel car plusieurs éléments portant le même nom peuvent se trouver dans le document.

Exemple

Accédons à la valeur de la troisième ligne de texte par la méthode `getElementsByName`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function test() {
var age = document.getElementsByName("in")[2].value;
alert(age + " ans");
}
//]]>
</script>
</head>
<body>
<form name="form" action="">
Nom : <input type="text" name="in" value="Nom" /><br />
Prénom : <input type="text" name="in" value="Prénom" /><br />
Age : <input type="text" name="in" value="Age" />
<p><input type="submit" value="Entrer" onclick="test()" /></p>
</form>
</body>
</html>
```

La méthode `getElementsByName`, appliquée au document, accède à la troisième balise ayant l'attribut `name="in"` par `getElementsByName("in")[2]`. Sa valeur est récupérée par *value*. Cette valeur, stockée dans la variable `age`, est affichée dans la boîte d'alerte.



L'attribut *name* est un héritage du Html 4.0. Son emploi est déprécié (*deprecated*) en Xhtml au profit de l'identifiant *id*. Ainsi l'emploi de getElementsByName est de moins en moins fréquent. Il est même ignoré dans certaines parutions récentes.

4. Par la méthode getElementsByTagName

La méthode `getElementsByTagName` parcourt le document à la recherche de toutes les balises d'un type spécifique, signalé en argument. Ces balises sont contenues dans une liste (*nodeList*) qui se gère comme les tableaux de type *Array*.

Le terme *Elements* de `getElementsByName` est bien au pluriel car il ne peut y avoir plusieurs balises de même type dans le document.

Exemple

Accédons à la valeur de la 3ème ligne de texte par getElementsByTagName.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
function test() {
var age = document.getElementsByTagName("input")[2].value;
alert(age + " ans");
}
//]]>
</script>
</head>
<body>
<form name="form" action="">
Nom : <input type="text" value="Nom" /><br />
Prénom : <input type="text" value="Prénom" /><br />
Age : <input type="text" value="Age" />
<p><input type="submit" value="Entrer" onclick="test()" /></p>
</form>
</body>
</html>
```

La méthode `getElementsByTagName`, appliquée au document, accède à la troisième balise `<input/>` par `getElementsByTagName("input")[2]`. Sa valeur est récupérée par *value*. Cette valeur, stockée dans la variable `age`, est affichée dans la boîte d'alerte.

La capture est identique à celle du point précédent.

Cette méthode, `getElementsByTagName`, est très fréquemment utilisée dans les documents XML.

5. Par les propriétés des nœuds

Il est théoriquement possible d'accéder à n'importe quel élément par un code du genre :

```
x.parentNode.lastChild.childNodes[2].firstChild.nextSibling;
```

Cette façon de procéder ne se révèle cependant pas très pratique à l'usage car :

- Le code devient rapidement illisible.
- Une simple mise à jour de la page risque de modifier complètement l'arborescence du document et nécessiterait alors la réécriture complète du code.
- Les navigateurs n'ont pas la même interprétation du DOM, spécialement en ce qui concerne les sauts de ligne dans le code que Firefox considère comme des éléments texte vide et que Explorer ignore (voir ci-avant au

point Particularité de Firefox).

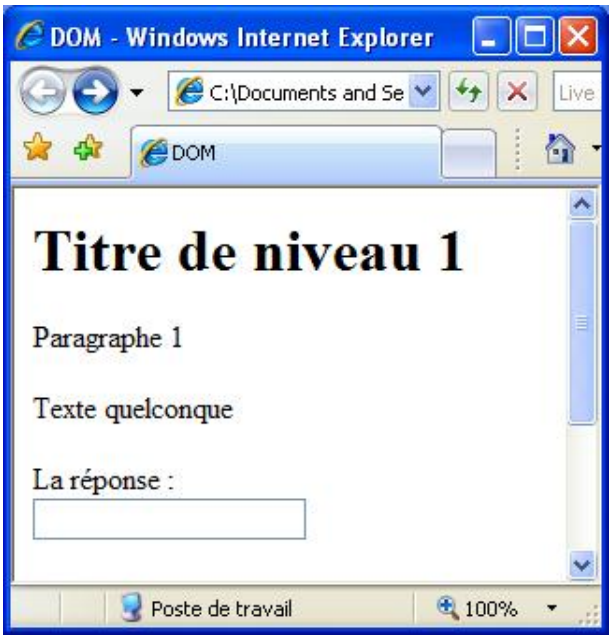
Ainsi, les développeurs utilisent les méthodes `getElementById`, `getElementsByName` ou `getElementsByTagName` pour se rapprocher de l'élément et, à partir de là, utilisent les propriétés firstChild, parentNode ou autres propriétés similaires pour accéder à l'élément souhaité.

Mettons en pratique la procédure déterminée soit un saut de longue distance et ensuite une exploration limitée.

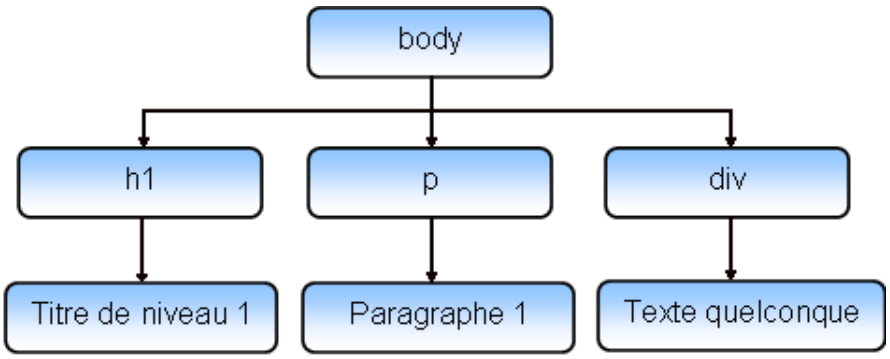
Exemple 1

Soit le code suivant :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
</head>
<body>
<h1 id="titre">Titre de niveau 1</h1>
<p>Paragraphe 1</p>
<div>Texte quelconque</div>
<form>
La réponse :<br />
<input id="texte" type="text" />
</form>
</body>
</html>
```



L'arborescence se présente comme suit :



- Au clic sur le document, affichons dans une ligne de texte, le texte compris entre les balises `<h1> ... </h1>`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/javascript">
// <![CDATA[
function valeur() {
var y=document.getElementById("texte");
var x=document.getElementById("titre").firstChild;
y.value=x.nodeValue;
}
// ]]>
</script>
</head>
<body onclick="valeur()">
<h1 id="titre">Titre de niveau 1</h1>
<p>Paragraphe 1</p>
<div>Texte quelconque</div>
<form>
La réponse :<br />
<input id="texte" type="text" />
</form>
</body>
</html>
```

L'élément `<h1>` est obtenu par la méthode `getElementById("titre")`. À partir de là, le nœud texte est obtenu par la propriété firstChid. Il suffit alors d'afficher la valeur par la propriété nodeValue.

- Au clic sur le document, afficher dans la ligne de texte, le texte compris entre les balises `<div> ... </div>`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/javascript">
// <![CDATA[
```

```
function valeur() {
var y = document.getElementById("texte");
var x = document.getElementById("titre").parentNode.childNodes[2].firstChild;
y.value=x.nodeValue;
}
// ]]>
</script>
</head>
<body onclick="valeur()">
<h1 id="titre">Titre de niveau 1</h1>
<p>Paragraphe 1</p>
<div>Texte quelconque</div>
<form>
La réponse :<br />
<input id="texte" type="text" />
</form>
</body>
</html>
```

La méthode `getElementById("titre")` accède à la balise `<h1>`. À partir de là, le nœud parent est obtenu par la propriété parentNodes, soit la balise `<body>`. On redescend alors vers l'élément enfant `<div>` par `childNodes[2]`. Il faut encore descendre d'un niveau pour atteindre le texte en utilisant firstChild. Le texte est affiché par la propriété nodeValue.

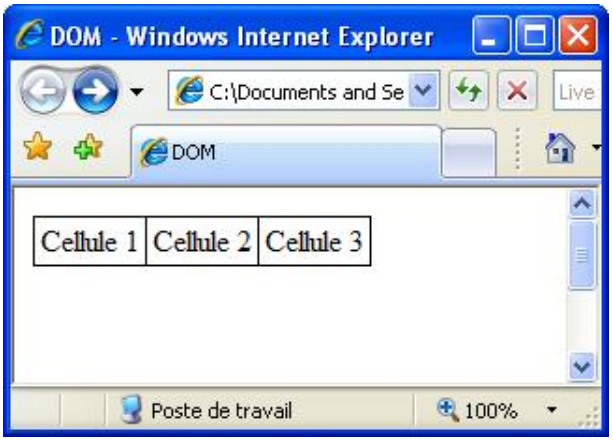
Notons que notre code n'est pas compatible avec Firefox, à cause des passages à la ligne qui sont alors considérés comme des éléments enfant (éléments de texte vide).

Exemple 2

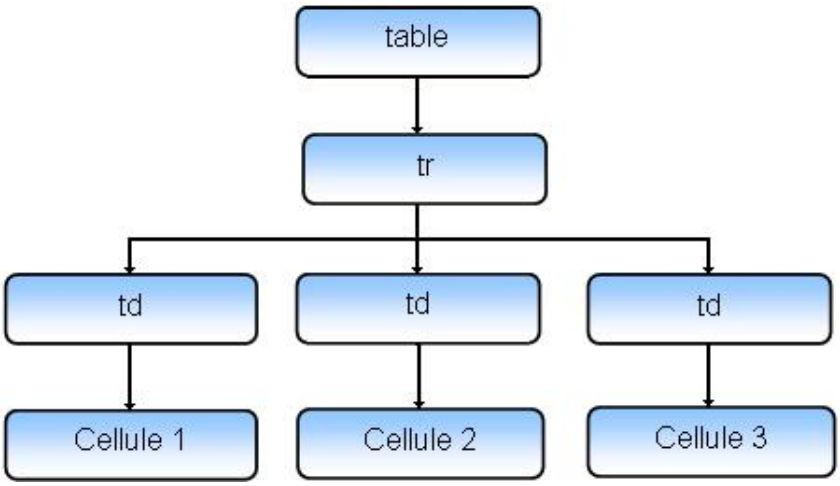
Soit à présent un tableau d'une ligne et trois colonnes :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
</head>
<body>
<table border="1" bordercolor="black" cellpadding=3"
style="border-collapse: collapse;">
<tr>
<td>Cellule 1</td>
<td id="di">Cellule 2</td>
<td>Cellule 3</td>
</tr>
</table>
```

```
</body>
</html>
```



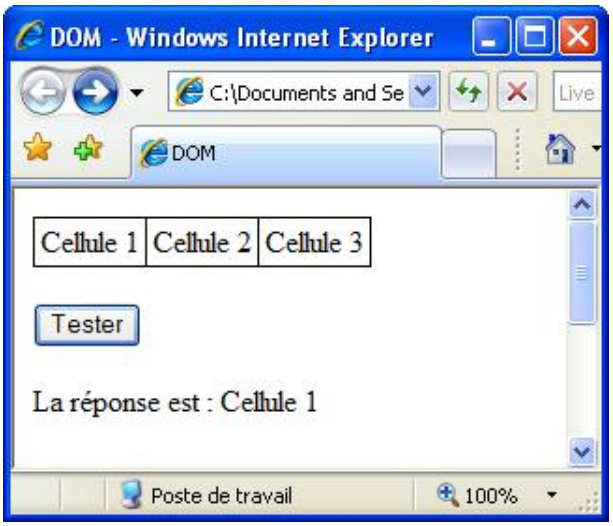
L'arborescence se présente comme suit :



- Au clic sur un bouton, afficher dans une balise `<div> ... </div>`, le texte contenu dans la première colonne du tableau, sachant que l'identifiant est placé au niveau de la seconde balise `<td>`, donc sur la deuxième colonne.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta<+>http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/javascript">
//<![CDATA[
function valeur() {
var y = document.getElementById("texte");
var x = document.getElementById("di").previousSibling.firstChild;
y.innerHTML =x.nodeValue;
}
//]]>
</script>
</head>
<body>
<table border="1" bordercolor="black" cellpadding=3"
style="border-collapse: collapse;">
<tr>
<td>Cellule 1</td>
<td id="di">Cellule 2</td>
<td>Cellule 3</td>
</tr>
```

```
</table>
<form action="">
<input type="button" value="Tester" onclick="valeur()" />
</form>
La réponse est :
<span id="texte"></span>
</body>
</html>
```



La balise <td> de seconde colonne est obtenue par getElementById("di"). Puis le premier <td> du tableau est accédé par la propriété previousSibling Le texte est atteint par firstChild. Celui-ci est alors affiché grâce à la propriété nodeValue.

L'écriture suivante est également valable :

```
document.getElementById("di").parentNode.firstChild.firstChild;
```

ou bien

```
document.getElementById("di").parentNode.childNodes[0].firstChild;
```

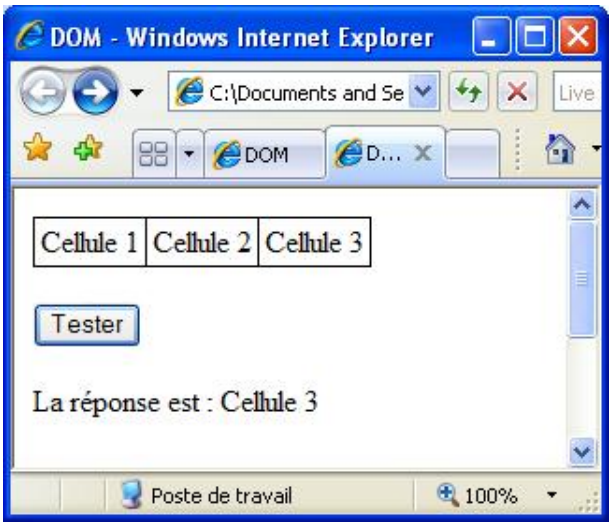
Pour modifier le contenu de la balise ... , nous avons utilisé la propriété innerHtml. Cette propriété innerHtml a été introduite par Internet Explorer et est donc propriétaire. Elle a été depuis reprise par les principaux navigateurs, dont Firefox, mais son implémentation peut dans certaines situations, donner des résultats différents.

La propriété innerHtml n'est pas un standard du W3C mais elle est tellement commode pour changer le contenu d'une balise ou que les développeurs l'ont largement adoptée.

- Au clic sur un bouton, afficher dans une balise ... , le texte contenu dans la dernière colonne du tableau.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/javascript">
// <![CDATA[
function valeur() {
var y=document.getElementById("texte");
var x=document.getElementById("di").nextSibling.firstChild;
y.innerHTML = x.nodeValue;
```

```
}
// ]]>
</script>
</head>
<body>
<table border="1" bordercolor="black" cellpadding=3"
style="border-collapse: collapse;">
<tr>
<td>Cellule 1</td>
<td id="di">Cellule 2</td>
<td>Cellule 3</td>
</tr>
</table>
<form action="">
<input type="button" value="Tester" onclick="valeur()" />
</form>
La réponse est :
<span id="texte"></span>
</body>
</html>
```



La balise `<td>` de seconde colonne est obtenue par `getElementById("di")`. Puis le premier `<td>` du tableau est accédé par la propriété nextSibling . Le texte est atteint par firstChild. Celui-ci est alors affiché grâce à la propriété nodeValue.

L'écriture suivante st également valable :

```
document.getElementById("di").parentNode.lastChild.firstChild;
```

ou bien

```
document.getElementById("di").parentNode.childNodes[2].firstChild;
```

Accéder aux attributs

Les attributs sont des propriétés de l'élément, et non pas des enfants de celui-ci. Ainsi, nous avons besoin de faire appel à des prescriptions du DOM niveau 2.

La propriété *attributes* renvoie une liste des attributs d'un élément spécifié. Cette liste de nœuds attributs est renvoyée sous forme d'un objet de type NamedNodeMap. Ce qui implique que les attributs seront accessibles par leur nom.

Soit,

```
var paragraphe = document.getElementsByTagName("p")[0];
var attributs = paragraphe.attributes;
```

La variable `paragraphe` récupère le premier paragraphe du document. La propriété *attributes* appliquée à cette variable liste l'ensemble des attributs de celui-ci.

La méthode `getNamedItem()` retrouve un nœud selon le nom spécifié en argument.

Soit :

```
var paragraphe = document.getElementsByTagName("p")[0];
var attributs = paragraphe.attributes;
var language= attributs.getNamedItem("lang");
```

La variable `language` contient l'attribut *lang* du premier paragraphe du document.

Exemple

Retrouvons la langue des différents titres :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/javascript">
// <![CDATA[
function trouver()
{var selection = document.getElementById("liste").selectedIndex;
var paragraphe = document.getElementsByTagName("h2")[selection];
var attributs = paragraphe.attributes;
var language= attributs.getNamedItem("lang").nodeValue;
alert(language);
}
// ]]>
</script>
</head>
<body>
<h2 lang="fr">Bienvenue!</h2>
<h2 lang="en">Welcome!</h2>
<h2 lang="de">Willkommen!</h2>
<h2 lang="it">Benvenuto!</h2>
<form action="">
<select id="liste">
<option>Titre 1</option>
<option>Titre 2</option>
<option>Titre 3</option>
<option>Titre 4</option>
</select>
<input type="button" onclick="trouver(liste)" value="Trouver la
langue" />
</form>
</body>
</html>
```

La variable `selection` contient l'index sélectionné (selectedIndex) de la liste déroulante. Par `getElementsByTagName ("h2")[selection]`, on accède au paragraphe concerné. La liste des attributs (`paragraphe.attributes`) est créée. Enfin

la variable language nous fournit la valeur (`nodeValue`) de l'attribut *lang* par le code `attributs.getNamedItem("lang")`.

Les attributs en XML peuvent contenir des données. Il est ainsi très utile de pouvoir y accéder dans les applications AJAX.

Modifier la hiérarchisation

Le W3C DOM vous permet de créer vos propres nœuds élément ou nœuds texte et de les insérer dans le document, modifiant ainsi la hiérarchie de celui-ci.

1. La méthode createElement

La méthode `createElement` permet, comme son nom le suggère, de créer un nouvel élément.

Exemple

```
var x = document.createElement("h1");
var x = document.createElement("div");
```

Il faut noter que l'élément ainsi créé n'apparaît pas encore dans le document. Il faut utiliser les méthodes `appendChild` ou `insertBefore` pour l'ajouter à celui-ci.

2. La méthode createTextNode

La méthode `createTextNode` crée un nœud texte dans le document.

Exemple

```
var x = document.createElement("h1");
var y = document.createTextNode("Titre de niveau 1");
```

Il faut noter que le nœud ainsi créé n'apparaît pas encore dans le document. Il faut utiliser les méthodes `appendChild` ou `insertBefore` pour l'ajouter à celui-ci.

3. La méthode appendChild

La méthode `appendChild` ajoute un élément comme dernier nœud enfant d'un nœud spécifié en argument.

La méthode `appendChild` s'applique donc toujours à un nœud parent.

Exemple

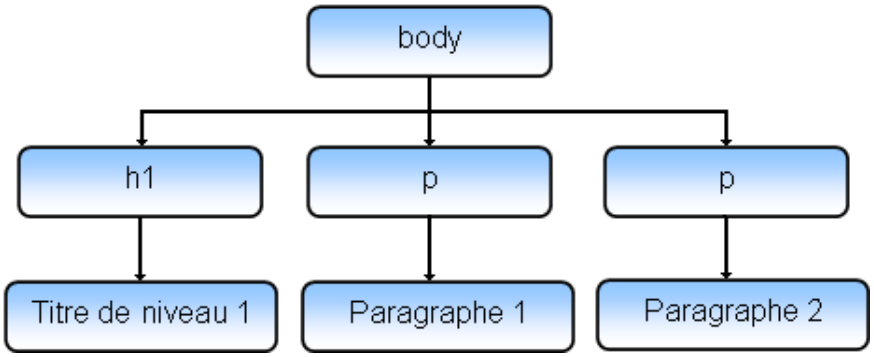
```
var x = document.getElementsByTagName('p')[0];
x.parentNode.appendChild(x);
```

Prenons une application complète pour illustrer de façon détaillée, le fonctionnement de la méthode `appendChild`.

Soit un fichier Xhtml :

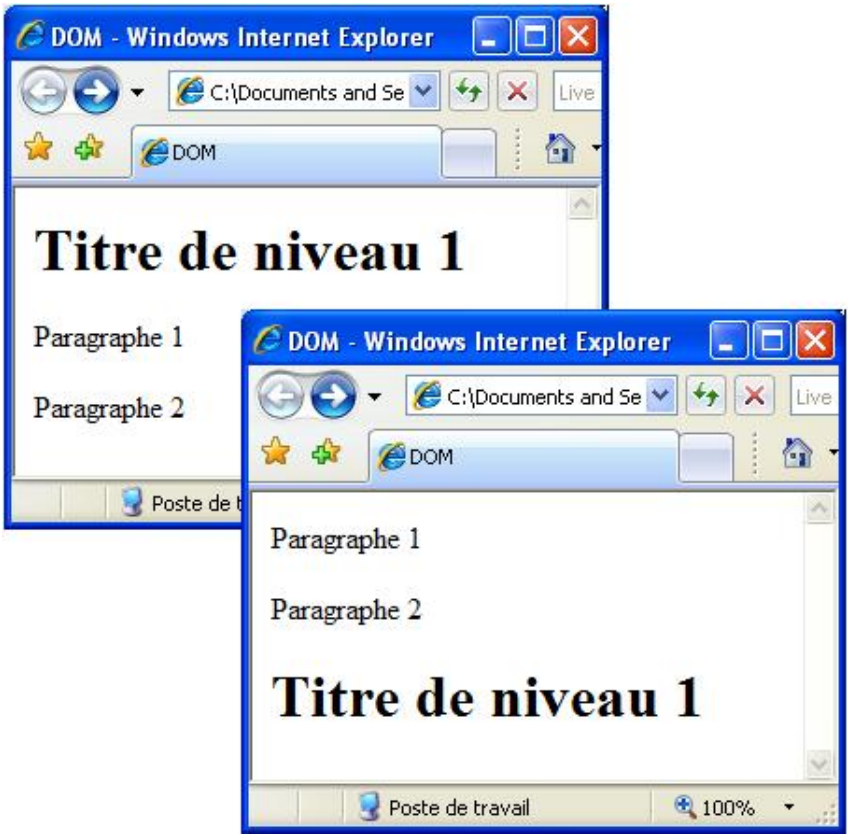
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
</head>
<body>
<h1>Titre de niveau 1</h1>
<p>Paragraphe 1</p>
<p>Paragraphe 2</p>
</body>
</html>
```

L'arbre des nœuds, limité à la balise body et ses descendants, est le suivant ;



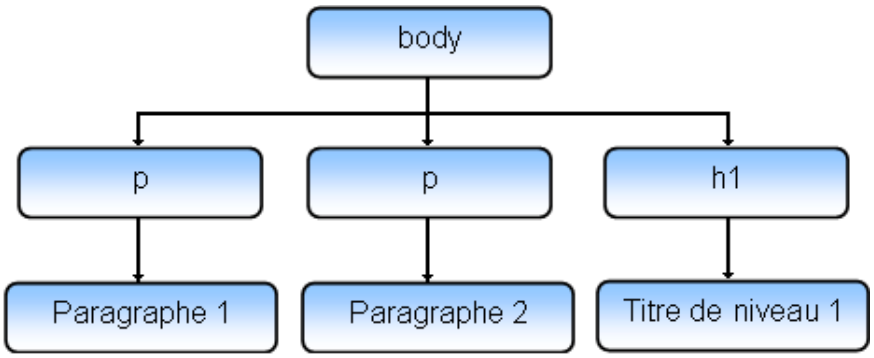
- Au clic de la souris dans la fenêtre du document, appliquons la méthode `appendChild` à la balise `<h1>`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>.
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function actualiser(){
var x = document.getElementsByTagName('h1')[0];
x.parentNode.appendChild(x);
}
//]]>
</script>
</head>
<body onclick=actualiser()>
<h1>Titre de niveau 1</h1>
<p>Paragraphe 1</p>
<p>Paragraphe 2</p>
</body>
</html>
```



La balise <h1> est obtenue par getElementsByTagName('h1')[0], puis la balise parent <body> est atteinte par la méthode parentNode. Enfin, la balise <h1> est placée comme dernier enfant par appendChild, ainsi <h1> se retrouve en dernière position.

L'arbre du document a donc été modifié :



- Ou encore, à partir du fichier Xhtml initial, ajoutons au document un élément créé par createElement.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function actualiser(){
var x = document.createElement("p");
var texte = document.createTextNode("Je suis le petit nouveau");
y = x.appendChild(texte);
document.body.appendChild(x);
}
//]]>
```

```
</script>
</head>
<body onclick=actualiser()>
<h1>Titre de niveau 1</h1>
<p>Paragraphe 1</p>
<p>Paragraphe 2</p>
</body>
</html>
```

Par la variable x, la méthode `createElement("p")` crée un nouveau paragraphe. Un nouveau texte est défini par `createTextNode("Je suis le petit nouveau")`. Le texte est associé à la balise par `x.appendChild(texte)`. Le nouvel élément dans sa totalité est inséré dans le document comme dernier enfant de la balise parent `<body>` par la propriété appendChild.

4. La méthode insertBefore

La méthode `insertBefore` permet d'insérer un nœud avant un autre nœud.

Cette méthode est utilisée lorsque l'on souhaite ajouter un élément sans qu'il devienne le dernier élément enfant (voir `appendChild`).

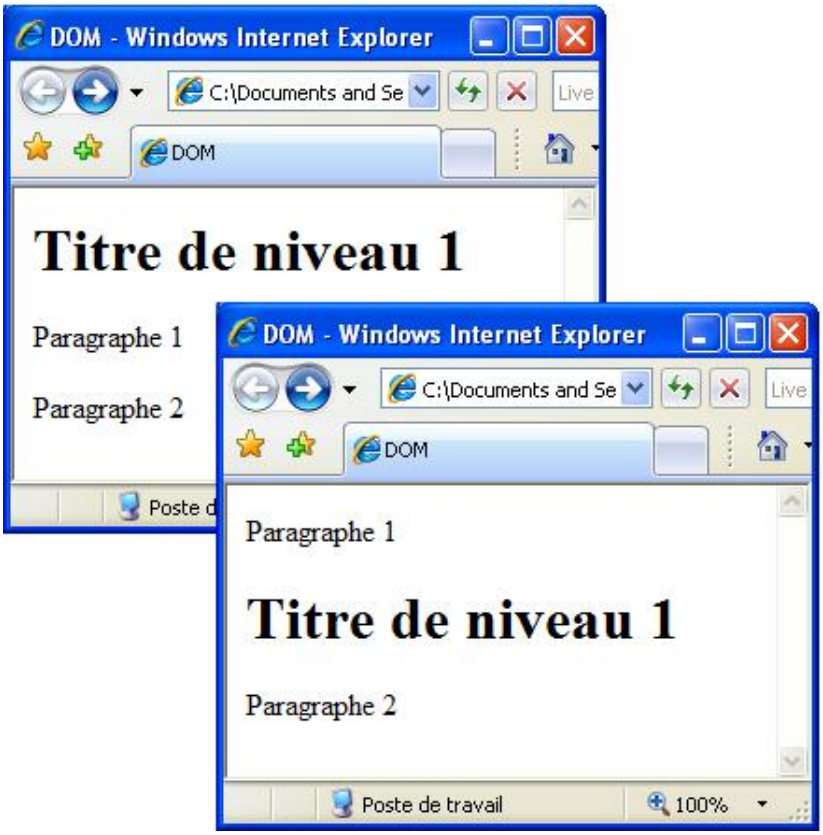
Exemple

```
x.parentNode.insertBefore(x,y);
```

Ainsi, en reprenant le document Xhtml du point précédent, insérons la balise `<h1>` après la première balise de paragraphe (`<p>`).

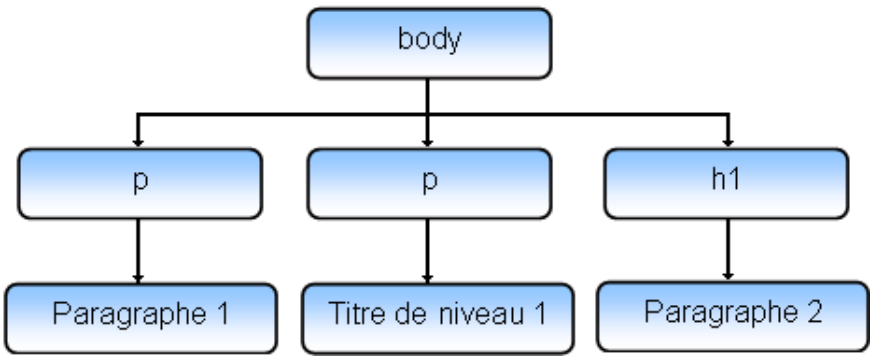
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
```

```
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function actualiser(){
var x = document.getElementsByTagName('p')[0];
var y = document.getElementsByTagName('h1')[0];
x.parentNode.insertBefore(x,y);
}
//]]>
</script>
</head>
<body onclick=actualiser()>
<h1>Titre de niveau 1</h1>
<p>Paragraphe 1</p>
<p>Paragraphe 2</p>
</body>
</html>
```



La variable x repère par `getElementsByTagName('p')[0]` le premier paragraphe `<p>`. La variable y désigne avec `getElementsByTagName('h1')[0]` la balise `<h1>`. Ensuite l'ordre des enfants de la balise `<body>(parentNode)` est interverti par `insertBefore(x,y)`.

L'arbre modifié se présente comme suit :



5. La méthode replaceChild

La méthode `replaceChild` remplace un nœud par un autre nœud.

Le nœud ainsi remplacé, est supprimé ainsi que tous ses descendants.

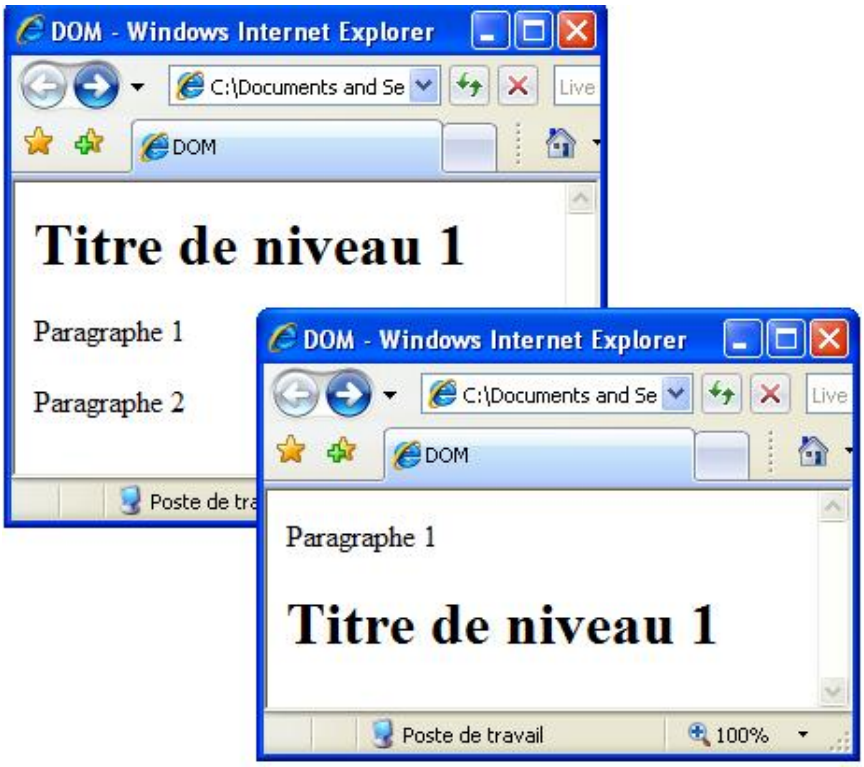
Exemple

```
x.parentNode.replaceChild(x,y);
```

Remplaçons le deuxième paragraphe de notre document Xhtml par l'élément <h1>.

Le code devient :

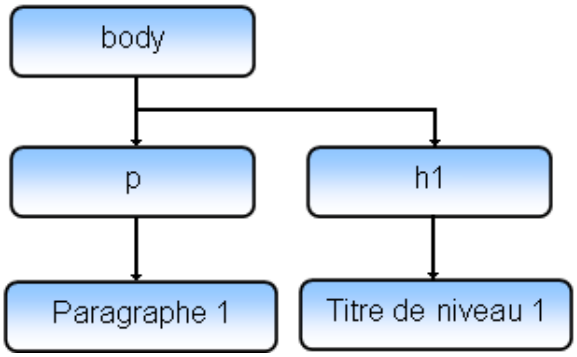
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function actualiser(){
var x = document.getElementsByTagName('h1')[0];
var y = document.getElementsByTagName('p')[1];
x.parentNode.replaceChild(x,y);
}
//]]>
</script>
</head>
<body onclick=actualiser()>
<h1>Titre de niveau 1</h1>
<p>Paragraphe 1</p>
<p>Paragraphe 2</p>
</body>
</html>
```



La balise <h1> est sélectionnée par `getElementsByTagName('h1')[0]` et le second paragraphe <p> par `getElementsByTagName('p')[1]`. Par la propriété replaceChild, on remplace le second élément par le premier.

Il faut noter que l'élément <h1> a changé de position et que le second paragraphe <p> a disparu du document. Le node texte de <h1> a suivi le changement de position.

L'arbre modifié devient :



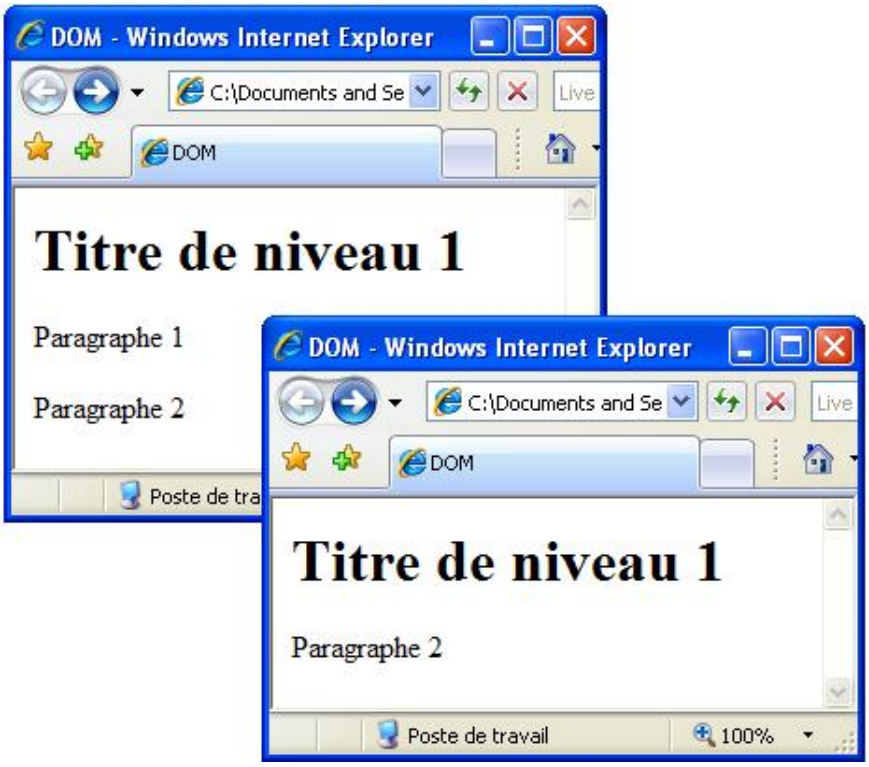
6. La méthode removeChild

La méthode removeChild supprime le nœud (et éventuellement ses descendants) fourni en argument.

```
var x = document.getElementsByTagName('p')[0];
x.parentNode.removeChild(x);
```

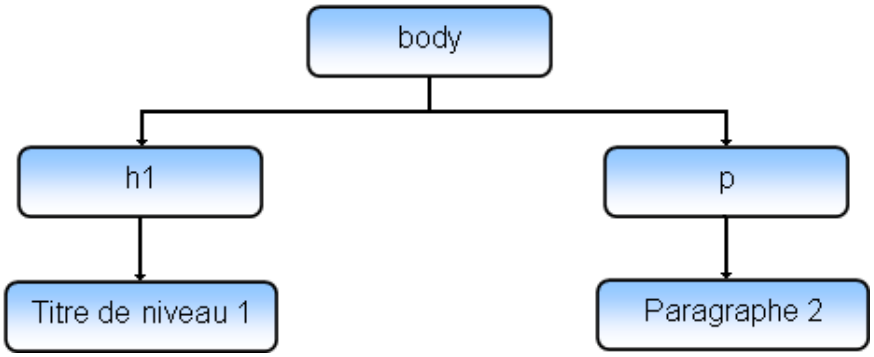
Supprimons le premier paragraphe <p> du document.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function actualiser(){
var x = document.getElementsByTagName('p')[0];
x.parentNode.removeChild(x);
}
//]]>
</script>
</head>
<body onclick=actualiser()>
<h1>Titre de niveau 1</h1>
<p>Paragraphe 1</p>
<p>Paragraphe 2</p>
</body>
</html>
```



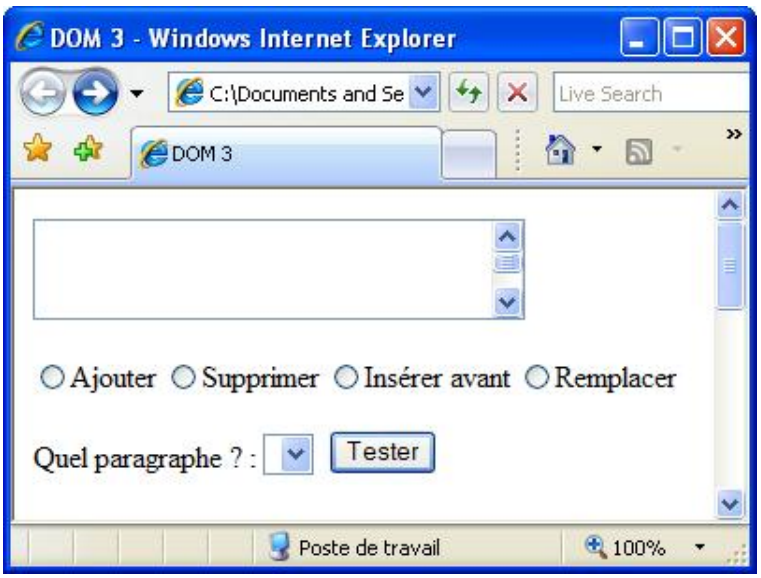
Après avoir repéré l'élément à supprimer, soit `getElementsByTagName ('p')[0]`, il est supprimé de la balise `<body>` par la propriété `removeChild`.

L'arbre du document se présente comme suit :



7. Une application récapitulative

Ce script permet d'ajouter, de supprimer, d'insérer et de remplacer des paragraphes de texte (voir les quatre boutons radio pour les quatre actions possibles). Le paragraphe à insérer dans la page provient d'une zone de texte. Si l'action nécessite de connaître un numéro de paragraphe celui-ci sera fourni par un menu déroulant, actualisé à chaque action antérieure.



Le code du fichier Xhtml initial est :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<form action="">
<p>
<textarea id="zonetexte" rows="3" cols="30"></textarea>
</p>
<p>
<input type="radio" name="bouton" />Ajouter
<input type="radio" name="bouton" />Supprimer
<input type="radio" name="bouton" />Insérer avant
<input type="radio" name="bouton" />Remplacer
</p>
<p>
Quel paragraphe ? : <select id="liste"></select> 
<input type="submit" value="Tester"/></p>
</form>
<div id="modifiable"> </div>
</body>
</html>
```

Commençons le script par la fonction ajouter()

```
function ajouter() {
var contenu = document.getElementById("zonetexte").value;
var newparag = document.createElement("p");
var newtexte = document.createTextNode(contenu);
newparag.appendChild(newtexte);
changement.appendChild(newparag);
document.getElementById("zonetexte").value="";
}
```

La zone de texte est atteinte par la méthode `getElementByID` prenant en paramètre son identifiant (`zonetexte`). La propriété value nous fournit sa valeur. Cette valeur est stockée dans la variable `contenu`. La variable `newparag` crée par la méthode `createElement` un nouveau nœud élément (ici un paragraphe `p`). La variable `newtexte` crée par `createTextNode`, un nouvel élément texte dont le contenu est fourni par la variable contenu. Le nœud texte est associé au nouveau nœud élément par `newparag.appendChild(newtexte)`. Le tout est inclus en dernière position dans le document `changement.appendChild(newparag)` où changement a été défini par ailleurs comme la zone de la balise `<div id="modifiable"> ... </div>`. La dernière ligne réinitialise la zone de texte.

Le script doit par ailleurs déclencher cette fonction ajouter() lorsque le bouton radio **Ajouter** est sélectionné. Il doit aussi modifier dynamiquement les options du menu déroulant <select>, lorsque le paragraphe est effectivement ajouté au document. Ce que fait la fonction changer().

```
function changer() {
var typeaction = -1;
var nombreparag = changement.getElementsByTagName("p").length;
var boutonradio = document.getElementsByTagName("form")[0].bouton;
for (var i=0; i<boutonradio.length; i++) {
if (boutonradio[i].checked) {
typeaction = i;
}
}
if (typeaction==-1) {
alert("Pas de bouton radio selectionné");
}
if (typeaction==0) {
ajouter();
}
document.getElementById("liste").options.length = 0;
for (i=0; i<changement.getElementsByTagName("p").length; i++) {
document.getElementById("liste").options[i] = new Option(i+1);
}
return false;
}
```

La variable `typeaction` est initialisée à -1 (`var typeaction = -1`). Le nombre de paragraphes (`var nombreparag`) est donné par la longueur (`length`) du tableau renvoyé par `getElementsByTagName("p")`.

Le nombre de boutons radio est donné par le nombre de boutons (`bouton`) présents dans le formulaire (`getElementsByTagName("form")[0]`). Une boucle *for* passe ces boutons en revue pour trouver l'élément sélectionné (`checked`). Le numéro d'ordre du bouton (`boutonradio[i]`) est le numéro du type d'action (`typeaction = i`).

Si aucun bouton n'a été sélectionné par l'utilisateur, la variable typeaction a toujours gardé sa valeur initiale. Un test sur cette valeur initiale (`if (typeaction==-1)`) permet de déclencher une boîte d'alerte.

Ici, le premier bouton étant sélectionné, le type d'action prend la valeur 0. Ainsi au moyen d'un simple test conditionnel (`if (typeaction==0)`), la fonction ajouter() peut être appelée.

Le nombre d'options (`options.length`) du menu déroulant (`liste`) est réinitialisé à 0. Une boucle *for* compte tous les paragraphes (`getElementsByTag-Name("p")`) de la zone `<div id="modifiable">` (changement). À chaque passage de la boucle, la liste des options est alimentée (`getElementById("liste").options[i]`) par le nombre en cours plus 1 de la boucle (`new Option(i+1)`).

Pour la fonction supprimer() :

```
function supprimer() {
var choixout = document.getElementById("liste").selectedIndex;
var tous = changement.getElementsByTagName("p");
var asupprimer = tous[choixout];
changement.removeChild(asupprimer);
}
```

La variable `choixout` mémorise l'option retenue par l'utilisateur dans le menu déroulant (`getElementById("liste").selectedIndex`). La variable `choixout` note ainsi le numéro d'ordre du paragraphe que l'on souhaite supprimer. La variable tous renvoie tous les paragraphes (`getElementsByTagName("p")`) sous forme d'un tableau. L'élément à supprimer (`var asupprimer`) est déterminé en appliquant la variable choixout au tableau (`tous[choixout]`). La suppression est obtenu par la méthode `removeChild(asupprimer)` appliquée à la zone de la balise `<div>`.

Il ne faut pas oublier d'ajouter à la fonction changer() :

```
if (typeaction==1 && nombreparag > 0) {
supprimer();
}
```

La fonction supprimer n'a de sens que s'il y a des paragraphes dans la zone `<div>` (`nombreparag > 0`).

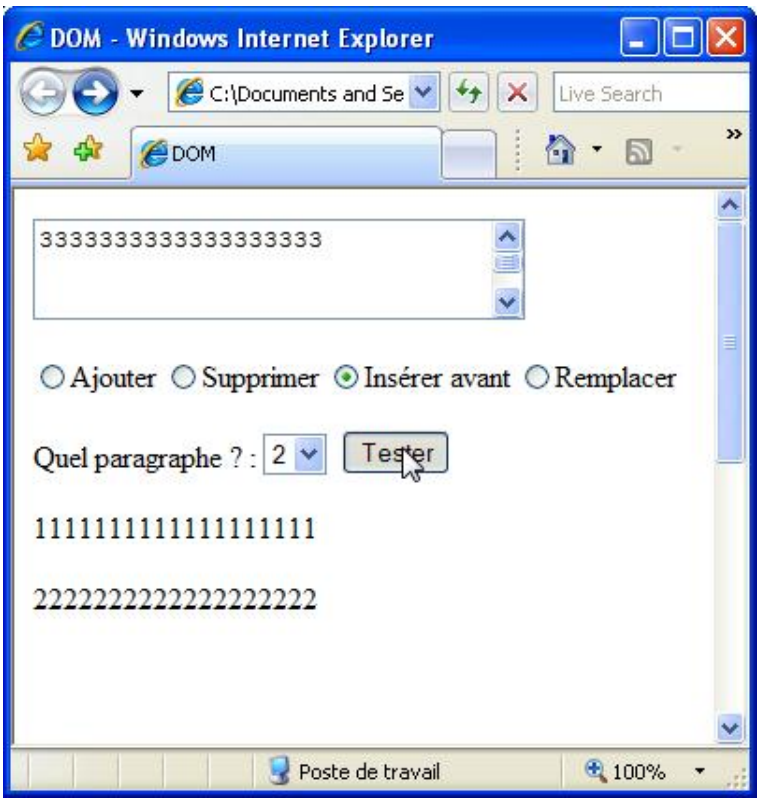
Passons maintenant à la fonction inserer() :

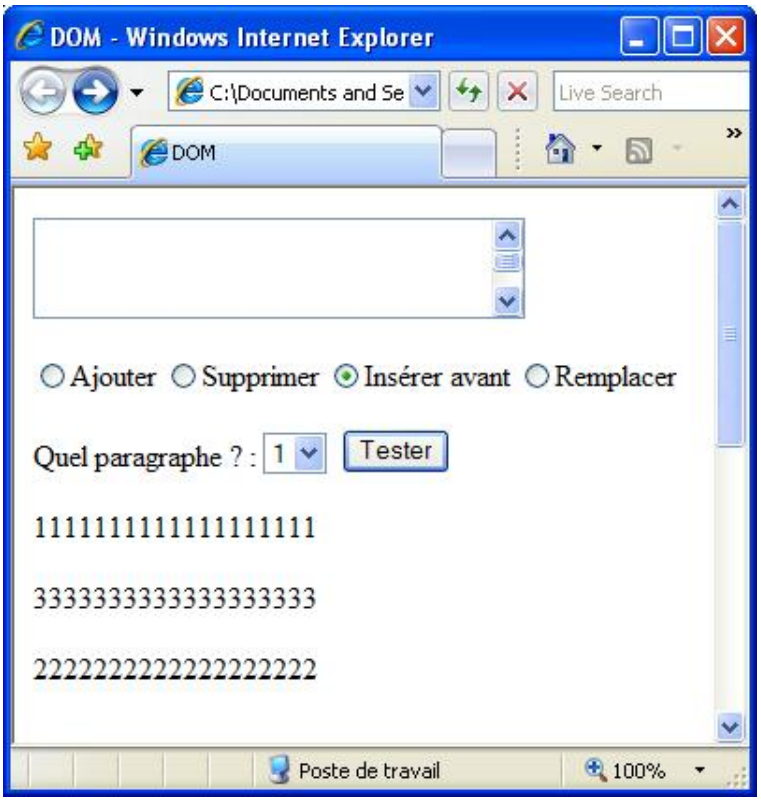
```
function inserer() {
var choixins = document.getElementById("liste").selectedIndex;
var intexte = document.getElementById("zonetexte").value;
var newparag = document.createElement("p");
var newtexte = document.createTextNode(intexte);
newparag.appendChild(newtexte);
var tous = changement.getElementsByTagName("p");
var oldparag = tous[choixins];
changement.insertBefore(newparag,oldparag);
document.getElementById("zonetexte").value="";
}
```

Le choix de l'utilisateur (`getElementById("liste").selectedIndex`) est stocké dans la variable *choixins*. La variable *intexte* récupère le contenu (value) de la zone de texte (`getElementById("zonetexte")`). Un nouveau paragraghe (`newparag`) est créé par la méthode `createElement("p")`. Un nouveau nœud de texte est créé par la méthode `createTextNode(intexte)`. Le nœud de texte est associé à l'élément (`newparag.appendChild(newtexte)`). La variable *tous* liste les paragraphes de la zone modifiable (`getElementsByTagName("p")`). Puis le paragraphe signalé par l'utilisateur (`var oldparag = tous[choixins]` est identifié. Le nouveau paragraphe (newparag) est inséré avant celui-ci (oldparag) par la méthode `insertBefore(newparag,oldparag)`. Enfin, la zone de texte est réinitialisée.

Il faut encore ajouter à la fonction changer() :

```
if (typeaction==2 && nombreparag > 0) {
inserer();
}
```





Voici la fonction `remplacer()`.

```
function remplacer() {
var inchoix = document.getElementById("liste").selectedIndex;
var intexte = document.getElementById("zonetexte").value;
var newparag = document.createElement("p");
var newtexte = document.createTextNode(intexte);
newparag.appendChild(newtexte);
var tous = changement.getElementsByTagName("p");
var oldparag = tous[inchoix];
```

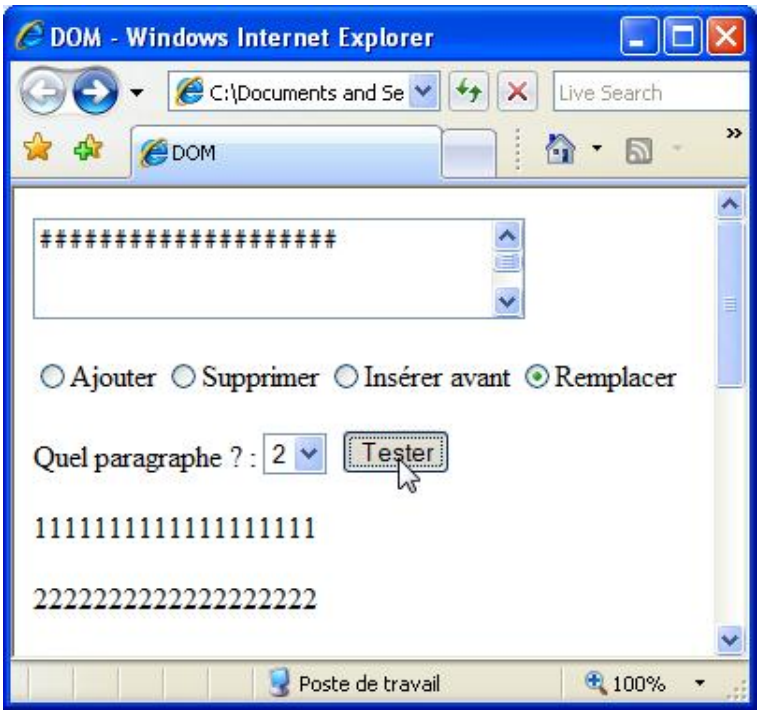
```
changement.replaceChild(newparag,oldparag);
document.getElementById("zonetexte").value="";
}
```

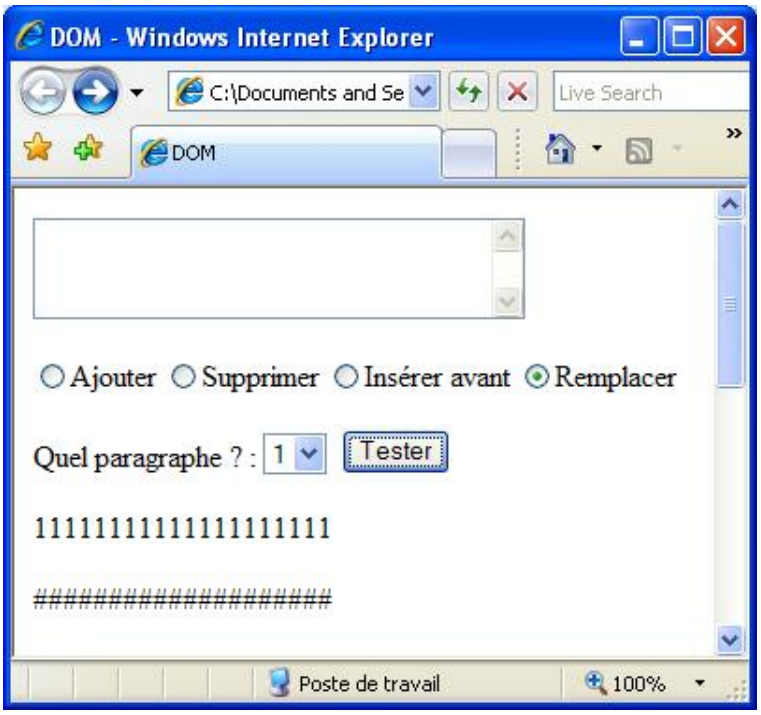
La variable *inchoix* note le choix de l'utilisateur dans le menu déroulant. La variable *intexte* reprend le contenu de la zone de texte. La variable *newparag* crée le nouveau paragraphe par la méthode `createElement("p")`. La variable *newtexte* crée un nouveau nœud de texte `createTextNode(intexte)`. Le nœud de texte est associé au nœud élément (`newparag.appendChild(newtexte)`).

Le paragraphe à supprimer (`oldparag`) est fourni par `tous[inchoix]`. Le remplacement du paragraphe s'effectue par la méthode `replaceChild(newparag,oldparag)`. La zone de texte est ensuite réinitialisée.

Et il faut ajouter la fonction changer() :

```
if (typeaction==3 && nombreparag > 0) {
remplacer();
}
```





Le fichier avec le script complet devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>DOM</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type="text/JavaScript">
//<![CDATA[
window.onload = init;
var changement;

function init() {
document.getElementsByTagName("form")[0].onsubmit = function()
{return changer();}
changement = document.getElementById("modifiable");
}

function ajouter() {
var contenu = document.getElementById("zonetexte").value;
var newtexte = document.createTextNode(contenu);
var newparag = document.createElement("p");
newparag.appendChild(newtexte);
changement.appendChild(newparag);
document.getElementById("zonetexte").value="";
}

function supprimer() {
var choixout = document.getElementById("liste").selectedIndex;
var tous = changement.getElementsByTagName("p");
var asupprimer = tous[choixout];
changement.removeChild(asupprimer);
}

function inserer() {
var choixins = document.getElementById("liste").selectedIndex;
var intexte = document.getElementById("zonetexte").value;
var newtexte = document.createTextNode(intexte);
var newparag = document.createElement("p");
newparag.appendChild(newtexte);
var tous = changement.getElementsByTagName("p");
var oldparag = tous[choixins];
changement.insertBefore(newparag,oldparag);
document.getElementById("zonetexte").value="";
}

function remplacer() {
var inchoix = document.getElementById("liste").selectedIndex;
var intexte = document.getElementById("zonetexte").value;
var newtexte = document.createTextNode(intexte);
var newparag = document.createElement("p");
newparag.appendChild(newtexte);
var tous = changement.getElementsByTagName("p");
var oldparag = tous[inchoix];
changement.replaceChild(newparag,oldparag);
document.getElementById("zonetexte").value="";
}

function changer() {
var typeaction = -1;

var nombreparag = changement.getElementsByTagName("p").length;
var boutonradio = document.getElementsByTagName("form")[0].bouton;
for (var i=0; i<boutonradio.length; i++) {
if (boutonradio[i].checked) {
typeaction = i;
```

```
}
}
if (typeaction==-1) {
alert("Pas de bouton radio selectionné");
}
if (typeaction==0) {
ajouter();

}
if (typeaction==1 && nombreparag > 0) {
supprimer();
}
if (typeaction==2 && nombreparag > 0) {
inserer();
}
if (typeaction==3 && nombreparag > 0) {
remplacer();
}
document.getElementById("liste").options.length = 0;
for (i=0; i<changement.getElementsByTagName("p").length; i++) {
document.getElementById("liste").options[i] = new Option(i+1);
}
return false;
}
//]]>
</script>
</head>
<body>
<form action="">
<p>
<textarea id="zonetexte" rows="3" cols="30"></textarea>
</p>
<p>
<input type="radio" name="bouton" />Ajouter
<input type="radio" name="bouton" />Supprimer
<input type="radio" name="bouton" />Insérer avant
<input type="radio" name="bouton" />Remplacer
</p>
<p>
Quel paragraphe ? : <select id="liste"></select> 
<input type="submit" value="Tester"/></p>
</form>
<div id="modifiable"> </div>
</body>
</html>
```

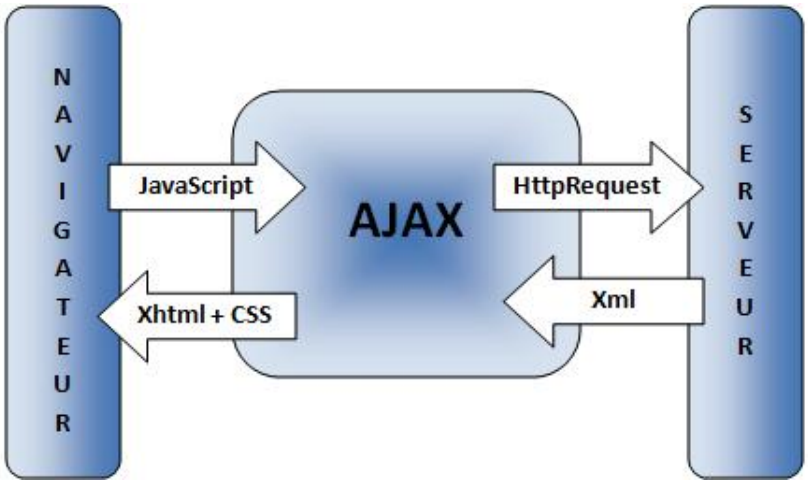
Présentation

L'objet XMLHttpRequest permet d'envoyer des requêtes HTTP vers le serveur, de recevoir des réponses et de mettre à jour une partie de la page Web. En mode asynchrone, cette mise à jour se réalise sans avoir à recharger la page et donc de façon totalement transparente pour l'utilisateur.

L'objet XMLHttpRequest s'utilise donc dans une architecture de type client/serveur.

Détaillons son mode de fonctionnement :

- L'objet XMLHttpRequest est créé par le moteur JavaScript du navigateur.
- Cet objet est alors utilisé pour effectuer une requête HTTP vers le serveur.
- La réponse est fournie par ce dernier au navigateur.
- À l'aide du Xhtml et des feuilles de style CSS, le résultat est ensuite affiché dans le navigateur.



L'avantage des applications AJAX réside principalement dans la diminution de la bande passante car seules les données réclamées sont affichées sans avoir à recharger tout le document, d'où une interactivité et une fluidité accrue.

L'objet XMLHttpRequest a initialement été développé en tant que contrôle ActiveX de Microsoft XML Core Services (MSXML), implémenté dans Internet Explorer 5.0, soit en 1999. Il y eu, au fil des années, différentes versions de ce composant. Citons :

- Microsoft.XMLHTTP
- MSXML2.XMLHTTP
- MSXML2.XMLHTTP.3.0
- MSXML2.XMLHTTP.4.0
- MSXML2.XMLHTTP.5.0
- MSXML2.XMLHTTP.6.0

En 2002, l'objet XMLHttpRequest a été repris par le projet Mozilla comme objet natif JavaScript. Il est ainsi reconnu dès les premières versions de Firefox. Les autres navigateurs du marché ont suivi en commercialisant Safari 1.2 en 2004, Konqueror, Opera en 2005 et dernièrement, à la surprise générale, Internet Explorer 7.

Il s'agit d'une bonne nouvelle pour les web-développeurs : Internet Explorer 7 intègre la requête XMLHTTP en tant qu'objet JavaScript natif. Ceci améliore la compatibilité syntaxique entre les différents navigateurs.

Bien heureusement, les appels à l'ancien contrôle ActiveX sont toujours supportés par Internet Explorer 7.

La compatibilité de l'objet `XMLHttpRequest` peut être jugée globalement bonne pour les navigateurs récents. Mais dans l'absolu, cet objet n'est pas reconnu par les navigateurs de la "vieille génération".

Il faut noter que l'objet XMLHttpRequest n'est pas un standard du Web. Cependant cette situation ne devrait pas perdurer car, suite au succès grandissant de l'objet XMLHttpRequest, le W3C a édité différents "working drafts" (5 avril 2006, 19 juin 2006, 27 septembre 2006) qui devraient déboucher sur une standardisation prochaine. Il ne faudra pas prévoir de grands changements par rapport à la version actuelle d'`XmlHttpRequest` que l'on trouve actuellement dans les navigateurs. Le but de cette démarche est surtout de standardiser spécifiquement les spécifications de l'objet XMLHttpRequest, afin que tous les concepteurs s'accordent sur la même implémentation.

Créer un objet XMLHttpRequest

Pour Internet Explorer (IE 5, IE 5.5, IE 6), il faut passer par des méthodes ActiveX propriétaires pour créer une instance de l'objet.

```
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
```

ou encore :

```
var xhr = new ActiveXObject("Msxml2.XMLHTTP");
```

Ces deux méthodes fournissent un résultat identique mais varient cependant dans leur mode de fonctionnement.

La première est indépendante des versions de MSXML. Avec Microsoft.XMLHTTP, l'ActiveXObject initialise la dernière bonne version de MSXML. Cet objet peut-être MSXML 1.0 mais, de nos jours, avec la mutation vers XP et son système de mise à jour, il va être vraisemblablement remplacé par des versions plus récentes.

La seconde se base sur une version particulière de MSXML2. En encodant MSXML2.XMLHTTP, le navigateur va passer en revue les différentes versions des librairies MSXML 2.0 pour déterminer la meilleure version pour créer cet objet. Chaque échec risque d'entraîner un message d'erreur. Il est utile de prévoir ces éventualités dans le code avec des instructions de type`try ... catch` (voir plus loin).

Pour les autres navigateurs (Mozilla, Firefox, Safari, Konqueror, Opéra, IE7), l'objet natif XMLHttpRequest est utilisé.

```
var xhr = new XMLHttpRequest();
```

Comme cette solution est la plus compatible à l'heure actuelle, c'est celle-ci qui sera adoptée dans les instructions.

Le code pourrait être :

```
if(window.XMLHttpRequest){
  var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
  var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Un test préliminaire vérifie si le navigateur supporte l'objet natif XMLHttpRequest soit, dans le cadre de cet ouvrage, pour Firefox et Internet Explorer 7. Si c'est le cas (réponse true), l'objet est créé dans la variable `var xhr = new XMLHttpRequest()`. Dans le cas contraire (réponse false), l'objet est créé par le contrôle ActiveX soit pour Internet Explorer 5, 5.5 ou 6. Dans ce cas l'objet est stocké dans la variable `xhr = new ActiveXObject("Microsoft. XMLHTTP")`.

Il est peut être nécessaire de prévoir un message d'alerte pour les (rares) utilisateurs dont le navigateur ne reconnaît pas l'objet `XMLHttpRequest` et pour lequel l'application AJAX ne fonctionne pas.

Le code devient alors :

```
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
```

Si l'on souhaite utiliser MSXML2.XMLHTTP, le code le plus fréquemment rencontré est :

```
if(window.XMLHttpRequest) {
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
try {
var xhr = new ActiveXObject("Msxml2.XMLHTTP");
}
catch (e) {
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
```

```
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
```

La version JavaScript 1.5 a introduit un gestionnaire d'exception avec les instructions`try ... catch`.

Le principe est simple. L'instruction try désigne le code qui doit être essayé. Si aucune erreur n'est survenue, le bloc de commandes de catch est purement et simplement ignoré. Si par contre, une exception ou erreur est détectée, le contrôle de l'exécution passe directement au bloc de commandes désigné par l'instruction catch. Le type d'exception est identifié dans une variable (identificateur). À la fin du bloc `catch`, le programme continue son exécution par l'instruction qui suit.

La suite donnée aux exceptions identifiées dans la variable (identificateur) peut prendre des formes multiples selon les besoins du programmeur. Elles peuvent être ignorées, traitées par du code ou affichées dans un message d'erreur. Il est important de noter que ces exceptions ne bloquent pas le déroulement du processus.

```
try
{
// instructions à essayer
}
catch (identificateur)
{
// instructions alternatives
}
```

Le code teste d'abord l'existence d'un objet natif XMLHttpRequest. Si celui-ci n'est pas trouvé, le script se rabat sur le contrôle ActiveX par *if(window.ActiveXObject)*. On demande alors d'essayer (try) de trouver le module MSXML2 (MSXML2.XMLHTTP). Ce qui est le cas le plus probable (voir point Présentation du présent chapitre). Si le module MSXML2 n'est pas trouvé, la commande Microsoft. XMLHTTP est prise en compte (catch).

Si le développeur veut être assuré que JavaScript prenne en compte la dernière version du contrôle ActiveX, on peut alors imaginer le script suivant. On essayera d'abord de créer l'objet avec la dernière version de MSXML soit par MSXML2.XMLHTTP.6.0. En cas d'échec, une version plus ancienne est essayée et ainsi de suite jusqu'à ce que l'objet soit créé.

Le code proposé est :

```
var xhr;
try {
xhr = new XMLHttpRequest();
}
catch(e) {
var XmlHttpVersions = new Array('MSXML2.XMLHTTP.6.0',
'MSXML2.XMLHTTP.5.0',
'MSXML2.XMLHTTP.4.0',
'MSXML2.XMLHTTP.3.0',
'MSXML2.XMLHTTP',
'Microsoft.XMLHTTP');
for (var i=0; i<XmlHttpVersions.length && ! xhr; i++) {
try {
xhr = new ActiveXObject(XmlHttpVersions[i]);
}
catch (e) {}
}
}
if (!xhr)
alert("L'objet XMLHttp n'a pas pu être créé.");
}
```

En premier lieu l'objet XMLHttpRequest() est instancié. Si l'opération échoue, l'objet ActiveXObject() est instancié, pour les versions d'Internet Explorer 6 ou antérieures. Dans ce cas, une boucle for passe en revue les différentes versions de 'MSXML2.XMLHTTP qui ont été encodées sous la forme d'un tableau Array. On essaye alors de créer l'objet ActiveX par la version 6 de MSXML2. Si cela échoue, l'erreur est ignorée car le bloc de commandes de catch est vide. Le code continue en tentant de créer l'objet par MSXML2.XMLHTTP.5.0 et ainsi de suite jusqu'à ce que l'objet soit créé.

Si l'objet XMLHttpRequest n'a pu être créé (`if (!xhr)`), une boîte d'alerte s'affiche pour avertir l'utilisateur.

Exemple

Vérifions au clic d'un bouton, si la variable xhr est bien un objet XMLHttpRequest. La réponse est affichée dans une boîte d'alerte.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
var xhr = null;
function getxhr(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
alert("xhr est un " + xhr);
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
alert("xhr est un " + xhr);
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
}
//]]>
</script>
</head>
<body>
<form action="">
<input type="button" value="Test" onclick="getxhr()" />
</form>
</body>
</html>
```

Le résultat dans Internet Explorer 7 est :

Notons que ceci est encore une application de type client.

Propriétés et méthodes

Avant de les étudier en détail, jetons un œil sur les propriétés et méthodes de l'objet XMLHttpRequest.

Les propriétés de l'objet XMLHttpRequest sont :

Propriété	**Description**
readyState	Retourne l'état de la requête.
onreadystatechange	Gestionnaire d'événements qui prend en charge les changements d'état de la requête.
status	Renvoie le code numérique de la réponse du serveur HTTP.
statusText	Renvoie le message lié au code numérique de la réponse du serveur HTTP.
responseText	Réponse du serveur sous forme de chaîne de caractères.
responseXML	Réponse du serveur sous forme d'un document XML.

Les méthodes de l'objet XMLHttpRequest sont :

Méthode	**Description**
`open()`	Initialise la requête selon une série de paramètres fournis en argument.
`send()`	Effectue la requête, avec éventuellement l'envoi de données.
`getAllResponseHeaders()`	Renvoie l'ensemble de l'en-tête HTTP de la réponse sous la forme d'une chaîne de caractères.
`getResponseHeader()`	Renvoie la valeur d'un seul champ de l'en-tête HTTP sous la forme d'une chaîne de caractères.
`setRequestHeader()`	Assigne une valeur à un champ d'en-tête HTTP éventuellement envoyé lors de la requête.
`overrideMimeType()`	<$I[]Force un document à être traité selon un type de contenu (Mime Type) particulier.
`abort()`	<$I[]>Annule la requête.

Effectuer une requête

Les méthodes utilisées pour toutes les requêtes HTTP sur le serveur sont `open()` et `send()` :

- `open()` configure la requête en spécifiant divers paramètres ;
- `send()` effectue réellement la requête et accède ainsi au serveur.

La méthode `open()` se note plus précisément open ("méthode","url du fichier", mode).

Commentaires :

- "méthode" prend la valeur GET pour recevoir un fichier du serveur distant ou POST pour envoyer un fichier vers le serveur. Notons l'emploi de guillemets.
- "url du fichier" est l'adresse absolue ou relative du fichier. Notons l'emploi de guillemets. En fonction des limitations de l'objet XMLHttpRequest, le fichier doit impérativement se situer dans le même domaine.
- mode prend la valeur false pour le mode asynchrone qui nous intéresse dans le cadre de notre étude ou true pour une exécution en mode normal, ou synchrone.

  En mode synchrone, tant que la réponse à la requête n'est pas parvenue, le script se met en pause et bloque le navigateur. Ce qui n'est jamais très convivial pour l'utilisateur.

  En mode asynchrone, le script poursuit son exécution sans bloquer le processus. Ce qui permet une navigation plus fluide et plus confortable pour l'utilisateur.

La méthode `send()` se note plus précisément `send(contenu)`

Commentaires :

- Avec GET, il faut noter send(null).
- Avec POST, il faut noter send("données"). Les données doivent être sous la forme d'une chaîne de requête, comme :

  ```
  nom=valeur&autrenom=autrevaleur&ainsi=desuite
  ```

Par exemple :

```
open("POST", "http://localhost/ajax/test.php", true);
send("param1=x&param2=y");
```

Il faut également noter que lors de l'envoi de données avec la méthode POST, il est recommandé de changer l'en-tête HTTP de la requête à l'aide de la ligne suivante (voir point Quelques propriétés détaillées - Méthode getResponse Header du présent chapitre) : `setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');`

Le code devient ;

```
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr_object.open("GET", "test.txt", false);
xhr_object.send(null);
```

où `test.txt` est l'adresse relative du fichier.

Cet ouvrage étant centré sur l'implémentation d'AJAX côté client, nous utiliserons surtout la méthode GET<$I[]>. La méthode POST<$I[]> envoie les données vers des programmes serveur PHP ou ASP.net par exemple : le traitement des données dépasse le cadre de cet ouvrage.

Quelques propriétés détaillées

1. Propriété readyState

La propriété readyState de l'objet XMLHttpRequest permet de connaître l'évolution de la requête. Cette évolution prend successivement 5 valeurs reprises dans le tableau suivant :

Valeur	**Description**
0 ou uninitialized	La requête n'est pas initialisée.
1 ou loading	Début de transfert des données.
2 ou loaded	Les données sont transférées.
3 ou interactive	Les données reçues sont partiellement accessibles.
4 ou complete	Les données sont complètement accessibles.

C'est la valeur 4 ou *complete* qui retient notre attention car, une fois obtenue, les données transférées peuvent alors être traitées.

L'accès à la propriété readyState s'effectue par l'événement *onreadystatechange* auquel doit être associé une fonction. Dans le cas présent, cette fonction permettra de connaître la valeur prise par readyState.

Le code devient :

```
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function() {
if(xhr.readyState == 4) {
// instructions
}
}
xhr.open("GET", test.txt, false);
xhr.send(null);
```

Il faut noter que le code ajouté pour connaître l'état de la requête, doit prendre place dans le script avant l'instruction send().

Exemple 1

Au clic sur un bouton, testons si le fichier test.txt a bien été réceptionné (valeur 4 de readyState). Le résultat est affiché dans une boîte d'alerte.

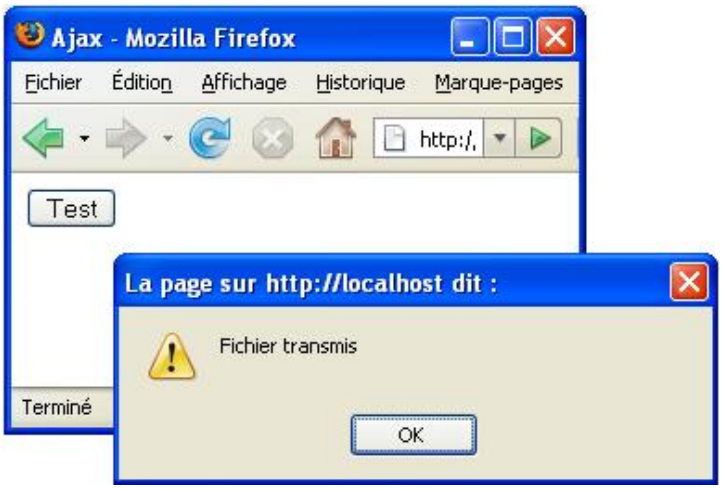
Le fichier exemple.htm :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function getxhr() {
var xhr=null;
if(window.XMLHttpRequest){
```

```
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4) {
alert("Fichier transmis");
}
}
xhr.open("GET", "test.txt", true);
xhr.send(null);
}
//]]>
</script>
</head>
<body>
<form action="">
<input type="button" value="Test" onclick="getxhr()" />
</form>
</body>
</html>
```

Le fichier test.txt comporte les mots "Je viens du serveur".

Le résultat dans Firefox 2 est :



Cet exemple constitue notre première application client-serveur.

Pour plus de commodité, elle nécessite l'installation d'un serveur HTTP local, par exemple IIS (*Internet Information Services*) ou EasyPHP (voir chapitre L'environnement de travail - outils côté-serveur).

Il faut ainsi charger le fichier exemple.htm et test.txt sur le disque dur dans le dossier Inetpub - wwwroot si vous utilisez IIS ou Program Files - EasyPHP1-8 - www si vous utilisez EasyPHP.

Dans le navigateur, le fichier exemple.htm est accessible en introduisant l'adresse http://localhost/exemple.htm si vous utilisez IIS ou http://127.0.0.1/exemple.htm si vous utilisez EasyPHP.

Si vous modifiez le fichier htm, il faut garder à l'esprit que le serveur local utilise la copie mise en cache du navigateur au lieu d'extraire le nouveau le fichier sur le disque dur, risquant ainsi de ne pas tenir compte des modifications apportées.

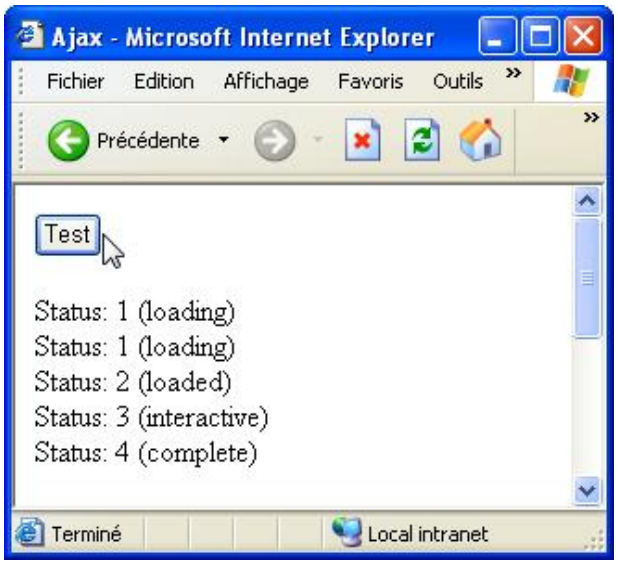
Exemple 2

Au clic sur un bouton, notons les différents états de la requête dans une balise `<div> ... </div>`. Le fichier test.txt est inchangé.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
```

```
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function getxhr(){
var xhr=null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
monDiv = document.getElementById("DivElement");
if(xhr.readyState == 1){
monDiv.innerHTML += "Status: 1 (loading)<br />";
}
if (xhr.readyState == 2){
monDiv.innerHTML += "Status: 2 (loaded)<br />";
}
if (xhr.readyState == 3){
monDiv.innerHTML += "Status: 3 (interactive)<br />";
}
if (xhr.readyState == 4){
monDiv.innerHTML += "Status: 4 (complete)<br />";
}
}
xhr.open("GET", "test.txt", true);
xhr.send(null);
}
//]]>
</script>
</head>
<body>
<form action="">
<input type="button" value="Test" onclick="getxhr()" />
</form>
<div id="DivElement"></div>
</body>
</html>
```

Le résultat dans Explorer 6 est :



Ne vous étonnez pas si vous n'obtenez pas exactement le même message. Selon l'implémentation de l'objet

XMLHttpRequest, certains états peuvent être ignorés. Ainsi, on rapporte qu'Opera ne tient compte que des états 3 et 4. Internet Explorer ne retient que les états 2, 3 et 4 lorsque des versions plus récentes de MSXML 2.0 sont utilisées.

2. Propriété status

La propriété status renvoie le code HTTP indiquant le résultat de la dernière requête effectuée.

Celui-ci vaut par exemple :

- 200 si la requête a été exécutée avec succès,
- 403 pour un accès interdit,
- 404 pour un fichier non trouvé
- 500 pour une erreur interne au serveur.

Vous trouverez ci-dessous une liste plus détaillée des codes HTTP.

Code	**Description**
100	Continue
101	Switching protocols
200	OK
201	Created
202	Accepted
203	Non-Authoritative formation
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
400	Bad Request
401	Unauthorized
402	Payment Required

403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentification Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Suitable
417	Expectation Failed
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

Une liste complète peut être consultée à l'adresse :

www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

Les informations fournies par readyState sont, dans la pratique, trop partielles car la requête peut avoir été effectuée mais sans succès (par exemple avec un code 404 fichier non trouvé). Ainsi, il est prudent de doubler le test *readyState == 4 par status==200* qui confirme que la requête a bien été effectuée avec succès.

Le code devient alors ;

```
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function() {
if(xhr.readyState == 4 && xhr.status == 200) {
// instructions
```

```
}
}
xhr.open("GET", test.txt, false);
xhr.send(null);
```

Le code complet de l'exemple précédent devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function getxhr(){
var xhr = null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200) {
alert("Fichier transmis");
}
}
xhr.open("GET", "test.txt", true);
xhr.send(null);
}
//]]>
</script>
</head>
<body>
<form action="">
<input type="button" value="Test" onclick="getxhr()" />
</form>
</body>
</html>
```

La capture d'écran est identique à celle du point Quelques propriétés détaillées.

On peut aussi imaginer un autre code.

```
if (xhr.readyState == 4)
{
if (xhr.status == 200)
{
// instructions
}
}
```

Ici, le test pour vérifier si status vaut 200 ne s'effectue que lorsque le fichier a été transmis (readystate == 4).

3. Propriété responseText

La propriété responseText contient la réponse de la requête sous forme de chaîne de caractères (String).

Exemple 1

Au clic d'un bouton, affichez dans une boîte d'alerte le texte du fichier test.txt.

Le fichier test.txt comporte la phrase : "Je viens du serveur".

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function getxhr(){
var xhr = null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200) {
alert(xhr. responseText);
}
}
xhr.open("GET", "test.txt", true);
xhr.send(null);
}
//]]>
</script>
</head>
<body>
<form action="">
<input type="button" value="Test" onclick="getxhr()" />
</form>
</body>
</html>
```

Le résultat dans Internet Explorer 7 est :

Exemple 2

Au clic d'un bouton, affichez dans une ligne de texte, le contenu du fichier test.txt. Le fichier test.txt comporte la phrase : "Je viens du serveur".

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
```

```
//<![CDATA[
function getxhr(){
var xhr = null;
if (window.XMLHttpRequest){
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
document.getElementById("txt").value = "Reçu du serveur - " +
xhr.responseText;
}
}
xhr.open("GET", "test.txt", true);
xhr.send(null);
}
//]]>
</script>
</head>
<body>
<form name="ajax" action="">
<p>
<input type="button" value="Envoyer la requête" onclick="getxhr()" />
<input type="text" name="texte" id="txt" size="32" value="" />
</p>
</form>
</body>
</html>
```

Le résultat dans Firefox :

Exemple 3

Au clic d'un bouton, affichez dans une boîte d'alerte, le texte du fichier test.xml.

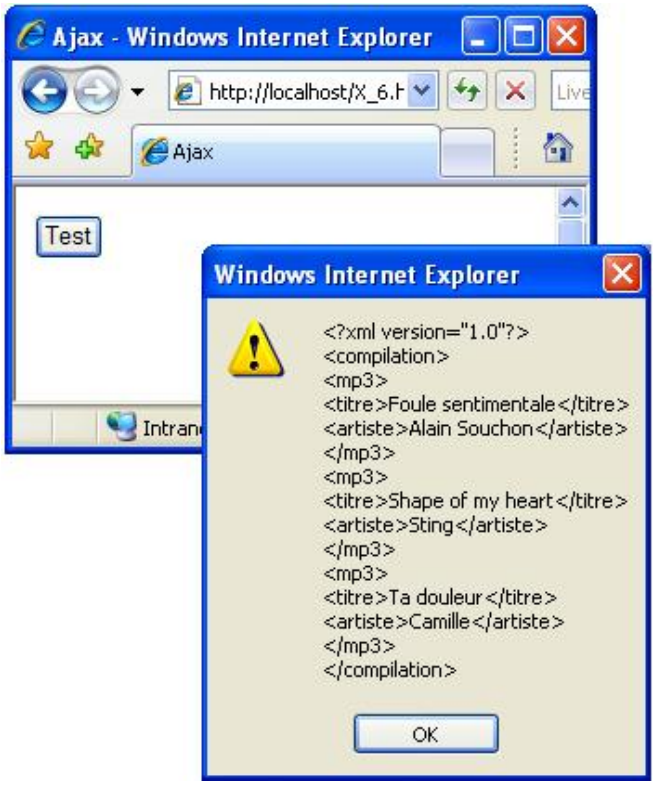
Le fichier test.xml est le suivant :

```
<?xml version="1.0"?>
<compilation>
<mp3>
<titre>Foule sentimentale</titre>
<artiste>Alain Souchon</artiste>
</mp3>
<mp3>
<titre>Shape of my heart</titre>
<artiste>Sting</artiste>
</mp3>
<mp3>
<titre>Ta douleur</titre>
<artiste>Camille</artiste>
```

```
</mp3>
</compilation>
```

Le code du fichier htm est :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function getxhr(){
var xhr=null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
alert(xhr.responseText);
}
}
xhr.open("GET", "test.xml", true);
xhr.send(null);
}
//]]>
</script>
</head>
<body>
<form action="">
<input type="button" value="Test" onclick="getxhr()" />
</form>
</body>
</html>
```



Le fichier XML est retourné dans cet exemple, comme une chaîne de caractère et non sous la forme d'un objet avec la structuration du fichier XML. Les éléments ne sont pas accessibles par des méthodes comme `getElementById`, `getElementsByTagName` ou équivalentes.

4. Propriété responseXML

La propriété responseXML renvoie les données comme un objet document XML. Il pourra alors être examiné et traité selon les méthodes et propriétés du Document Object Model (DOM).

Soit avec le fichier XML suivant :

```
<?xml version="1.0"?>
<racine>
Je suis un texte
</racine>
```

L'élément "Je suis un test" est accessible par le code :

```
var xmldoc = xhr.responseXML;
var root_node = xmldoc.getElementsByTagName('racine')[0];
alert(root_node.firstChild.nodeValue);
```

Nous reviendrons plus longuement sur l'accès aux données XML et leur traitement dans le chapitre suivant consacré à AJAX.

Il est à remarquer lors de l'étude de ce chapitre que la méthode `XMLHttp- Request` porte finalement assez mal son nom car elle ne traite pas que du XML mais aussi du texte et tous ses dérivés comme des fichiers Xhtml ou Html, des fichiers JavaScript ou tout autre code.

Exemple

Au clic d'un bouton, affichez dans une boîte d'alerte le texte du fichier test.xml.

Le fichier test.xml est identique à celui du point précédent.

Le code du fichier htm est :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function getxhr(){
var xhr = null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200) {
alert(xhr.responseXML);
}
}
xhr.open("GET", "test.xml", true);
xhr.send(null);
}
//]]>
</script>
</head>
<body>
<form>
<input type="button" value="Test" onclick="getxhr()" />
</form>
</body>
</html>
```

Avec responseXML, le fichier XML est bien retourné comme un objet.

Quelques méthodes détaillées

1. Méthode getAllResponseHeaders

Pour ce qui suit, il faut bien distinguer un en-tête HTTP et un en-tête de fichier Html. Les balises comprises dans un en-tête Html, soit entre les balises `<head> ... </head>` envoient des informations concernant la page Html, tandis que les en-têtes HTTP envoient des informations relatives au fichier.

L'en-tête HTTP est un court message que le serveur Web envoie au navigateur juste avant de lui transmettre le document lui-même. Ce message sert par exemple à donner la taille du document ou à indiquer qu'il a disparu (code 404). Il peut aussi servir à préciser l'encodage du fichier grâce à la ligne Content-Type.

Il faut aussi noter qu'avec une requête HTTP, le serveur renvoie les en-têtes de la ressource ou du fichier, pas le fichier lui-même. Cela signifie que des renseignements sur un document comme Content-type ou last-modified sont connus sans avoir à télécharger le document lui-même.

La méthode `getAllResponseHeaders()` renvoie tous les en-têtes HTTP de la réponse sous la forme d'une chaîne de caractères.

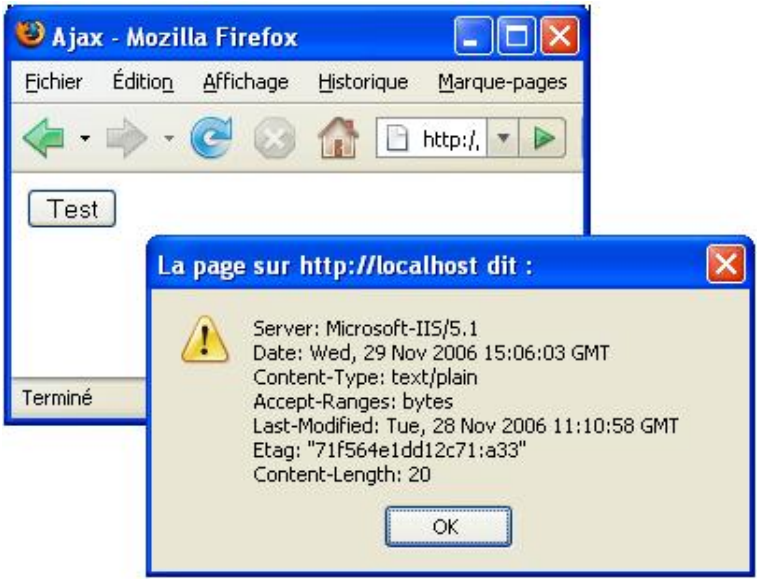
Exemple

Au clic d'un bouton, afficher dans une boîte d'alerte, tous les en-têtes du fichier test.txt.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html" />
<script type='text/JavaScript'>
//<![CDATA[
function getxhr(){
var xhr = null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200) {
alert(xhr.getAllResponseHeaders());
}
}
xhr.open("GET","test.txt",true);
xhr.send(null);
}
//]]>
</script>
</head>
<body>
<form>
<input type="button" value="Test" onclick="getxhr()" />
</form>
</body>
</html>
```

Sous Internet Explorer :

Sous Firefox :



2. Méthode getResponseHeader

La méthode `getResponseHeader()` renvoie la valeur d'un seul champ de l'entête HTTP sous forme d'une chaîne de caractères.

Elle prend la forme de `getResponseHeader("nom du paramètre d'en-tête HTTP")`.

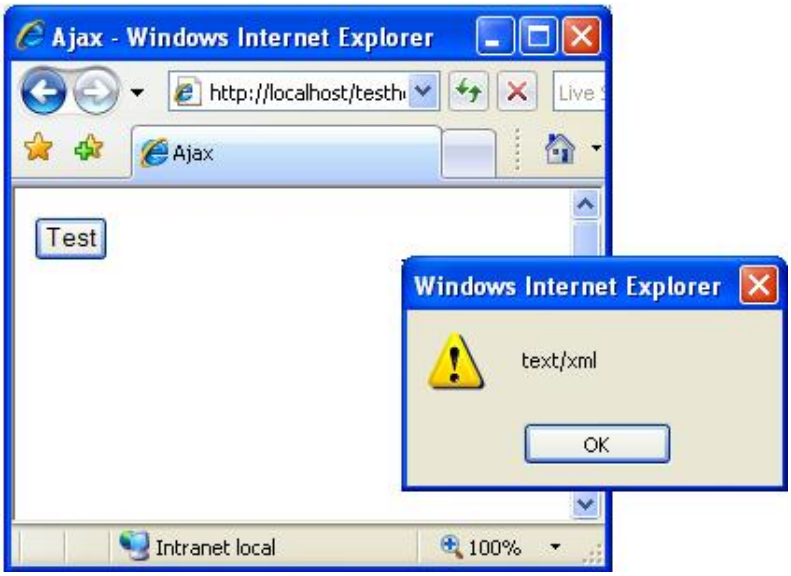
Soit, `xhr.getResponseHeader("Content-Type")` qui ne retourne que les renseignements concernant le type de contenu (Content-Type).

Exemple

Au clic d'un bouton, afficher dans une boîte d'alerte, l'en-tête de contenu de fichier (Content-type) du fichier test.xml :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function getxhr(){
```

```
var xhr = null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200) {
alert(xhr.getResponseHeader("Content-type"));
}
}
xhr.open("GET","test.xml",true);
xhr.send(null);
}
//]]>
</script>
</head>
<body>
<form>
<input type="button" value="Test" onclick="getxhr()" />
</form>
</body>
</html>
```



3. Méthode setRequestHeader

Cette méthode assigne une valeur à un champ d'en-tête HTTP qui est envoyée lors de la requête.

Elle prend la forme de setRequestHeader("nom du paramètre d'en-tête"," valeur associée au paramètre").

Elle est spécialement utilisée lors d'un envoi avec la méthode POST pour spécifier l'encodage :

```
xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
```

4. Méthode overrideMimeType

Cette méthode force un document à être traité selon un type de contenu particulier (Mime Type). On appelle souvent cette méthode lors de l'utilisation de responseXML sur un serveur transmettant du XML avec un en-tête Content-Type incorrect.

```
xhr.req.overrideMimeType('text/xml');
```

Introduction

L'approche AJAX<$I[] a été définie au chapitre Présentation génerale d'AJAX, comme l'utilisation conjointe du JavaScript, du XML, du XSL, du DOM et de l'objet XMLHttpRequest. Le présent chapitre a comme objectif de réaliser la synthèse des différents composants étudiés pour aboutir à des applications AJAX.

Rappelons également que la définition d'AJAX (JavaScript et XML en mode asynchrone) ne doit pas être prise au pied de la lettre car l'objet XMLHttpRequest permet de prendre en charge non seulement du XML mais également du texte brut.

Récupérer et traiter du texte

C'est assurément la façon la plus simple d'appréhender le concept AJAX. Elle reste néanmoins très riche au niveau des possibilités offertes aux développeurs.

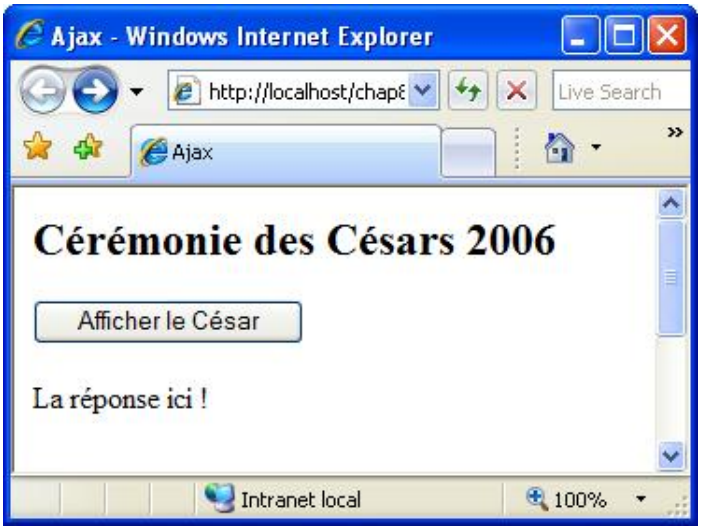
Affichons, de façon asynchrone, une réponse sous forme de texte dans une zone de la page par la propriété responseText.

Soit une page Xhtml. Au clic de l'utilisateur sur le bouton de formulaire, la réponse est affichée dans l'élément `<div id=" affichage "> ... </div>`.

Le fichier texte comporte les mots : "César du meilleur film français : Jacques Audiard pour "De battre mon cœur s'est arrêté". Il est enregistré sur le serveur sous le nom de cesar2006.txt.

Le fichier Xhtml se présente comme suit :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
</head>
<body>
<h2>Cérémonie des Césars 2006</h2>
<form action="">
<input type="button" value="Afficher le César" onclick="extraire()" />
</form>
<div id="affichage">
La réponse ici !
</div>
</body>
</html>
```



Élaborons le script pas à pas.

Au clic sur le bouton, la fonction *extraire()* est appelée.

```
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
```

```
}
```

Après avoir défini la variable `xhr` comme une variable globale, le script définit la requête HTTP vers le serveur de façon compatible avec les différents navigateurs. Ces lignes de script ont été largement abordées au chapitre L'objet XmlHttpRequest - Créer un objet XMLHttpRequest.

```
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var txtdocument = xhr.responseText;
afficher(txtdocument);
}
}
xhr.open("GET", "cesar2006.txt", true);
xhr.send(null);
}
}
```

La requête doit aller rechercher selon la méthode GET, le fichier cesar2006.txt de façon asynchrone (voir chapitre L'objet XmlHttpRequest - Effectuer une requête).

Au changement d'état de la requête (`onreadystatechange`), on s'assure tout d'abord, comme étudié au chapitre L'objet XmlHttpRequest - Quelques propriétés détaillées qu'elle a bien abouti (`readyState == 4 et status == 200`). Le fichier est renvoyé simplement comme un fichier texte par responseText (cf chapitre L'objet XmlHttpRequest - Quelques propriétés détaillées). Le script passe alors la main à la fonction *afficher*() avec le fichier texte en argument (`afficher (txtdocument)`).

```
function afficher(txtdocument) {
var target = document.getElementById("affichage");
target.innerHTML = txtdocument;
```

La fonction `afficher`() repère tout d'abord la zone d'affichage pour la réponse, par son identifiant `getElementById ("affichage");`. La cible est ainsi déterminée, et la réponse est alors affichée par la propriété inneHTML.

Le script complet devient donc :

```
<script type='text/JavaScript'>
//<![CDATA[
function afficher(txtdocument) {
var target = document.getElementById("affichage");
target.innerHTML = txtdocument;
}
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var txtdocument = xhr.responseText;
afficher(txtdocument);
}
}
xhr.open("GET", "cesar2006.txt", true);
xhr.send(null);
}
}
//]]>
</script>
```

Le document Xhtml devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function afficher(txtdocument) {
var target = document.getElementById("affichage");
target.innerHTML = txtdocument;
}
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var txtdocument = xhr.responseText;
afficher(txtdocument);
}
}
xhr.open("GET", "cesar2006.txt", true);
xhr.send(null);
}
}
//]]>
</script>
</head>
<body>
<h2>Cérémonie des Césars 2006</h2>
<form action="">
<input type="button" value="Afficher le César" onclick="extraire()" />
</form>
<div id="affichage">
La réponse ici !
</div>
</body>
</html>
```

Le fichier se présente dès lors comme suit à l'écran :

Diverses sources recommandent d'afficher la réponse, fournie par l'approche Ajax, en évidence, à l'intention des internautes débutants qui pourraient être pertubés dans leurs habitudes de navigation, ce qui peut se réaliser aisément par une feuille de style CSS.

```
<style type="text/css">
<!--
#affichage {
width: 250px;
font-family: sans-serif;
font-weight: bold;
font-size: 16px;
margin: 10px;
padding: 5px;
background-color: #9cf;
}
-->
</style>
```

Ainsi la zone d'affichage possède un arrière-plan de couleur et est délimitée par une bordure. En outre la police de caractères est différente.

Le fichier Xhtml final prend la forme suivante :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<<meta http-equiv="Content-Style-Type" content="text/css" />
<script type='text/JavaScript'>
//<![CDATA[
function afficher(txtdocument) {
var target = document.getElementById("affichage");
target.innerHTML = txtdocument;
}
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
```

```
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var txtdocument = xhr.responseText;
afficher(txtdocument);
}
}
xhr.open("GET", "cesar2006.txt", true);
xhr.send(null);
}
}
//]]>
</script>
<style type="text/css">
<!--
#affichage {
width: 250px;
font-family: sans-serif;
font-weight: bold;
font-size: 16px;
margin: 10px;
padding: 5px;
background-color: #9cf;
}
-->
</style>
</head>
<body>
<h2>Cérémonie des Césars 2006</h2>
<form action="">
<input type="button" value="Afficher le César" onclick="extraire()" />
</form>
<div id="affichage">
La réponse ici !
</div>
</body>
</html>
```

Il ne faut pas s'arrêter à une interprétation limitative de la propriété *responseText*. En effet, cela signifie que la réponse est renvoyée sous forme de texte au sens informatique du terme. Tous les types de format sont acceptés ; des sfichiers Xhtml, des données traitées par du PHP en passant par du code JavaScript. Le tout dépend de l'utilisation ultérieure de ces données textuelles.

Soit un fichier js.txt qui contient le texte : alert("Message d'alerte provenant du serveur").

Prenons un fichier Xhtml (js.htm). Ce fichier ne contient aucune instruction *alert()*. Pourtant une boîte d'alerte peut être créée en exécutant le code JavaScript à partir de la chaîne de caractères contenue dans le fichier js.txt. Ce qui est

effectué par la méthode *eval()*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function getxhr(){
var xhr = null;
if(window.XMLHttpRequest){
var xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200) {
eval(xhr. responseText);
}
}
xhr.open("GET", "js.txt", true);
xhr.send(null);
}
//]]>
</script>
</head>
<body>
<form action="">
<input type="button" value="Test" onclick="getxhr()" />
</form>
</body>
</html>
```

Une boîte d'alerte s'affiche donc, alors qu'elle n'était pas présente dans le code de la page Xhtml.

Récupérer et traiter du XML

1. Par les nœuds - Internet Explorer

Soit le fichier cesar2006.xml :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cinema>
<evenement>Cérémonie des Césars</evenement>
<sujet>Palmarès</sujet>
<date>2006</date>
<cesar>
<categorie>
<prix>César du meilleur acteur</prix>
<nom>Michel Bouquet</nom>
<film>Le promeneur du Champ de Mars</film>
</categorie>
<categorie>
<prix>César du meilleur film français</prix>
<nom>Jacques Audiard</nom>
<film>De battre mon coeur s'est arrêté</film>
</categorie>
<categorie>
<prix>César du meilleur film étranger</prix>
<nom>Clint Eastwood</nom>
<film>Million dollar baby</film>
</categorie>
</cesar>
</cinema>
```

Sa structure est la suivante :

```
<cinema>
      <evenement></evenement>
      <sujet></sujet>
      <date></date>
      <cesar>
         <categorie>
             <prix></prix>
             <nom></nom>
             <film></film>
         </categorie>
      </cesar>
</cinema>
```

On souhaite, au clic d'un bouton, extraire les données incluses dans la troisième balise `<categorie>`. Soient les informations :

```
César du meilleur film étranger
Clint Eastwood
Million dollar baby
```

Le fichier Xhtml de départ comporte simplement un bouton de formulaire et une balise `<div id="affichage">` servant de conteneur pour la réponse.

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<h2>Cérémonie des Césars 2006</h2>
<form action="">
```

```
<input type="button" value="Afficher le César" onclick="extraire()" />
</form>
<div id="affichage">
La réponse ici !
</div>
</body>
</html>
```

Étudions le script en détail.

```
<script type='text/JavaScript'>
//<![CDATA[
```

Le script commence par l'initialisation de l'objet XmlHttpRequest ou du contrôle ActiveX correspondant, comme cela a été étudié au chapitre L'objet XmlHttpRequest - Créer un objet XmlhttpRequest de cet ouvrage.

```
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
```

Si l'objet xhr existe, le script lance la requête HTTP en veillant à utiliser la propriété *responseXML* pour récupérer le fichier *cesar2006.xml*. Celui-ci est alors transmis à la fonction *afficher()*.

```
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
afficher(xmldocument);
}
}
xhr.open("GET", "cesar2006.xml", true);
xhr.send(null);
}
}
```

La fonction *afficher()* doit tout d'abord, parcourir l'arborescence du fichier XML pour retrouver l'information souhaitée. Nous utilisons pour ce faire, les acquis du chapitre Le DOM (Document Object Model).

Ce document XML a une balise racine appelée `<cinema>`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cinema>
.
.
.
</cinema>
```

On accède à ce nœud racine avec la propriété JavaScript*documentElement*.

```
function afficher(xmldocument) {
var cinema;
cinema = xmldocument.documentElement;
.
.
}
```

La variable `cinema` contient maintenant l'arborescence du fichier XML avec tous ses éléments enfants.

Le nœud correspondant à la balise `<cesar>` est atteint par la propriété *lastChild* du nœud racine soit la variable `cinema`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<cinema>
<evenement>Cérémonie des Césars</evenement>
<sujet>Palmarès</sujet>
<date>2006</date>
<cesar>
.
.
.
</cesar>
</cinema>
```

La fonction `afficher()` devient :

```
function afficher(xmldocument) {
var cinema, cesar, categorie;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild;
categorie = cesar.lastChild;
.
.
}
```

Il nous faut ensuite accéder à la troisième balise `<categorie>`. On utilise à nouveau la propriété *lastChild* mais cette fois appliquée au nœud cesar.

```
<?xml version="1.0" encoding="ISO-88 59-1"?>
<cinema>
<evenement>Cérémonie des Césars</evenement>
<sujet>Palmarès</sujet>
<date>2006</date>
<cesar>
<categorie>
<prix>César du meilleur acteur</prix>
<nom>Michel Bouquet</nom>
<film>Le promeneur du Champ de Mars</film>
</categorie>
<categorie>
<prix>César du meilleur film français</prix>
<nom>Jacques Audiard</nom>
<film>De battre mon coeur s'est arrêté</film>
</categorie>
<categorie>
<prix>César du meilleur film étranger</prix>
<nom>Clint Eastwood</nom>
<film>Million dollar baby</film>
</categorie>
</cesar>
</cinema>
```

La fonction `afficher()` devient :

```
function afficher(xmldocument) {
var cinema, cesar, categorie;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild;
categorie = cesar.lastChild;
.
.
}
```

Il est possible maintenant d'accéder à la balise `<prix>` en utilisant la propriété *firstChild* de la troisième balise `<categorie>` ou de notre variable `categorie`.

```
<categorie>
<prix>César du meilleur film étranger</prix>
<nom>Clint Eastwood</nom>
<film>Million dollar baby</film>
</categorie>
```

La fonction `afficher()` devient :

```
function afficher(xmldocument) {
var cinema, cesar, categorie;
var prix;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild;
categorie = cesar.lastChild;
prix = categorie.firstChild;
.
.
}
```

Nous pouvons à ce stade extraire le texte associé à la balise `<prix>`.

```
<prix>César du meilleur film étranger</prix>
```

Le nœud texte "César du meilleur film étranger" est un élément enfant de la balise `<prix>`. La propriété *firstChild* de la variable *prix* permet d'y accéder. Le texte lui-même est récupéré par la propriété *nodeValue*.

Le script devient :

```
function afficher(xmldocument) {
var cinema, cesar, categorie;
var prix, textediv;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild;
categorie = cesar.lastChild;
prix = categorie.firstChild;
textediv = prix.firstChild.nodeValue;
.
.
}
```

Il suffit alors d'afficher (*innerHTML*) ce texte dans la balise `<div id="affichage"> ... </div>`, repérée dans le document par la méthode `getElementById`.

```
function afficher(xmldocument) {
var cinema, cesar, categorie;
var prix, textediv;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild;
categorie = cesar.lastChild;
prix = categorie.firstChild;
textediv = prix.firstChild.nodeValue;
var target = document.getElementById("affichage");
target.innerHTML = textediv;
.
.
}
```

Procédons de même pour la balise `<nom>`, elle est atteinte par la propriété *next-Sibling* appliquée au nœud *prix*. Le nœud texte "Clint Eastwood" est un élément enfant de la balise `<nom>`. On peut y accéder par la propriété *firstChild* de la variable `nom`. Le texte lui-même est récupéré par la propriété *nodeValue*.

```
<nom>Clint Eastwood</nom>
```

Le code se complète alors comme suit :

```
function afficher(xmldocument) {
var cinema, cesar, categorie;
var prix, nom, textediv;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild;
categorie = cesar.lastChild;
prix = categorie.firstChild;
nom = prix.nextSibling;
```

```
textediv = (prix.firstChild.nodeValue + "<br />" +
nom.firstChild.nodeValue);
var target = document.getElementById("affichage");
target.innerHTML = textediv;
.
.
}
```

Nous pouvons terminer par la balise <film> atteinte en utilisant la propriété *lastChild* de la balise <categorie>.

```
<film>Million dollar baby</film>
```

Le nœud texte "Million dollar baby" est un élément enfant de la balise <film> accédée par la propriété firstChild de la variable film. Le texte lui-même est récupéré par la propriété *nodeValue*.

Le code final de la fonction *afficher()* est :

```
function afficher(xmldocument) {
var cinema, cesar, categorie;
var prix, nom, film, textediv;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild;
categorie = cesar.lastChild;
prix = categorie.firstChild;
nom = prix.nextSibling;
film = categorie.lastChild;
textediv = (prix.firstChild.nodeValue + "<br />" +
nom.firstChild.nodeValue + "<br />" + film.firstChild.nodeValue);
var target = document.getElementById("affichage");
target.innerHTML = textediv;
}

//]]>
</script>
```

Le script est terminé.

Le code complet du script est :

```
<script type='text/JavaScript'>
//<![CDATA[
function afficher(xmldocument) {
var cinema, cesar, categorie;
var prix, nom, film, textediv;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild;
categorie = cesar.lastChild;
prix = categorie.firstChild;
nom = prix.nextSibling;
film = categorie.lastChild;
textediv = (prix.firstChild.nodeValue + "<br />" +
nom.firstChild.nodeValue + "<br />" + film.firstChild.nodeValue);
var target = document.getElementById("affichage");
target.innerHTML = textediv;
}
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
```

```
var xmldocument = xhr.responseXML;
afficher(xmldocument);
}
}
xhr.open("GET", "cesar2006.xml", true);
xhr.send(null);
}
}
//]]>
</script>
```

Le fichier Xhtml se présente alors comme suit :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function afficher(xmldocument) {
var cinema, cesar, categorie;
var prix, nom, film, textediv;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild;
categorie = cesar.lastChild;
prix = categorie.firstChild;
nom = prix.nextSibling;
film = categorie.lastChild;
textediv = (prix.firstChild.nodeValue + "<br />" +
nom.firstChild.nodeValue + "<br />" + film.firstChild.nodeValue);
var target = document.getElementById("affichage");
target.innerHTML = textediv;
}

var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
afficher(xmldocument);
}
}
xhr.open("GET", "cesar2006.xml", true);
xhr.send(null);
}
}
//]]>
</script>
</head>
<body>
<h2>Cérémonie des Césars 2006</h2>
<form action="">
<input type="button" value="Afficher le César" onclick="extraire()" />
</form>
<div id="affichage">
```

```
La réponse ici !
</div>
</body>
</html>
```

À ce stade de notre travail, la capture d'écran est la suivante :

Ajoutons une feuille de style afin d'égayer un peu la présentation.

```
<style>
<!--
#affichage {
width: 250px;
height: 50px;
font-family: sans-serif;
font-variant: small-caps;
font-size: 14px;
line-height: 18px;
background-color: #9cf;
padding: 5px;
-->
</style>
```

Le fichier Xhtml final est le suivant :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<script type='text/JavaScript'>
//<![CDATA[
function afficher(xmldocument) {
var cinema, cesar, categorie;
var prix, nom, film, textediv;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild;
categorie = cesar.lastChild;
prix = categorie.firstChild;
nom = prix.nextSibling;
film = categorie.lastChild;
textediv = (prix.firstChild.nodeValue + "<br />" +
nom.firstChild.nodeValue + "<br />" + film.firstChild.nodeValue);
var target = document.getElementById("affichage");
```

```
target.innerHTML = textediv;
}
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n’est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
afficher(xmldocument);
}
}
xhr.open("GET", "cesar2006.xml", true);
xhr.send(null);
}
}
//]]>
</script>
<style>
<!--
#affichage {
width: 250px;
height: 50px;
font-family: sans-serif;
font-variant: small-caps;
font-size: 14px;
line-height: 18px;
background-color: #9cf;
padding: 5px;
-->
</style>
</head>
<body>
<h2>Cérémonie des Césars 2006</h2>
<form action="">
<input type="button" value="Afficher le César" onclick="extraire()" />
</form>
<div id="affichage">
La réponse ici !
</div>
</body>
</html>
```

2. Par les nœuds - Firefox

Firefox et les autres navigateurs de la famille Mozilla traitent tous les espaces vides ainsi que les indentations du code XML comme des nœuds de texte vide (voir chapitre Le DOM (Document Object Model) - Particularité de Firefox).

Reprenons notre fichier cesar2006.xml.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cinema>
<evenement>Cérémonie des Césars</evenement>
<sujet>Palmarès</sujet>
<date>2006</date>
<cesar>
<categorie>
<prix>César du meilleur acteur</prix>
<nom>Michel Bouquet</nom>
<film>Le promeneur du Champ de Mars</film>
</categorie>
<categorie>
<prix>César du meilleur film français</prix>
<nom>Jacques Audiard</nom>
<film>De battre mon coeur s'est arrêté</film>
</categorie>
<categorie>
<prix>César du meilleur film étranger</prix>
<nom>Clint Eastwood</nom>
<film>Million dollar baby</film>
</categorie>
</cesar>
</cinema>
```

Pour Internet Explorer, le nœud racine est `<cinema>`. Le premier nœud enfant (*firstChild*) est la balise `<evenement>`. Le second nœud enfant est la balise `<sujet>` et ainsi de suite.

Les choses sont différentes avec Firefox. Pour ce navigateur, le nœud racine est toujours `<cinema>` mais le premier élément enfant est un nœud texte vide qui inclut le passage à la ligne après la balise `<cinema>`. Ainsi pour Firefox, le fichier XML se présente comme suit :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cinema>
  nœud texte
<evenement>Cérémonie des Césars</evenement>
nœud texte
<sujet>Palmarès</sujet>
  nœud texte
<date>2006</date>
```

```
nœud texte
<cesar>
nœud texte
<categorie>
    nœud texte
<prix>César du meilleur acteur</prix>
  nœud texte
<nom>Michel Bouquet</nom>
  nœud texte
<film>Le promeneur du Champ de Mars</film>
nœud texte
</categorie>
.
.
```

Il faudra tenir compte de cette particularité dans l'élaboration du code.

Commencons par accéder à l'élément racine du document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cinema>
.
.
.
</cinema>
```

Tout comme pour Internet Explorer, la propriété document.Element de l'objet document XML est reprise.

```
function affiche(xmldocument) {
var cinema;
cinema = xmldocument.documentElement;
.
.
}
```

Les choses se compliquent pour atteindre la balise `</cesar>`. On pourrait croire que la balise `</cesar>` est le dernier enfant (*lastChild*) de la balise `<cinema>`. Mais pour Firefox, le dernier enfant de la balise `<cinema>` est un nœud texte qui suit la balise `</cesar>`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cinema>
  nœud texte
<evenement>Cérémonie des Césars</evenement>
nœud texte
<sujet>Palmarès</sujet>
  nœud texte
<date>2006</date>
nœud texte
<cesar>
  .
.
    .
  </cesar>
  nœud texte
</cinema>
```

Ainsi, au lieu d'utiliser `cinema.lastChild`, il faut remonter d'un cran dans l'arborescence pour éviter le nœud texte et accéder à l'élément `<cesar> ... </cesar>`, ce qui peut se réaliser avec la propriété `previousSibling`.

```
function affiche(xmldocument) {
var cinema, cesar;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild.previousSibling;
.
.
}
```

Pour les mêmes raisons, la troisième balise `<categorie>` ne peut pas être atteinte par la propriété lastChild appliquée à

l'élément <cesar>.

```
<cesar>
.
.
.
<categorie>
     nœud texte
<prix>César du meilleur film étranger</prix>
nœud texte
<nom>Clint Eastwood</nom>
nœud texte
<film>Million dollar baby</film>
nœud texte
</categorie>
nœud texte
</cesar>
```

Elle est atteinte par `categorie = cesar.lastChild.previousSibling`.

Et ainsi de suite ...

Le code complet pour Firefox devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function afficher(xmldocument) {
var cinema, cesar, categorie;
var prix, nom, film, textediv;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild.previousSibling;
categorie = cesar.lastChild.previousSibling;
prix = categorie.firstChild.nextSibling;
nom = prix.nextSibling.nextSibling;
film = categorie.lastChild.previousSibling;
textediv = (prix.firstChild.nodeValue + "<br />" +
nom.firstChild.nodeValue + "<br />" + film.firstChild.nodeValue);
var target = document.getElementById("affichage");
target.innerHTML = textediv;
}
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
afficher(xmldocument);
}
}
xhr.open("GET", "cesar2006.xml", true);
xhr.send(null);
}
}
```

```
//]]>
</script>
</head>
<body>
<h2>Cérémonie des Césars 2006</h2>
<form action="">
<input type="button" value="Afficher le César" onclick="extraire()" />
</form>
<div id="affichage">
La réponse ici !
</div>
</body>
</html>
```

Cela fonctionne correctement en utilisant le navigateur Firefox, mais le script est compliqué à mettre en place. Non seulement, il faut "sauter" les nœuds texte mais en plus, le code pour Firefox diffère de celui d'Internet Explorer. La solution est un script compatible pour les deux navigateurs.

3. Par les nœuds - solution compatible

Pour rendre le script compatible, il suffit, avant toute manipulation de code, d'enlever les espaces vides du document XML. Ainsi, ces problématiques nœuds de texte vide introduits par Firefox disparaissent et le même code peut être utilisé pour naviguer dans le document que l'on utilise Firefox ou bien Internet Explorer.

Nous allons élaborer une fonction enleverespaces() qui permet de supprimer les espaces vides dans le code XML.

Au début de cette fonction, une boucle for permet de passer en revue les différents nœuds enfant grâce à la propriété *childNodes* (voir chapitre Le DOM (Document Object Model) - Propriétés de l'Objet Node).

```
function enleverespaces(xmldocument) {
var index;
for (index = 0; index < xmldocument.childNodes.length; index++) {
var noeudc = xml.childNodes(index);
.
.
}
}
```

À cette étape de la boucle, le nœud courant est stocké dans la variable nœudc.

S'il s'agit d'un nœud élément (`noeudc.nodeType == 1`), cet élément peut, à son tour, avoir des nœuds enfant pour lesquels il faut enlever les espaces vides. Dans ce cas, ce nœud courant est renvoyé à la fonction `enleverespaces()` en le fournissant comme argument de la fonction, soit `enleverespaces(noeudc)`.

```
function enleverespaces(xmldocument) {
var index;
```

```
for (index = 0; index < xmldocument.childNodes.length; index++) {
var noeudc = xmldocument.childNodes[index];
if (noeudc.nodeType == 1) {
enleverespaces(noeudc);
}
.
.
}
}
```

Par ailleurs, si le nœud courant est un nœud texte (`noeudc.nodeType == 3`), il s'agit peut-être d'un espace vide. Nous allons tester le texte présent dans le nœud (propriété *nodeValue*) au moyen d'une expression régulière(voir chapitre Le JavaScript - Manipulation des chaînes de caractères) qui permet de déceler les espaces vides, soit l'instruction (`/^\s+$/.test(noeudc.nodeValue)`).

Nous allons donc vérifier si le nœud courant comporte des espaces vides et est un nœud texte (`if ((/^\s+$/.test (noeudc.nodeValue)) && (noeudc. nodeType == 3))`). Dans l'affirmative, il suffit alors d'appliquer la propriété removeChild pour enlever ce nœud. Il ne faut pas oublier de décrémenter l'index d'une position.

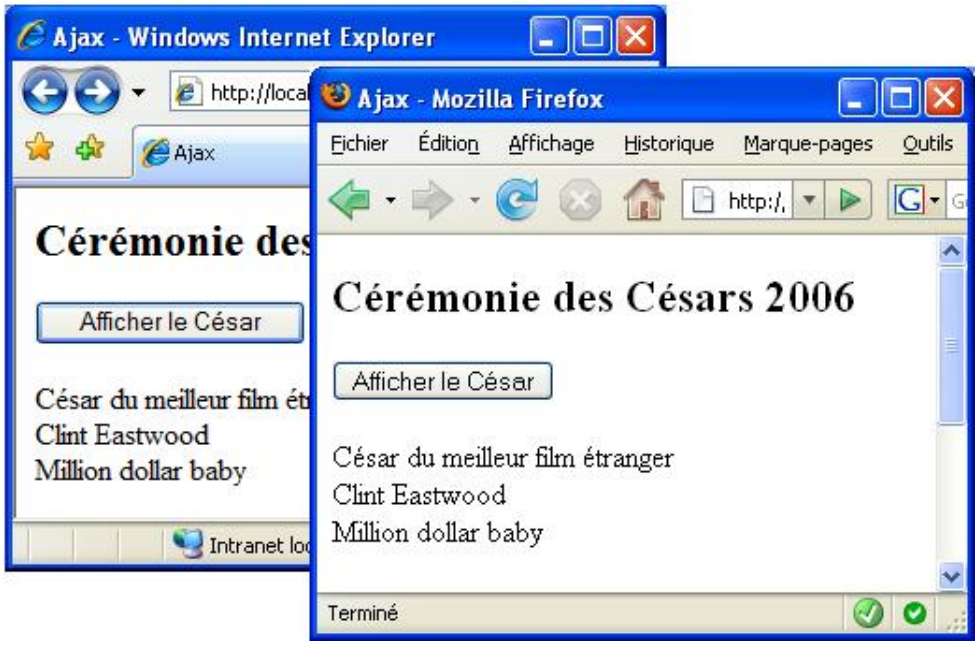
```
function enleverespaces(xmldocument) {
var index;
for (index = 0; index < xmldocument.childNodes.length; index++) {
var noeudc = xmldocument.childNodes[index];
if (noeudc.nodeType == 1) {
enleverespaces(noeudc);
}
if ((/^\s+$/.test(noeudc.nodeValue)) && (noeudc.nodeType == 3)) {
xmldocument.removeChild(xmldocument.childNodes[index--]);
}
}
}
```

Pour rendre le script compatible avec les navigateurs, cette fonction est appelée pour éliminer les espaces vides du code XML pour Firefox. L'incompatibilité introduite par Firefox ayant disparu, il est dès lors possible d'utiliser le même code pour Firefox et Internet Explorer.

Le fichier Xhtml compatible se présente comme suit :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function enleverespaces(xmldocument) {
var index;
for (index = 0; index < xmldocument.childNodes.length; index++) {
var noeudc = xmldocument.childNodes[index];
if (noeudc.nodeType == 1) {
enleverespaces(noeudc);
}
if ((/^\s+$/.test(noeudc.nodeValue)) && (noeudc.nodeType == 3)) {
xmldocument.removeChild(xmldocument.childNodes[index--]);
}
}
}
function afficher(xmldocument) {
var cinema, cesar, categorie;
var prix, nom, film, textediv;
cinema = xmldocument.documentElement;
cesar = cinema.lastChild;
categorie = cesar.lastChild;
prix = categorie.firstChild;
nom = prix.nextSibling;
film = categorie.lastChild;
textediv = (prix.firstChild.nodeValue + "<br />" +
```

```
nom.firstChild.nodeValue + "<br />" + film.firstChild.nodeValue);
var target = document.getElementById("affichage");
target.innerHTML = textediv;
}
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
if(navigator.userAgent.indexOf("Firefox") != -1) {
enleverespaces(xmldocument);
}
afficher(xmldocument);
}
}
xhr.open("GET", "cesar2006.xml", true);
xhr.send(null);
}
}
//]]>
</script>
</head>
<body>
<h2>Cérémonie des Césars 2006</h2>
<form action="">
<input type="button" value="Afficher le César" onclick="extraire()" />
</form>
<div id="affichage">
La réponse ici !
</div>
</body>
</html>
```



4. Par la méthode GetElementsByTagName

Le procédé précédent permet d'accéder aux objets par les propriétés des nœuds, est certes élégant mais il faut bien admettre que les sauts successifs sur les éléments d'un document XML produit un code complexe. Il se révèle en outre peu pratique dans le cas d'une mise à jour de la page car le moindre élément ajouté ou retiré nécessiterait la réécriture complète du code.

L'utilisation de la méthode `getElementsByTagName` se révèle plus simple et plus fonctionnelle (chapitre Le DOM (Document Object Model) - Accèder aux noeuds).

Considérons toujours notre document *cesar2006.xml* : il s'agit toujours d'accéder aux informations des balises `<prix>`, `<nom>` et `<film>` de la troisième balise `<categorie>`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cinema>
<evenement>Cérémonie des Césars</evenement>
<sujet>Palmarès</sujet>
<date>2006</date>
<cesar>
<categorie>
<prix>César du meilleur acteur</prix>
<nom>Michel Bouquet</nom>
<film>Le promeneur du Champ de Mars</film>
</categorie>
<categorie>
<prix>César du meilleur film français</prix>
<nom>Jacques Audiard</nom>
<film>De battre mon coeur s'est arrêté</film>
</categorie>
<categorie>
<prix>César du meilleur film étranger</prix>
<nom>Clint Eastwood</nom>
<film>Million dollar baby</film>
</categorie>
</cesar>
</cinema>
```

Construisons le script.

```
<script type='text/JavaScript'>
//<![CDATA[
function afficher(xmldocument) {
noeudsprix = xmldocument.getElementsByTagName("prix");
noeudsnom = xmldocument.getElementsByTagName("nom");
noeudsfilm = xmldocument.getElementsByTagName("film");
.
.
.
}
```

Les variables `noeudsprix`, `noeudsnom` et `noeudsfilm` contiennent dans une liste toutes les balises `<prix>`, `<nom>` et `<film>` du document XML. Il suffit alors d'extraire la valeur (`nodeValue`) du premier élément enfant (`firstChild`) du troisième nœud `<prix>(noeudsprix[2])`. Bien entendu, les nœuds `<nom>` et `<film>` sont extraits de la même façon. Une fois ces informations recueillies, la propriété `innerHTML` permet l'affichage de celles-ci.

```
function afficher(xmldocument) {
noeudsprix = xmldocument.getElementsByTagName("prix");
noeudsnom = xmldocument.getElementsByTagName("nom");
noeudsfilm = xmldocument.getElementsByTagName("film");
var textediv = (noeudsprix[2].firstChild.nodeValue + "<br />" +
noeudsnom[2].firstChild.nodeValue + "<br />" +
noeudsfilm[2].firstChild.nodeValue);
var target = document.getElementById("affichage");
target.innerHTML = textediv;
}
```

Le code complet du document Xhtml avec la méthode `getElementsByTagName` devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function afficher(xmldocument) {
noeudsprix = xmldocument.getElementsByTagName("prix");
noeudsnom = xmldocument.getElementsByTagName("nom");
noeudsfilm = xmldocument.getElementsByTagName("film");
var textediv = (noeudsprix[2].firstChild.nodeValue + "<br />" +
noeudsnom[2].firstChild.nodeValue + "<br />" +
noeudsfilm[2].firstChild.nodeValue);
var target = document.getElementById("affichage");
target.innerHTML = textediv;
}
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre<+>navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
afficher(xmldocument);
}
}
xhr.open("GET", "cesar2006.xml", true);
xhr.send(null);
}
}
//]]>
</script>
</head>
<body>
<h2>Cérémonie des Césars 2006</h2>
<form action="">
<input type="button" value="Afficher le César" onclick="extraire()" />
</form>
<div id="affichage">
La réponse ici !
</div>
</body>
</html>
```

Le résultat est identique à celui de l'exploration du document par les propriétés des nœuds.

Ce code avec la méthode `getElementsByTagName` est parfaitement compatible pour Internet Explorer et Firefox.

Il vous est possible d'ajouter une feuille de style pour agrémenter la présentation comme cela a été réalisé aux points précédents.

5. Traitement des attributs

Un document XML peut également comporter des attributs de balises. Ceux-ci contiennent souvent des informations d'un intérêt non négligeable. Ainsi, il importe de pouvoir également y accéder.

Pour rappel (cf. chapitre Le DOM (Document Object Model) - Accéder aux attributs), nous utilisons la propriété *attributes* qui renvoie une liste de tous les attributs d'un élément spécifié. Cette liste des nœuds attributs est renvoyée sous forme d'un objet de type NamedNodeMap. Ce qui implique que les attributs sont par la suite, accessibles par leur nom.

Soit un document XML (stock.xml) qui comporte des attributs :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<stock>
<album>
<article="en stock">1001</article>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
</album>
<album>
<article etat="en stock">1002</article>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
</album>
<album>
<article etat="en rupture">1003</article>
<titre>Sheller en solitaire</titre>
<artiste>William Sheller</artiste>
</album>
<album>
<article etat="en rupture">1004</article>
<titre>Caché derrière</titre>
<artiste>Laurent Voulzy</artiste>
</album>
<album>
<article etat="en stock">1005</article>
<titre>Défoule sentimentale</titre>
<artiste>Alain Souchon</artiste>
</album>
</stock>
```

On souhaite savoir, après avoir consulté le fichier stock.xml, si l'article est en stock ou en rupture.

Le fichier Xhtml de départ présente la forme suivante :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<meta http-equiv="Content-Style-Type" content="text/javascript" />
</head>
<body>
<form action="">
Article : <input type="text" id="entree" size="4" />
<input type="button" value="Voir stock" onclick="extraire()" />
<input type="reset" value="Annuler" />
</form>
<div id="affichage">
La réponse ici !
</div>
</body>
</html>
```

Passons à la partie script.

```
<script type='text/JavaScript'>
//<![CDATA[
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
```

Au clic sur le bouton, la fonction *extraire()* est appelée. Elle crée un objet *XMLHttpRequest* ou un objet ActiveX pour rendre le script compatible.

```
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
afficher(xmldocument);
}
}
xhr.open("GET",<+>"stock.xml", true);
xhr.send(null);
}
}
```

Une requête asynchrone est effectuée sur le fichier stock.xml. Si l'opération s'est déroulée correctement, le script passe la main à la fonction *afficher()* prenant en argument le fichier XML (`afficher(xmldocument)`).

```
function afficher(xmldocument) {
var entree = document.getElementById("entree").value;
var target = document.getElementById("affichage");
var article = xmldocument.getElementsByTagName("article");
```

Différents objets sont alors déterminés. La variable `entree` reprend le contenu de la ligne de texte. La variable `target` accède à la zone d'affichage définie par la balise `<div id="affichage">` et enfin la variable article liste les balises `<article>` du document XML.

```
for (i=0; i<article.length; i++){
if(article[i].firstChild.nodeValue==entree){
var attributs=article[i].attributes;
var etatstock=attributs.getNamedItem("etat").nodeValue;
```

Ensuite, au moyen d'une boucle *for*, les différents éléments de la liste `article` sont passés en revue. Si le numéro de référence (`article[i]. firstChild.nodeValue`) contenu dans la balise `<article>` est égal à la valeur encodée (variable entree), une variable attributs est créée comprenant tous les attributs de la balise `<article>` retenue. Enfin,

il suffit d'aller chercher l'attribut etat (`attributs.getNamedItem("etat")`) et d'en prendre la valeur (`nodeValue`).

```
target.innerHTML="Cet article est " + etatstock;
}
}
}
//]]>
</script>
```

Le script se termine par l'affichage du message.

Le script complet est le suivant :

```
<script type='text/JavaScript'>
//<![CDATA[
function afficher(xmldocument) {
var entree = document.getElementById("entree").value;
var target = document.getElementById("affichage");
var article = xmldocument.getElementsByTagName("article");
for (i=0; i<article.length; i++){
if(article[i].firstChild.nodeValue==entree){
var attributs=article[i].attributes;
var etatstock=attributs.getNamedItem("etat").nodeValue;
target.innerHTML="Cet article est " + etatstock;
}
}
}
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
afficher(xmldocument);
}
}
xhr.open("GET", "stock.xml", true);
xhr.send(null);
}
}
//]]>
</script>
```

Le fichier Xhtml final devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function afficher(xmldocument) {
var entree = document.getElementById("entree").value;
var target = document.getElementById("affichage");
var article = xmldocument.getElementsByTagName("article");
for (i=0; i<article.length; i++){
```

```
if(article[i].firstChild.nodeValue==entree){
var attributs=article[i].attributes;
var etatstock=attributs.getNamedItem("etat").nodeValue;
target.innerHTML="Cet article est " + etatstock;
}
}
}
var xhr = null;
function extraire(){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
afficher(xmldocument);
}
}
xhr.open("GET", "stock.xml", true);
xhr.send(null);
}
}
//]]>
</script>
</head>
<body>
<form action="">
Article : <input type="text" id="entree" size="4" />
<input type="button" value="Voir stock" onclick="extraire()" />
<input type="reset" value="Annuler" />
</form>
<div id="affichage">
La réponse ici !
</div>
</body>
</html>
```

Récupérer et traiter avec XSL

Avec ce chapitre sur le traitement du XSL dans les applications AJAX, nous sommes à la limite du fonctionnement des navigateurs actuels et les solutions qu'ils proposent sont souvent spécifiques et donc hautement incompatibles.

Nous avons étudié au chapitre Introduction au XSL la liaison d'une feuille de style XSL (fichier .xsl) dans un document XML (fichier .xml).

Celle-ci se réalisait par une ligne de code dans le fichier XML. Pour rappel :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="xsldemo.xsl"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
Etc.
```

Ce système, bien que très puissant, ne prend cependant en charge que l'affichage du document XML dans un navigateur dont la vocation première est, rappelons-le, d'afficher du Html ou du Xhtml.

Il faut, avec le modèle AJAX, avoir la possibilité d'accéder aux éléments du fichier XML et XSL. Accessoirement, il faut aussi pouvoir changer de façon dynamique (par le JavaScript), la feuille de style associée au document XML.

Il a fallu donc mettre en place une technique différente de celle habituellement utilisée. C'est ce que nous allons aborder dans l'étude de ce sous-chapitre.

Nous allons utiliser comme fichier XML (xml.xml), le code suivant, déjà donné en exemple au chapitre Introduction au XSL .

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<compilation>
<mp3>
<titre>Sarbacane</titre>
<artiste>Francis Cabrel</artiste>
<date>1990</date>
</mp3>
<mp3>
<titre>Nickel</titre>
<artiste>Alain Souchon</artiste>
<date>1991</date>
</mp3>
<mp3>
<titre>Sheller en solitaire</titre>
<artiste>William Sheller</artiste>
<date>1992</date>
</mp3>
<mp3>
<titre>Caché derrière</titre>
<artiste>Laurent Voulzy</artiste>
<date>1993</date>
</mp3>
<mp3>
<titre>Rio Grande</titre>
<artiste>Eddy Mitchell</artiste>
<date>1994</date>
</mp3>
<mp3>
<titre>Samedi soir sur la Terre</titre>
<artiste>Francis Cabrel</artiste>
<date>1995</date>
</mp3>
<mp3>
<titre>Défoule sentimentale</titre>
<artiste>Alain Souchon</artiste>
<date>1996</date>
</mp3>
</compilation>
```

Et comme fichier de feuille de style XSL (fichier xsl.xsl) :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
<table border="1" style="border-collapse: collapse"
bordercolor="#000000" cellpadding="3">
<tr style="background: #9cf;">
<td>Année</td>
<td>Album</td>
<td>Artiste</td>
</tr>
<xsl:for-each select="compilation/mp3">
<tr>
<td><xsl:value-of select="date"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Il faut remarquer qu'il n'y a pas de liaison entre le fichier XML et le fichier XSL.

1. Solution pour Internet Explorer 7

La démarche se réalise toujours en trois étapes :

- Le chargement depuis le serveur du fichier XML.
- Le chargement depuis le serveur du fichier XSL.
- Le traitement pour associer le fichier XSL au fichier XML.

Pour réaliser le traitement, Microsoft s'est tourné vers les contrôles ActiveX de son moteur MSXML (voir chapitre L'objet XMLHttpRequest - Créer un objet XMLHttpRequest).

Dérivée de cette technologie, l'instruction *xml.transformNode(xsl)* permet d'associer de façon dynamique, une feuille de style XSL à un document XML. Il faut noter que ce code est spécifique à Microsoft et ne repond en rien aux spécifications du W3C.

Depuis qu'Internet Explorer 7 admet également l'objet XMLHttpRequest de façon native, un code peut être élaboré, proche de celui abordé aux points et chapitres précédents.

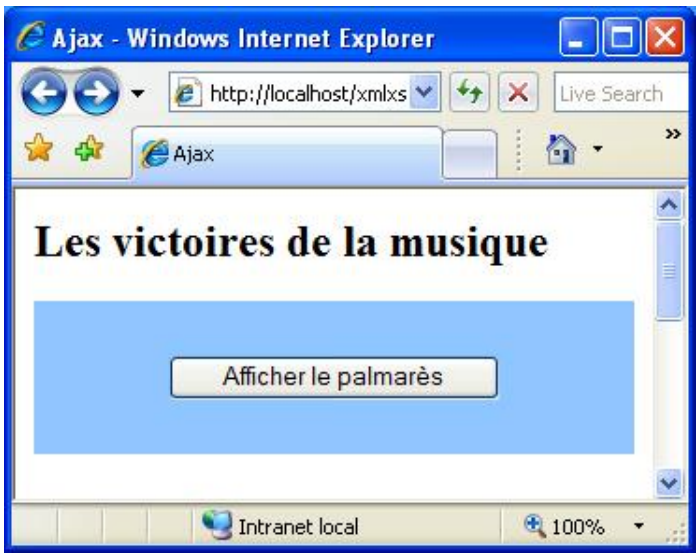
Soit notre fichier Xhtml de départ :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Script-Type" content="text/css" />
```

```
<style type="text/css">
<!--
#transform {background-color: #9cf;
            padding: 10px;
            text-align: center;}
-->
</style>
</head>
<body>
<h2>XML et XSL</h2>
<div id="transform">
<form action="">
<input type="button" value="Afficher le palmarès"
onclick="transform(xml, xsl, 'transform')" />
</form>
</div>
</body>
</html>
```

Commentaires

- Le script est appelé au clic sur le bouton du formulaire.
- Une déclaration de feuilles de style (*#transform*) met en avant la zone définie par la balise `<div id="transform"> ... </div>` dans laquelle se réalise la transformation.



Passons au script et à son étude.

```
<script type="text/javascript">
//<![CDATA[
var xml = loadXML('xml.xml');
var xsl = loadXML('xsl.xsl');
```

La fonction loadXML() est appelée une première fois pour charger le fichier XML (`'xml.xml'`) et une seconde fois pour charger le fichier XSL (`'xsl.xsl'`).

```
function loadXML(url){
var xhr;
xhr = new XMLHttpRequest();
xhr.open('GET', url, false);
xhr.send(null);
if (xhr.readyState == 4){
xhr = xhr.responseXML;
}
return xhr;
}
```

La fonction loadXML(url) charge le fichier XML ou XSL de façon asynchrone selon le code étudié précédemment.

```
function transform(xml, xsl, id){
if (window.ActiveXObject) {
var target = document.getElementById(id);
target.innerHTML = xml.transformNode(xsl);
}
}
```

La fonction *transform()*, appelée au clic du bouton, réalise la troisième étape en associant la feuille de style XSL au document XML (`xml.transform-Node(xsl);`).

Le nouveau fichier ainsi obtenu est affiché (`innerHTML`) dans la balise `<div id="transform">` par un appel à la méthode `getElementById`.

```
//]]>
</script>
```

Fin du script

Le fichier Xhtml complet devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Script-Type" content="text/css" />
<style type="text/css">
<!--
#transform {background-color: #9cf;
            padding: 10px;
            text-align: center;}
-->
</style>
<script type="text/javascript">
//<![CDATA[
var xml = loadXML('xml.xml');
var xsl = loadXML('xsl.xsl');
function loadXML(url){
var xhr;
xhr = new XMLHttpRequest();
xhr.open('GET', url, false);
xhr.send(null);
if (xhr.readyState == 4){
xhr = xhr.responseXML;
}
return xhr;
}
function transform(xml, xsl, id){
if (window.ActiveXObject) {
var target = document.getElementById(id);
target.innerHTML = xml.transformNode(xsl);
}
}
//]]>
</script>
</head>
<body>
<h2>Les victoires de la musique</h2>
<div id="transform">
<form action="">
<input type="button" value="Afficher le palmarès"
onclick="transform(xml, xsl, 'transform')" />
</form>
</div>
```

```
</body>
</html>
```

2. Solution pour Internet Explorer 6 (et précédents)

Cette solution se base uniquement sur des contrôles ActiveX du MSXML et de son compère XMLDOM.

Ce code est compatible pour Internet Explorer 6, Internet Explorer 5.5, Internet Explorer 5 et même pour internet Explorer 7. En effet, même si ce dernier prend en charge l'objet XMLHttpRequest de façon native, el reprend également le processus avec les contrôles Activex de ses prédécesseurs.

Ainsi, nous découvons un code différent de celui utilisé jusqu'à présent.

Le fichier Xhtml de départ n'est que faiblement modifié :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Script-Type" content="text/css" />
<style type="text/css">
<!--
#transform {background-color: #9cf;
            padding: 10px;
            text-align: center;}
-->
</style>
<script type="text/javascript">
//<![CDATA[
var xml = loadXML('xml.xml');
var xsl = loadXML('xsl.xsl');
function loadXML(url){
var xhr;
xhr = new XMLHttpRequest();
```

```
xhr.open('GET', url, false);
xhr.send(null);
if (xhr.readyState == 4){
xhr = xhr.responseXML;
}
return xhr;
}
function transform(xml, xsl, id){
if (window.ActiveXObject) {
var target = document.getElementById(id);
target.innerHTML = xml.transformNode(xsl);
}
}
//]]>
</script>
</head>
<body>
<h2>Les victoires de la musique</h2>
<div id="transform">
<form action="">
<input type="button" value="Afficher le palmarès"
onclick="transform(xml, xsl, 'transform')" />
</form>
</div>
</body>
</html>
```

En voici la capture d'écran sous Internet Explorer 6.

Passons au script et à son étude.

```
<script type="text/javascript">
//<![CDATA[
function transform(id){
var xml = new ActiveXObject("Microsoft.XMLDOM");
xml.async = false;
xml.load("xml.xml");
```

Le contrôle ActiveX est initialisé (`new ActiveXObject("Microsoft. XMLDOM")`).

Le mode synchrone généralement utilisé est fixé (`async = false`). Il semble cependant, selon la documentation de Microsoft, que le mode asynchrone pourrait être adopté. Puis la méthode load() prend en argument le fichier XML pour le charger (`load("xml.xml")`).

```
var xsl = new ActiveXObject("Microsoft.XMLDOM");
xsl.async = false;
xsl.load("xsl.xsl");
```

Le traitement du fichier XSL est identique.

```
var target = document.getElementById(id);
target.innerHTML = xml.transformNode(xsl);
}
```

Le fichier XML est transformé à partir du fichier XSL (`xml.transformNode(xsl)`) qui est affiché (`innerHTML`) dans la balise `<div id="transform"> ... </div>`.

```
//]]>
</script>
```

Fin de script.

Le fichier Xhtml final devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Script-Type" content="text/css" />
<style type="text/css">
<!--
#transform {background-color: #9cf;
            padding: 10px;
            text-align: center;}
-->
</style>
<script type="text/javascript">
//<![CDATA[
function transform(id){
var xml = new ActiveXObject("Microsoft.XMLDOM");
xml.async = false;
xml.load("xml.xml");
var xsl = new ActiveXObject("Microsoft.XMLDOM");
xsl.async = false;
xsl.load("xsl.xsl");
var target = document.getElementById(id);
target.innerHTML = xml.transformNode(xsl);
}
//]]>
</script>
</head>
<body>
<h2>Les victoires de la musique</h2>
<div id="transform">
<form action="">
<input type="button" value="Afficher le palmarès"
onclick="transform('transform')" />
</form>
</div>
</body>
</html>
```

3. Solution pour Firefox

Avec le navigateur Firefox (depuis la version Firefox 1.2), JavaScript peut exécuter des transformations XSLT à travers l'objet*XSLTProcessor*.

Un fois l'objet *XSLTProcessor* créé, il faut utiliser la méthode `importStylesheet(xsl)` pour réaliser la transformation. La feuille de style xsl doit être chargée dans le document par l'objet XMLHttpRequest avant d'appeler la méthode `importStylesheet()`. Nous allons y revenir plus en détail dans la partie script.

Voici le fichier Xhtml de départ (inchangé) :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Script-Type" content="text/css" />
<style type="text/css">
<!--
#transform {background-color: #9cf;
            padding: 10px;
            text-align: center;}
-->
</style>
</head>
<body>
<h2>Les victoires de la musique</h2>
<div id="transform">
<form action="">
<input type="button" value="Afficher le palmarès"
onclick="transform(xml, xsl, 'transform')" />
</form>
</div>
</body>
</html>
```

Le script devient :

```
<script type="text/javascript">
//<![CDATA[
var xml = loadXML('xml.xml');
var xsl = loadXML('xsl.xsl');
```

Les deux fichiers XML et XSL sont chargés en mémoire en faisant appel à la fonction `loadXML()`.

```
function loadXML(url) {
var xhr;
if (document.implementation.createDocument) {
xhr = new XMLHttpRequest();
xhr.open('GET', url, false);
xhr.send(null);
if (xhr.readyState == 4){
xhr = xhr.responseXML;
}
}
return xhr;
}
```

La fonction *loadXML()*, après avoir vérifié si le navigateur utilisé est bien Firefox (`if (document.implementation.createDocument)`) charge en mode synchrone les fichiers à l'aide de l'objet XMLHttpRequest.

```
function transform(xml, xsl, id) {
if (window.XSLTProcessor) {
var fragment;
var xsltProcessor = new XSLTProcessor();
xsltProcessor.importStylesheet(xsl);
```

La fonction *transform()*, après s'être assuré que l'objet XSLTProcessor est disponible (`if (window.XSLTProcessor)`), crée une nouvelle instance de celui-ci (`xsltProcessor = new XSLTProcessor()`). Le fichier de style xsl est alors importé dans l'objet XSLTProcessor (`xsltProcessor.importStyle-sheet(xsl)`).

```
fragment = xsltProcessor.transformToFragment(xml, document);
```

Pour que la transformation soit effective, XSLTProcessor doit exécuter la méthode `transformToDocument()` qui retourne un document XML entier ou la méthode `transformToFragment()` qui retourne un fragment de document pouvant être facilement ajouté à un document XML existant.

Nous retenons la méthode `transformToFragment()`. Cette dernière qui prend les deux variables suivantes comme paramètres :

- le document XML à transformer ,
- l'objet document qui accueille le fragment généré.

La documentation de Firefox signale que si le fragment généré est inséré dans le document HTML courant, le passage du paramètre `document` est suffisant. Cette transformation permet d'extraire des données après le chargement de la page, sans avoir à la rafraîchir.

```
var target = document.getElementById(id);
target.innerHTML="";
target.appendChild(fragment);
document.appendChild(target);
}
}
```

Il est maintenant possible d'afficher le document XML transformé par la feuille de style XSL.

La variable target repère la balise `<div id="transform">`, dont le contenu de celui-ci est effacé (*target.innerHTML=""*). Le fichier XML transformé (variable `fragment`) est alors ajouté comme dernier enfant (`appendChild(fragment)`) à la variable `target`. Le tout est inclus dans le document courant (`document.appendChild(target)`.

```
//]]>
</script>
```

Fin de script.

Le fichier Xhtml final devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<style type="text/css">
<!--
#transform {background-color: #9cf;
            padding: 10px;
            text-align: center;}
-->
</style>
<script type="text/javascript">
//<![CDATA[
var xml = loadXML('xml.xml');
var xsl = loadXML('xsl.xsl');
function loadXML(url) {
var xhr;
if (document.implementation.createDocument) {
xhr = new XMLHttpRequest();
xhr.open('GET', url, false);
xhr.send(null);
if (xhr.readyState == 4){
xhr = xhr.responseXML;
}
}
return xhr;
}
function transform(xml, xsl, id) {
if (window.XSLTProcessor) {
var fragment;
var xsltProcessor = new XSLTProcessor();
xsltProcessor.importStylesheet(xsl);
fragment = xsltProcessor.transformToFragment(xml, document);
var target = document.getElementById(id);
target.innerHTML="";
target.appendChild(fragment);
document.appendChild(target);
}
}
//]]>
```

```
</script>
</head>
<body>
<h2>Les victoires de la musique</h2>
<div id="transform">
<form action="">
<input type="button" value="Afficher le palmarès"
onclick="transform(xml, xsl, 'transform')" />
</form>
</div>
</body>
</html>
```

4. Solution compatible

Il serait assez convivial d'avoir un script compatible, à la fois pour les navigateurs de la famille Internet Explorer et pour le navigateur Firefox.

Le script suivant reprend le code élaboré pour chacun d'eux aux points précédents.

Le fichier Xhtml est inchangé.

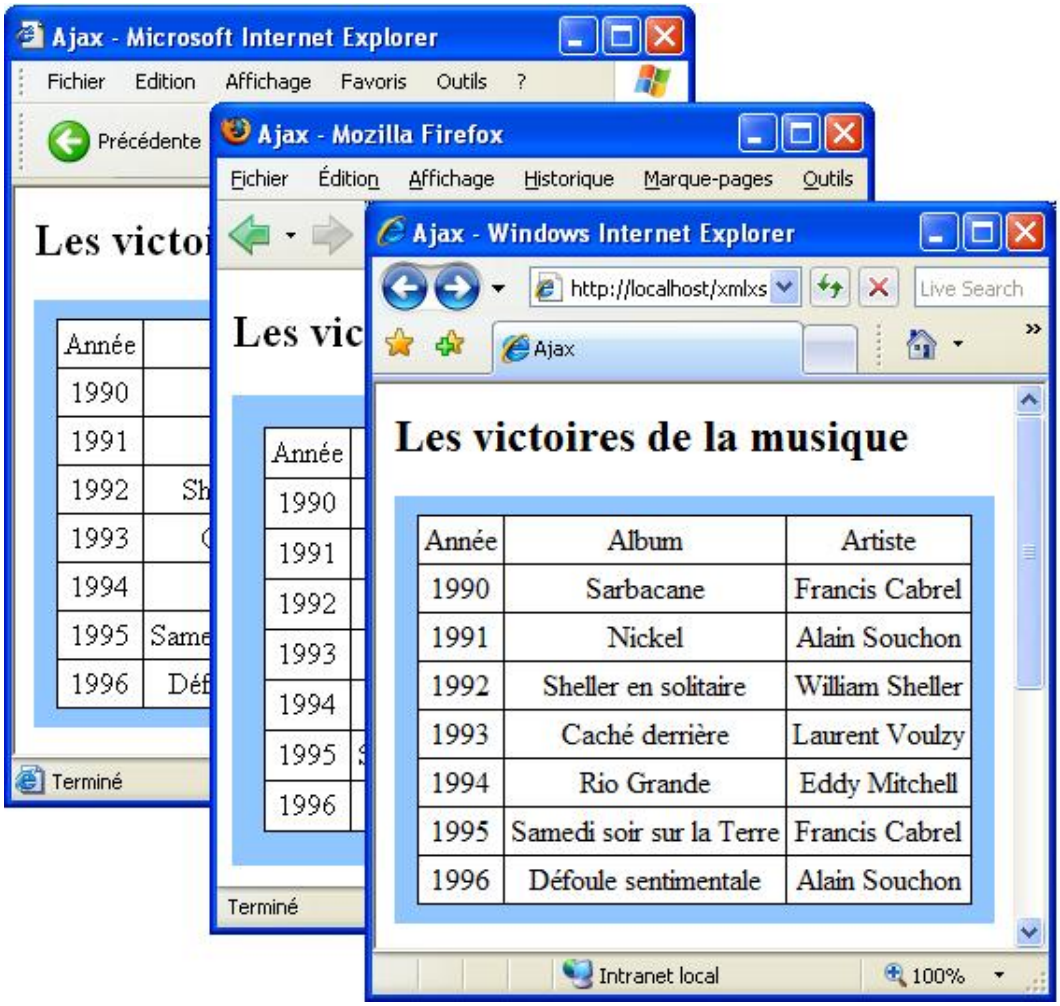
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<style type="text/css">
<!--
#transform {background-color: #9cf;
            padding: 10px;
            text-align: center;}
-->
```

```
</style>
</head>
<body>
<h2>Les victoires de la musique</h2>
<div id="transform">
<form action="">
<input type="button" value="Afficher le palmarès"
onclick="transform(xml, xsl, 'transform')" />
</form>
</div>
</body>
</html>
```

Le script compatible devient :

```
<script type="text/javascript">
//<![CDATA[
var xml = loadXML('xml.xml');
var xsl = loadXML('xsl.xsl');
function loadXML(url) {
var xhr;
// code pour Firefox
if (document.implementation.createDocument) {
xhr = new XMLHttpRequest();
xhr.open('GET', url, false);
xhr.send(null);
if (xhr.readyState == 4){
xhr = xhr.responseXML;
}
}
// code pour IE7
else if (window.XMLHttpRequest){
xhr = new XMLHttpRequest();
xhr.open('GET', url, false);
xhr.send(null);
if (xhr.readyState == 4){
xhr = xhr.responseXML;
}
}
//code pour IE6, IE5.5 et IE5
else if (window.ActiveXObject) {
var xhr = new ActiveXObject("Microsoft.XMLDOM");
xhr.async = false;
xhr.load(url);
}
return xhr;
}

function transform(xml, xsl, id){
if (window.XSLTProcessor) {
var fragment;
var xsltProcessor = new XSLTProcessor();
xsltProcessor.importStylesheet(xsl);
fragment = xsltProcessor.transformToFragment(xml, document);
var target = document.getElementById(id);
target.innerHTML="";
target.appendChild(fragment);
document.appendChild(target);
}
else if (window.ActiveXObject) {
var target = document.getElementById(id);
target.innerHTML = xml.transformNode(xsl);
}
}
//]]>
</script>
```



5. Une application XSL dynamique

Cette application permet de découvrir qu'il est possible de changer dynamiquement la feuille de style d'un document XML pour la remplacer par une autre.

Soit notre fichier XML (xml.xml), utilisé tout au long de ce chapitre. Le fichier de style habituel (xsl.xsl) lui est associé dans un premier temps. Après une action de l'utilisateur, le script transforme le fichier XML de départ avec une nouvelle feuille de style.

Cette feuille de style (xsl1.xsl), tirée du chapitre Introduction du XSL - Trier avec le langage XSL, permet de trier le dossier XML par ordre alphabétique.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>XSL</title>
</head>
<body>
<table border="1" style="border-collapse: collapse"
bordercolor="#000000" bgcolor="#ffffff" cellpadding="3">
<tr>
<td>Artiste</td>
<td>Titre</td>
<td>Année</td>
</tr>
<xsl:for-each select="compilation/mp3">
<xsl:sort select="artiste" order="ascending" />
<tr>
```

```
<td><xsl:value-of select="artiste"/></td>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="date"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Le fichier Xhtml de départ est notre fichier usuel qui a cependant été légèrement modifié.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<style type="text/css">
<!--
#transform {background-color: #9cf;
            padding: 10px;
            text-align: center;}
-->
</style>
</head>
<body onload="init('xml.xml', 'xsl.xsl', 'transform')">
<h2>Les victoires de la musique</h2>
<div id="transform">
</div>
<form action="" style="text-align:center">
<input type="button" value="Trier par nom"
onclick="init('xml.xml', 'xsl1.xsl', 'transform')" />
</form>
</body>
</html>
```

Ainsi, au chargement de la page (`<body onload="init()"`), la fonction *init()* est appelée avec, comme paramètres, le fichier xml.xml, la feuille de style xsl.xsl et la zone de la page, identifiée par la balise `<div id="transform">`.

Au clic sur le bouton du formulaire, la même fonction est déclenchée mais cette fois avec comme paramètre de la feuille de style, le fichier xsl1.xsl.

Le script est lui aussi modifié afin de pouvoir tenir compte des adresses de fichiers, fournies en paramètres de la fonction d'appel.

Nous avons simplement ajouté au script, la fonction *init()* qui charge les fichiers XML et XSL en fonction des adresses fournies en paramètres (`xml = loadXML(url1)` et `xsl = loadXML(url2)`). La fonction init() passe ensuite la main à la fonction *transform()*.

```
<script type="text/javascript">
//<![CDATA[
function init(url1, url2,id){
var xml = loadXML(url1);
var xsl = loadXML(url2);
transform(xml, xsl, id);
}
function loadXML(url) {
var xhr;
// code pour Firefox
if (document.implementation.createDocument) {
xhr = new XMLHttpRequest();
xhr.open('GET', url, false);
xhr.send(null);
if (xhr.readyState == 4){
xhr = xhr.responseXML;
}
}
// code pour IE7
else if (window.XMLHttpRequest){
xhr = new XMLHttpRequest();
xhr.open('GET', url, false);
xhr.send(null);
if (xhr.readyState == 4){
xhr = xhr.responseXML;
}
}
//code pour IE6, IE5.5 et IE5
else if (window.ActiveXObject) {
var xhr = new ActiveXObject("Microsoft.XMLDOM");
```

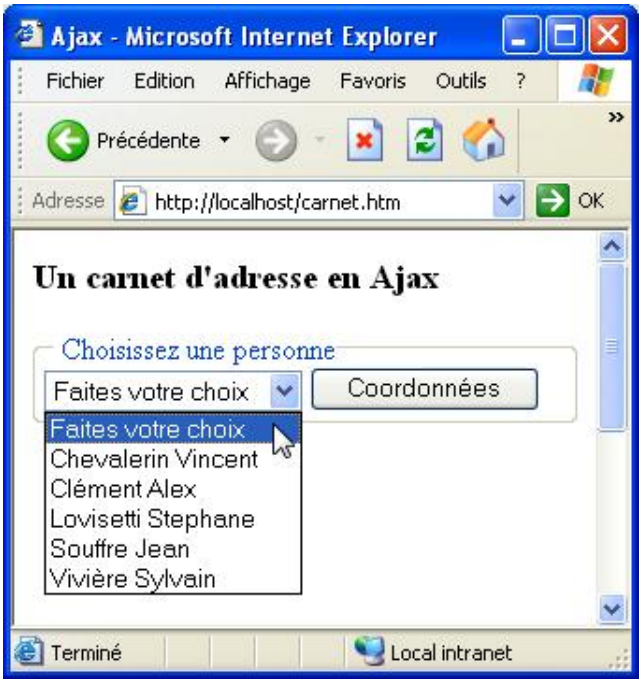
```
xhr.async = false;
xhr.load(url);
}
return xhr;
}
function transform(xml, xsl, id){
if (window.XSLTProcessor) {
var fragment;
var xsltProcessor = new XSLTProcessor();
xsltProcessor.importStylesheet(xsl);
fragment = xsltProcessor.transformToFragment(xml, document);
var target = document.getElementById(id);
target.innerHTML="";
target.appendChild(fragment);
document.appendChild(target);
}
else if (window.ActiveXObject) {
var target = document.getElementById(id);
target.innerHTML = xml.transformNode(xsl);
}
}
//]]>
</script>
```

Le résultat nous fournit le tableau trié par nom.

Un carnet d'adresses

Ce script propose une liste de sélection avec plusieurs noms. Au clic sur le bouton, les coordonnées de la personne sont affichées.

Les informations relatives aux personnes sont présentes sur le serveur sous forme d'un fichier XML.



Pour rappel, tous les fichiers de ces exemples doivent être situés sur le serveur local et sont accessibles dans le navigateur, par une adresse de type localhost/...

Élaborons d'abord le fichier XML. Chaque contact du répertoire comporte les coordonnées suivantes :

- Nom

- Prénom
- Adresse
- Code postal
- Ville
- Téléphone

La structure du fichier XML prend la forme suivante :

```
<carnet>
  <contact>
    <nom></nom>
    <prenom></prenom>
    <adresse></adresse>
    <cp></cp>
    <ville></ville>
    <tel></tel>
  </contact>
</carnet>
```

Le fichier XML (carnet.xml) proposé est :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<carnet>
<contact>
<nom>Chevalerin</nom>
<prenom>Vincent</prenom>
<adresse>13, rue du Maréchal Joffre</adresse>
<cp>59000</cp>
<ville>Lille</ville>
<tel>+33-(0)-320219729</tel>
</contact>
<contact>
<nom>Clément</nom>
<prenom>Alex</prenom>
<adresse>17, boulevard Jean Jaurès</adresse>
<cp>31000</cp>
<ville>Toulouse</ville>
<tel>+33-(0)-565906248</tel>
</contact>
<contact>
<nom>Lovisetti</nom>
<prenom>Stephane</prenom>
<adresse>9, avenue Poissonnière</adresse>
<cp>75000</cp>
<ville>Paris</ville>
<tel>+33-(0)-173409540</tel>
</contact>
<contact>
<nom>Souffre</nom>
<prenom>Jean</prenom>
<adresse>121, rue Robert Caumont</adresse>
<cp>33000</cp>
<ville>Bordeaux</ville>
<tel>+33-(0)-556002266</tel>
</contact>
<contact>
<nom>Vivière</nom>
<prenom>Sylvain</prenom>
<adresse>64, rue des Sources</adresse>
<cp>69000</cp>
<ville>Lyon</ville>
<tel>+33-(0)-478383899</tel>
```

```
</contact>
</carnet>
```

Passons maintenant au fichier Xhtml de départ :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
</head>
<body>
<h3>Un carnet d'adresse en Ajax</h3>
<form action="">
<fieldset>
<legend>Choisissez une personne</legend>
<select name="personnes">
<option>Faites votre choix</option>
<option>Chevalerin Vincent</option>
<option>Clément Alex</option>
<option>Lovisetti Stephane</option>
<option>Souffre Jean</option>
<option>Vivière Sylvain</option>
</select>
<input type="button" onclick="extraire(personnes)"
value="Coordonnées" />
</fieldset>
</form>
<div id="cadre" style="display:none;">
Nom Prénom : <span id="idnom"></span> 
<span id="idprenom"></span><br />
Adresse : <span id="idadresse"></span><br />
Ville : <span id="idcp"></span> <span id="idville"></span>
<br />
Tel : <span id="idtelephone"></span>
</div>
</body>
</html>
```

Commentaires :

- La liste de sélection est introduite par la balise `<select> ... </select>`.
- Les balises `<fieldset> ... </fieldset>` et `<legend> ... </legend>` sont des balises classiques en Html 4.1 et Xhtml 1.0 mais peu connues. Elles permettent d'organiser la présentation des formulaires.
- Le résultat s'affiche dans la zone `<div id="cadre"> ... </div>` qui, à l'ouverture de la page, est cachée (`style="display:none;"`).
- Au clic du bouton, la fonction `extraire(personnes)` permet de lancer le script.

Abordons la partie JavaScript.

```
<script type='text/JavaScript'>
//<![CDATA[
var xhr = null;
function extraire(personnes){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
```

```
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
```

Après avoir initialisé la variable xhr, élaborons le code de la fonction `extraire()`. Celle-ci est, pour rappel, appelée au clic du bouton **Coordonnées** dans le fichier Xhtml.

Les premières lignes de script tiennent compte des différences de traitement d'une requête HTTP d'un navigateur à l'autre. Firefox et Internet Explorer 7 passent par l'objet JavaScript natif XMLHttpRequest, tandis qu'Internet Explorer 6 et versions précédentes utilisent un contrôle ActiveX (voir chapitre L'objet XMLHttpRequest - Créer un objet XMLHttpRequest).

```
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
afficher(xmldocument,personnes);
}
}
xhr.open("GET", "carnet.xml", true);
xhr.send(null);
}
}
```

Après s'être assuré que l'objet `xhr` existe bien (`if(xhr)`), une requête est mise en place selon la méthode GET, vers le fichier carnet.xml et selon un mode asynchrone. La requête est alors effectivement exécutée lors de l'instruction `xhr.send(null)`.

La bonne exécution de cette requête HTTP est surveillée en fonction des informations retournées sur ses propres changements d'état, soit par (`onreadystatechange`). L'assurance que la requête a bien abouti est obtenu à deux conditions : d'une part, les données sont accessibles (`readyState == 4`) d'autre part le code numérique de la réponse du serveur est satisfaisant (`status == 200`). Le fichier est alors retourné sous la forme d'un document XML (`xhr.responseXML`). Ce dernier est stocké dans la variable `xmldocument`. Le fichier XML peut maintenant être traité par le JavaScript du navigateur par la fonction *afficher()*. Le fichier XML (`xmldocument`) et les données du menu déroulant (`personnes`) sont fournis comme arguments de la fonction.

```
function afficher(xmldocument,personnes) {
```

La fonction `afficher()` se charge de récupérer les différentes coordonnées (nom, prénom, adresse, etc) de la personne sélectionnée et d'afficher le résultat dans la page Xhtml.

```
var ordre = (personnes.selectedIndex - 1);
if(ordre!= -1) {
var affichage= document.getElementById("cadre");
affichage.style.display = 'block'
```

Il importe de connaître la personne retenue par l'utilisateur dans le menu déroulant. C'est le rôle de `selectedIndex`. Sa valeur est stockée dans la variable `ordre`. La première option du menu étant le texte "Faites votre choix", il faut prendre la précaution de diminuer cet index d'une unité (`selectedIndex - 1`). Tout affichage est exclu si cette valeur est négative (`if(ordre!= -1)`). Le contenu de la balise `<div id="cadre"> ... </div>` devient apparent à ce stade du processus (`style.display = 'block'`) sachant qu'il est caché dans le fichier Xhtml initial.

```
nouedsnom = xmldocument.getElementsByTagName("nom");
noeudsprenom = xmldocument.getElementsByTagName("prenom");
noeudsadresse = xmldocument.getElementsByTagName("adresse");
nouedscp = xmldocument.getElementsByTagName("cp");
noeudsville = xmldocument.getElementsByTagName("ville");
noeudstel = xmldocument.getElementsByTagName("tel");
```

Les différentes balises (ex : `<nom>`) du fichier XML sont récupérées dans une liste (ex : `noeuds.nom`) grâce à la méthode `getElementsByTagName`. Les balises suivantes sont traitées de la même façon : `<prenom>`, `<adresse>`, `<cp>`, `<ville>` et `<telephone>`.

```
var textenom = nouedsnom[ordre].firstChild.nodeValue;
var texteprenom = noeudsprenom[ordre].firstChild.nodeValue;
var texteadresse = noeudsadresse[ordre].firstChild.nodeValue;
var textecp = nouedscp[ordre].firstChild.nodeValue;
var texteville = noeudsville[ordre].firstChild.nodeValue;
var textetel = noeudstel[ordre].firstChild.nodeValue;
```

Il faut maintenant atteindre le texte associé à la balise <nom> pour le contact sélectionné. Cette balise est déterminée par son numéro d'ordre du contact dans le menu déroulant (la variable ordre), associé à la liste noeudsnom. Dans l'arborescence du fichier XML, le nom du contact est le premier enfant de la balise <nom>(firstChild) dont la valeur de ce nœud texte est obtenue par la propriété nodeValue, et ainsi de suite pour les autres valeurs.

```
var targetnom = document.getElementById("idnom");
var targetprenom = document.getElementById("idprenom");
var targetrue = document.getElementById("idadresse");
var targetcp = document.getElementById("idcp");
var targetville = document.getElementById("idville");
var targettelephone = document.getElementById("idtelephone");
targetnom.innerHTML = textenom;
targetprenom.innerHTML = texteprenom;
targetrue.innerHTML = texteadresse;
targetcp.innerHTML = textecp;
targetville.innerHTML = texteville;
targettelephone.innerHTML = textetel;
}
}
```

La variable targetnom repère la balise ... du document Xhtml, par son identifiant id (getElementById("idnom");). Le contenu de la balise est modifié par la propriété innerHTML (targetnom.innerHTML) en fonction des informations recueillies par la variable textenom.

```
//]]>
</script>
```

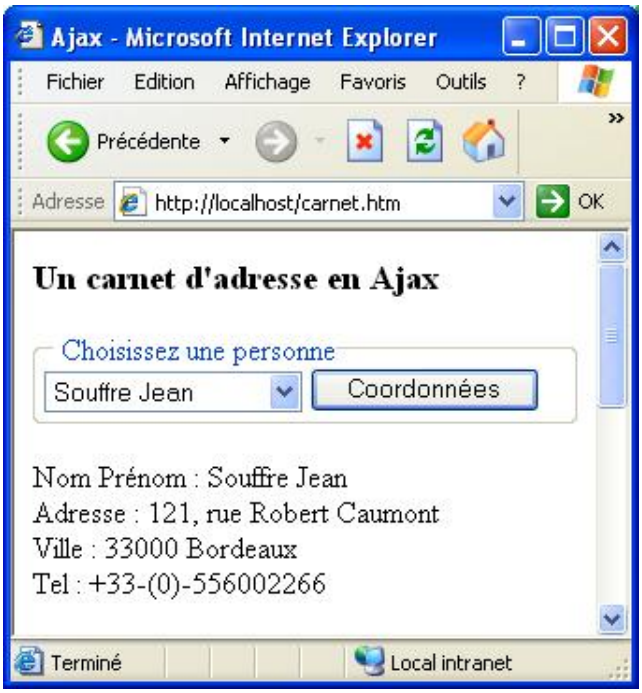
Le script est terminé.

Sachant que la fonction *afficher()* doit être disponible avant la fonction *extraire()*, le script complet devient :

```
<script type='text/JavaScript'>
//<![CDATA[
function afficher(xmldocument,personnes) {
var ordre = (personnes.selectedIndex - 1);
if(ordre!= -1) {
var affichage= document.getElementById("cadre");
affichage.style.display = 'block'
nouedsnom = xmldocument.getElementsByTagName("nom");
noeudsprenom = xmldocument.getElementsByTagName("prenom");
noeudsadresse = xmldocument.getElementsByTagName("adresse");
nouedscp = xmldocument.getElementsByTagName("cp");
noeudsville = xmldocument.getElementsByTagName("ville");
noeudstel = xmldocument.getElementsByTagName("tel");
var textenom = nouedsnom[ordre].firstChild.nodeValue;
var texteprenom = noeudsprenom[ordre].firstChild.nodeValue;
var texteadresse = noeudsadresse[ordre].firstChild.nodeValue;
var textecp = nouedscp[ordre].firstChild.nodeValue;
var texteville = noeudsville[ordre].firstChild.nodeValue;
var textetel = noeudstel[ordre].firstChild.nodeValue;
var targetnom = document.getElementById("idnom");
var targetprenom = document.getElementById("idprenom");
var targetrue = document.getElementById("idadresse");
var targetcp = document.getElementById("idcp");
var targetville = document.getElementById("idville");
var targettelephone = document.getElementById("idtelephone");
targetnom.innerHTML = textenom;
targetprenom.innerHTML = texteprenom;
targetrue.innerHTML = texteadresse;
targetcp.innerHTML = textecp;
targetville.innerHTML = texteville;
targettelephone.innerHTML = textetel;
}
}
var xhr = null;
function extraire(personnes){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
```

```
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
afficher(xmldocument,personnes);
}
}
xhr.open("GET", "carnet.xml", true);
xhr.send(null);
}
}
//]]>
</script>
```

Voici le résultat dans un navigateur :



Agrémentons le fichier Xhtml de quelques déclarations de style pour en améliorer la présentation.

```
<style type="text/css">
#cadre {
width: 250px;
height: 80px;
margin-left: 10px;
padding: 5px;
border: solid black 1px;
background-color: #9cf;
}
#idnom, #idprenom {
font-family: sans-serif;
font-weight: bold;
font-size: 11pt;
}
#idadresse {
font-style: italic;
}
#idtelephone {
font-weight: bold;
font-family: monospace;
```

```
}
</style>
```

Commentaires :

- Les différentes coordonnées (`<div id="cadre"`) sont affichées dans une boîte dotée d'une fine bordure et d'un arrière-plan de couleur. Ce dernier met en avant le résultat du script.
- Le nom et le prénom (`<span id="idnom">` et `<span id="idprenom">`) sont affichés en gras.
- L'adresse (`<span id="idadresse">`) est mise en lettres italiques.
- Le téléphone (`<span id="idtelephone">`) est présenté selon une police à chasse fixe.

Le fichier Xhtml complet, avec le script et les feuilles de style devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Script-Type" content="text/css" />
<script type='text/JavaScript'>
//<![CDATA[
function afficher(xmldocument,personnes) {
var ordre = (personnes.selectedIndex - 1);
if(ordre!= -1) {
var affichage= document.getElementById("cadre");
affichage.style.display = 'block'
nouedsnom = xmldocument.getElementsByTagName("nom");
noeudsprenom = xmldocument.getElementsByTagName("prenom");
noeudsadresse = xmldocument.getElementsByTagName("adresse");
nouedscp = xmldocument.getElementsByTagName("cp");
noeudsville = xmldocument.getElementsByTagName("ville");
noeudstel = xmldocument.getElementsByTagName("tel");
var textenom = nouedsnom[ordre].firstChild.nodeValue;
var texteprenom = noeudsprenom[ordre].firstChild.nodeValue;
var texteadresse = noeudsadresse[ordre].firstChild.nodeValue;
var textecp = nouedscp[ordre].firstChild.nodeValue;
var texteville = noeudsville[ordre].firstChild.nodeValue;
var textetel = noeudstel[ordre].firstChild.nodeValue;
var targetnom = document.getElementById("idnom");
var targetprenom = document.getElementById("idprenom");
var targetrue = document.getElementById("idadresse");
var targetcp = document.getElementById("idcp");
var targetville = document.getElementById("idville");
var targettelephone = document.getElementById("idtelephone");
targetnom.innerHTML = textenom;
targetprenom.innerHTML = texteprenom;
targetrue.innerHTML = texteadresse;
targetcp.innerHTML = textecp;
targetville.innerHTML = texteville;
targettelephone.innerHTML = textetel;
}
}
var xhr = null;
function extraire(personnes){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
```

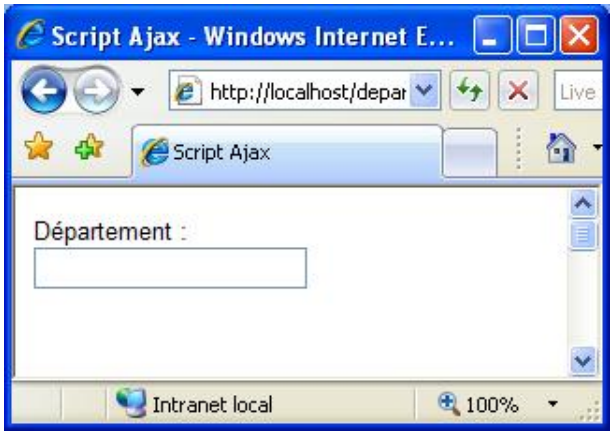
```
alert("Votre navigateur n’est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
afficher(xmldocument,personnes);
}
}
xhr.open("GET", "carnet.xml", true);
xhr.send(null);
}
}
//]]>
</script>
<style>
#cadre {
width: 250px;
height: 80px;
margin-left: 10px;
padding: 5px;
border: solid black 1px;
background-color: #9cf;
}
#idnom, #idprenom {
font-family: sans-serif;
font-weight: bold;
font-size: 11pt;
}
#idadresse {
font-style: italic;
}
#idtelephone {
font-weight: bold;
font-family: monospace;
}
</style>
</head>
<body>
<h3>Un carnet d’adresse en Ajax</h3>
<form action="">
<fieldset>
<legend>Choisissez une personne</legend>
<select name="personnes">
<option>Faites votre choix</option>
<option>Chevalerin Vincent</option>
<option>Clément Alex</option>
<option>Lovisetti Stephane</option>
<option>Souffre Jean</option>
<option>Vivière Sylvain</option>
</select>
<input type="button" onclick="extraire(personnes)"
value="Coordonnées" />
</fieldset>
</form>
<div id="cadre" style="display:none;">
Nom Prénom : <span id="idnom"></span> <span id="idprenom"></span>
<br />
Adresse : <span id="idadresse"></span><br />
Ville : <span id="idcp"></span> <span id="idville"></span><br />
Tel : <span id="idtelephone"></span>
</div>
</body>
</html>
```

Le script de cet exemple est compatible entre Internet Explorer et Firefox.

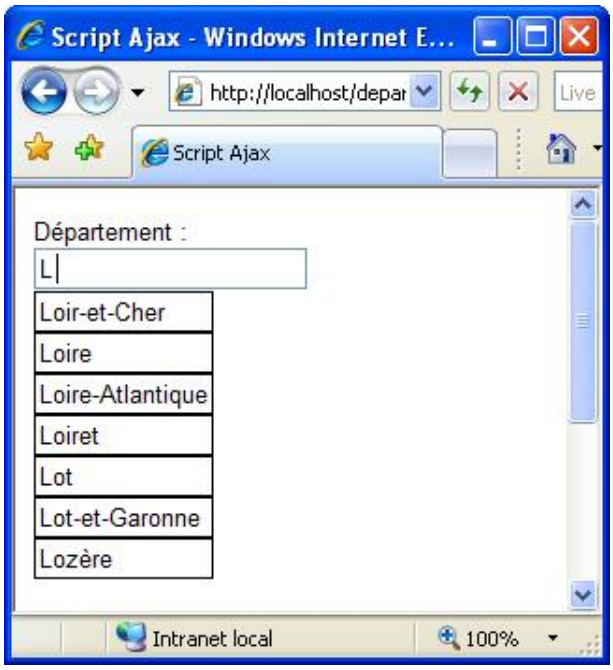
Suggestion d'encodage

La philosophie de cet exemple est de présenter, en cours de saisie de données, au clavier par l'utilisateur des suggestions de valeurs.

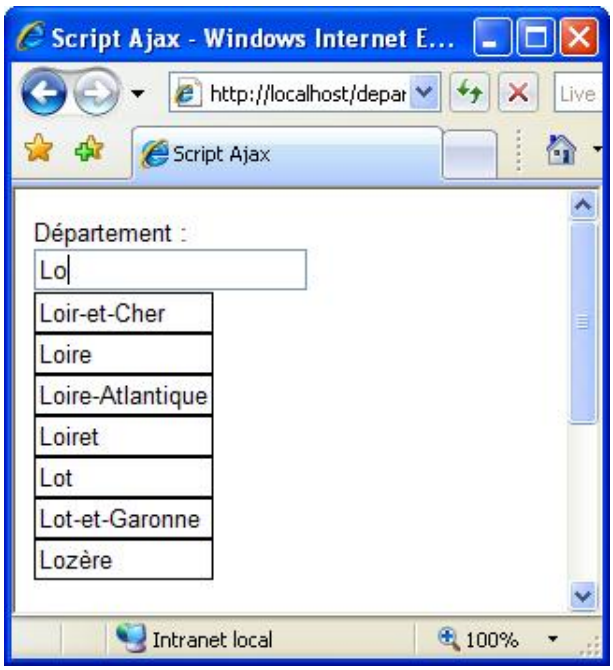
Soit une page Web dans laquelle le visiteur peut saisir le nom de son département.



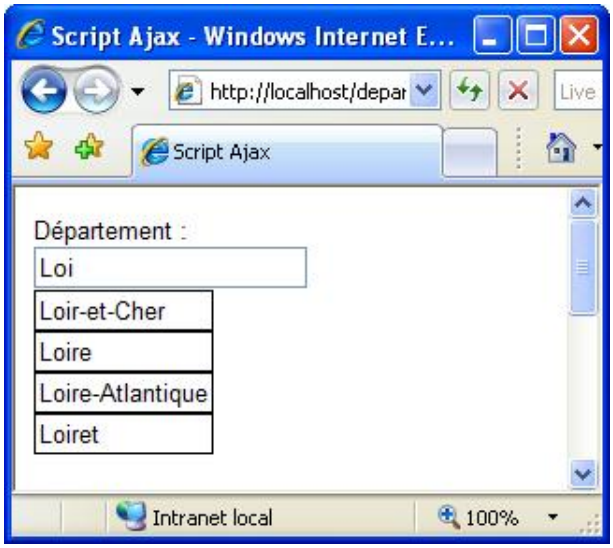
À la saisie clavier de la lettre "L", diverses propositions sont affichées dans un menu déroulant, soit tous les départements commençant par la lettre L : Landes, Loir-et-Cher, Loire, Loire-Atlantique, Loiret, Lot, Lot-et-Garonne, Lozère.



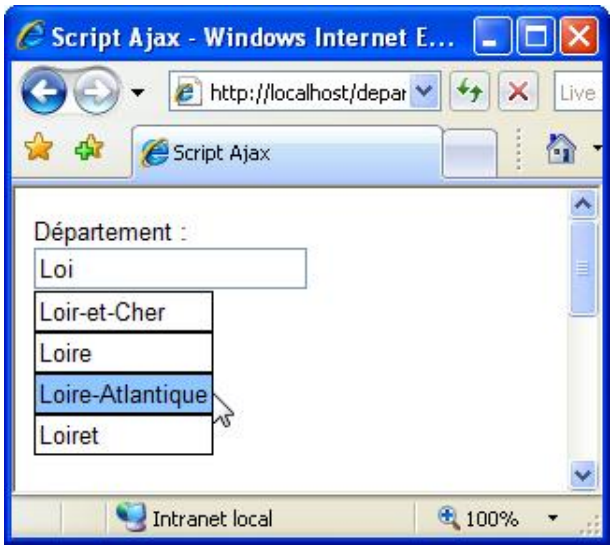
L'utilisateur poursuit la saisie et tape la lettre "o". Le menu déroulant ne présente alors que les départements commençant par "Lo" et ce instantanément, sans que la page soit rechargée.



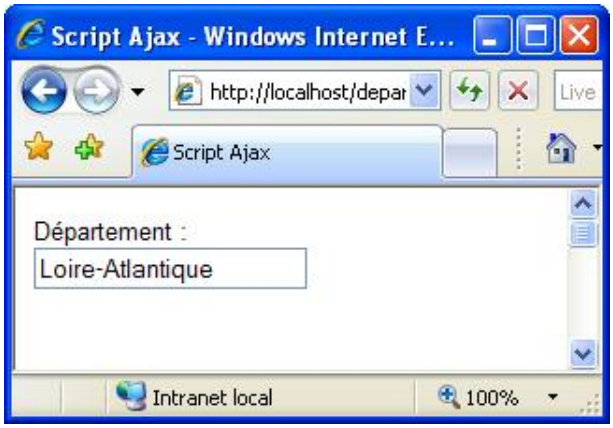
Continuons avec la saisie de la lettre "i". Une nouvelle modification du menu déroulant fait apparaître les départements commençant par les lettres "Loi" (Loir-et-Cher, Loire, Loire-Atlantique et Loiret).



Supposons qu'à ce stade, le département de la Loire-Atlantique est retenu.



Il suffit alors de cliquer sur le département de la Loire-Atlantique pour que celui-ci s'affiche automatiquement dans la zone de texte, en faisant disparaître dans le même temps, la liste déroulante.



Cet exemple s'inspire directement de Google Suggest (labs.google.com/suggest/) qui suggère les 10 mots-clés les plus pertinents pour une recherche avec le nombre de pages se référant à des mots-clés (voir chapitre Présentation générale d'AJAX, Exemples sur le web).

Cet exemple illustre parfaitement l'apport du Web 2.0 et du concept AJAX, dans le but de faciliter l'utilisation quotidienne du Web ainsi que le confort d'utilisation.

Au niveau du code, ce script comporte dans son ensemble :

- Un fichier XML, situé sur le serveur, qui reprend la liste des départements.
- Un fichier Xhtml avec, dans le cas présent, une simple ligne de texte.
- Un fichier JavaScript externe qui, après avoir récupéré la ou les lettre(s) saisies dans la zone de texte, exécute une requête HTTP pour interroger le fichier XML, récupère les départements correspondants et les affiche sous la forme d'un menu popup déroulant.
- Un fichier de feuilles de style CSS externe prenant en charge la présentation du menu déroulant.

Commençons par le fichier XML (departements.xml).

L'arborescence de ce fichier se présente comme suit :

- L'élément racine du document est la balise `<choix> ... </choix>`.
- Pour chaque département, une balise `<item>` est prévue.
- Chaque item contient la balise `<dep> ... </dep>` pour le nom du département et la balise `<valeur> ... </valeur>` pour le numéro qui lui est associé.

```
<choix>
<item>
<dep> ... </dep>
<valeur> ... </valeur>
</item>
</choix>
```

Ce fichier, disponible dans l'espace de téléchargement, se présente comme suit :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<choix xml:lang="FR">
<item><dep>Ain</dep><valeur>01</valeur></item>
<item><dep>Aisne</dep><valeur>02</valeur></item>
<item><dep>Allier</dep><valeur>03</valeur></item>
<item><dep>Alpes-de-Haute-Provence</dep><valeur>04</valeur></item>
<item><dep>Hautes-Alpes</dep><valeur>05</valeur></item>
<item><dep>Alpes-Maritimes</dep><valeur>06</valeur></item>
<item><dep>Ardèche</dep><valeur>07</valeur></item>
<item><dep>Ardennes</dep><valeur>08</valeur></item>
<item><dep>Ariège</dep><valeur>09</valeur></item>
<item><dep>Aube</dep><valeur>10</valeur></item>
<item><dep>Aude</dep><valeur>11</valeur></item>
<item><dep>Aveyron</dep><valeur>12</valeur></item>
<item><dep>Bouches-du-Rhône</dep><valeur>13</valeur></item>
<item><dep>Calvados</dep><valeur>14</valeur></item>
<item><dep>Cantal</dep><valeur>15</valeur></item>
<item><dep>Charente</dep><valeur>16</valeur></item>
<item><dep>Charente-Maritime</dep><valeur>17</valeur></item>
<item><dep>Cher</dep><valeur>18</valeur></item>
<item><dep>Corrèze</dep><valeur>19</valeur></item>
<item><dep>Côte-d'Or</dep><valeur>21</valeur></item>
<item><dep>Côtes-d'Armor</dep><valeur>22</valeur></item>
<item><dep>Creuse</dep><valeur>23</valeur></item>
<item><dep>Dordogne</dep><valeur>24</valeur></item>
<item><dep>Doubs</dep><valeur>25</valeur></item>
<item><dep>Drôme</dep><valeur>26</valeur></item>
<item><dep>Eure</dep><valeur>27</valeur></item>
<item><dep>Eure-et-Loir</dep><valeur>28</valeur></item>
<item><dep>Finistère</dep><valeur>29</valeur></item>
<item><dep>Corse-du-Sud</dep><valeur>2A</valeur></item>
<item><dep>Haute-Corse</dep><valeur>2B</valeur></item>
<item><dep>Gard</dep><valeur>30</valeur></item>
<item><dep>Gers</dep><valeur>31</valeur></item>
<item><dep>Gironde</dep><valeur>33</valeur></item>
<item><dep>Hérault</dep><valeur>34</valeur></item>
<item><dep>Ille-et-Vilaine</dep><valeur>35</valeur></item>
<item><dep>Indre</dep><valeur>36</valeur></item>
<item><dep>Indre-et-Loire</dep><valeur>37</valeur></item>
<item><dep>Isère</dep><valeur>38</valeur></item>
<item><dep>Jura</dep><valeur>39</valeur></item>
<item><dep>Landes</dep><valeur>40</valeur></item>
<item><dep>Loir-et-Cher</dep><valeur>41</valeur></item>
<item><dep>Loire</dep><valeur>42</valeur></item>
<item><dep>Haute-Loire</dep><valeur>43</valeur></item>
<item><dep>Loire-Atlantique</dep><valeur>44</valeur></item>
<item><dep>Loiret</dep><valeur>45</valeur></item>
<item><dep>Lot</dep><valeur>46</valeur></item>
<item><dep>Lot-et-Garonne</dep><valeur>47</valeur></item>
<item><dep>Lozère</dep><valeur>48</valeur></item>
<item><dep>Maine-et-Loire</dep><valeur>49</valeur></item>
<item><dep>Manche</dep><valeur>50</valeur></item>
```

```
<item><dep>Marne</dep><valeur>51</valeur></item>
<item><dep>Haute-Marne</dep><valeur>52</valeur></item>
<item><dep>Mayenne</dep><valeur>53</valeur></item>
<item><dep>Meurthe-et-Moselle</dep><valeur>54</valeur></item>
<item><dep>Meuse</dep><valeur>55</valeur></item>
<item><dep>Morbihan</dep><valeur>56</valeur></item>
<item><dep>Moselle</dep><valeur>57</valeur></item>
<item><dep>Nièvre</dep><valeur>58</valeur></item>
<item><dep>Nord</dep><valeur>59</valeur></item>
<item><dep>Oise</dep><valeur>60</valeur></item>
<item><dep>Orne</dep><valeur>61</valeur></item>
<item><dep>Pas-de-Calais</dep><valeur>62</valeur></item>
<item><dep>Puy-de-Dôme</dep><valeur>63</valeur></item>
<item><dep>Pyrénées-Atlantiques</dep><valeur>64</valeur></item>
<item><dep>Hautes-Pyrénées</dep><valeur>65</valeur></item>
<item><dep>Pyrénées-Orientales</dep><valeur>66</valeur></item>
<item><dep>Bas-Rhin</dep><valeur>67</valeur></item>
<item><dep>Haut-Rhin</dep><valeur>68</valeur></item>
<item><dep>Rhône</dep><valeur>69</valeur></item>
<item><dep>Haute-Saône</dep><valeur>70</valeur></item>
<item><dep>Saône-et-Loire</dep><valeur>71</valeur></item>
<item><dep>Sarthe</dep><valeur>72</valeur></item>
<item><dep>Savoie</dep><valeur>73</valeur></item>
<item><dep>Haute-Savoie</dep><valeur>74</valeur></item>
<item><dep>Paris</dep><valeur>75</valeur></item>
<item><dep>Seine-Maritime</dep><valeur>76</valeur></item>
<item><dep>Seine-et-Marne</dep><valeur>77</valeur></item>
<item><dep>Yvelines</dep><valeur>78</valeur></item>
<item><dep>Deux-Sèvres</dep><valeur>79</valeur></item>
<item><dep>Somme</dep><valeur>80</valeur></item>
<item><dep>Tarn</dep><valeur>81</valeur></item>
<item><dep>Tarn-et-Garonne</dep><valeur>82</valeur></item>
<item><dep>Var</dep><valeur>83</valeur></item>
<item><dep>Vaucluse</dep><valeur>84</valeur></item>
<item><dep>Vendée</dep><valeur>85</valeur></item>
<item><dep>Vienne</dep><valeur>86</valeur></item>
<item><dep>Haute-Vienne</dep><valeur>87</valeur></item>
<item><dep>Vosges</dep><valeur>88</valeur></item>
<item><dep>Yonne</dep><valeur>89</valeur></item>
<item><dep>Territoire-de-Belfort</dep><valeur>90</valeur></item>
<item><dep>Essonne</dep><valeur>91</valeur></item>
<item><dep>Hauts-de-Seine</dep><valeur>92</valeur></item>
<item><dep>Seine-Saint-Denis</dep><valeur>93</valeur></item>
<item><dep>Val-de-Marne</dep><valeur>94</valeur></item>
<item><dep>Val-d’Oise</dep><valeur>95</valeur></item>
</choix>
```

Le fichier Xhtml (departement.htm) est assez sommaire. Il comporte un formulaire avec comme unique élément une ligne de texte et un élément `<div> ... </div>` identifiée par `id="popups"`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Script Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<link rel="stylesheet" rev="stylesheet" href="departements.css" />
<script src="departements.js" type="text/javascript">
</script>
</head>
<body>
<form action="">
Département :<br />
<input type="text" id="formulaire" /><br />
<div id="popups"> </div>
</form>
```

```
</body>
</html>
```

Le fichier JavaScript est plus complexe à cause de la multiplicité des tâches qu'il doit gérer. Voici le code pas à pas de ce fichier :

```
window.onload = initAll;
var xhr = null;
var departements = new Array();
```

Comme il s'agit d'un fichier JavaScript externe, on initialise la fonction `initAll()` au chargement de la page (`onload`).

Après la déclaration de la variable `xhr`, un tableau est créé (`newArray()`) afin d'y insérer les différents départements.

```
function initAll() {
document.getElementById("formulaire").onkeyup = searchSuggest;
```

La fonction `initAll()` récupère d'abord l'action de l'utilisateur dans la ligne de texte du formulaire (`getElementById ("formulaire")`) .L'evénement associé à celle-ci est de type *onkeyup*. Ainsi à chaque lettre saisie, la fonction `searchSuggest()` sera appelée (voir plus loin dans le script).

Une requête HTTP peut alors être lancée sur le serveur.

```
if (window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject) {
try {
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e) { }
}
}
```

Comme nous l'avons vu au chapitre L'objet XMLHttpRequest - Créer un objet XMLHttpRequest , ce bout de script est destiné à distinguer les procédures différentes mises en œuvre pour Internet Explorer 7 et Firefox (`xhr = new XMLHttpRequest();`) et Internet Explorer 6 ou versions antérieures pour lesquelles un contrôle ActiveX est encore utilisé (`xhr = new ActiveX-Object("Microsoft.XMLHTTP");`).

```
if (xhr) {
xhr.onreadystatechange = setdepartements;
xhr.open("GET", "departements.xml", true);
xhr.send(null);
}
else {
alert("Désolé, votre navigateur n'est pas compatible avec AJAX");
}
}
```

Si l'objet *xhr* a bien été créé (`if (xhr)`), la requête est initialisée selon la méthode GET, pour lire le fichier *departements.xml* en mode asynchrone. La requête est ensuite effectuée par l'instruction `send(null)`. Dès qu'il y a des changements d'état de la requête (`onreadystatechange`), la fonction `setdepartements()` est appelée.

```
function setdepartements() {
if (xhr.readyState == 4) {
if (xhr.status == 200) {
```

Cette fonction `setdepartements()` teste d'abord si les données retournées sont complètement accessibles (`xhr.readyState == 4`) puis si la requête a été exécutée avec succès (`xhr.status == 200`).

```
if (xhr.responseXML) {
var tousdepartements = xhr.responseXML.getElementsByTagName("item");
```

Si l'objet `xhr.responseXML` existe, la variable tousdepartements contient, sous forme d'un document XML, toutes les balises `<item>` du fichier, répertoriées grâce au nom de la balise (`getElementsByTagName("item")`).

```
for (var i=0; i<tousdepartements.length; i++) {
```

```
departements[i] = tousdepartements[i].getElementsByTagName("dep")[0].firstChild;
}
}
}
else {
alert("Il y a un problème avec la requête " + xhr.status);
}
}
}
```

Par une boucle *for*, chaque item va être parcouru un à un (`tousdepartements[i]`). La balise `<dep>` (`getElementsByTagName("dep")[0]`) est identifée et son premier enfant est mémorisé, soit le nœud texte contenant le nom du département. Ce nom est stocké dans le tableau *departements*.

Les données du document XML ayant ainsi été récupérées, la fonction `searchsuggest()` va à présent les traiter.

```
function searchSuggest() {
var str = document.getElementById("formulaire").value;
document.getElementById("formulaire").className = "";
```

La variable `str` contiendra la ou les lettres encodée(s) (`value`) par l'utilisateur dans la ligne de texte (`getElementById("formulaire")`).

À présent, considérons la propriété de style associée à la ligne de texte. À ce stade, rien n'est spécifié (`getElementById("formulaire").className = ""`).

```
if (str != "") {
document.getElementById("popups").innerHTML = "";
```

Si le contenu de la zone de texte n'est pas vide (`str!= ""`), le contenu de la balise `<div id="popups">` du document Xhtml est initialisé. À cet instant, ce contenu est vide (`innerHTML = ""`).

```
for (var i=0; i<departements.length; i++) {
var ce_departement = departements[i].nodeValue;
if (ce_departement.toLowerCase().indexOf(str.toLowerCase()) == 0) {
var tempDiv = document.createElement("div");
tempDiv.innerHTML = ce_departement;
tempDiv.onclick = choix;
tempDiv.className = "suggestions";
document.getElementById("popups").appendChild(tempDiv);
}
}
```

Grâce à une boucle *for*, tous les éléments du tableau `departements` sont traités : il s'agit des données récupérées dans la fonction `setdepartements()` étudiée ci-avant (`departements[i]`). Pour chaque élément, la valeur du nœud (`nodeValue`) est retenue, soit le nom du département. Ce nom est stocké dans la variable `ce_departement`.

Puis un test conditionnel vérifie si le contenu de la ligne de texte se trouve dans la ou les première(s) lettres du département en utilisant la méthode `indexOf` étudiée au chapitre Le JavaScript - Manipulation des chaînes de caractères (`ce_departement.indexOf(str) == 0`)). Par précaution le nom du département et la variable str (`toLowerCase()`) sont transformés en lettres minuscules.

Si c'est le cas, un nouvel élément `<div>` (`createElement("div")`) est créé ayant pour contenu le nom du département (`innerHTML = ce_departement`). Le cas pour lequel l'utilisateur chppisit cet élément par un clic de la souris est prévu. Ce clic renvoie à la fonction choix() qui sera détaillée ultérieurement (`onclick = choix`). L'affichage de cet élément `<div>` est pris en charge par la propriété de style qui s'applique à la classe suggestions du fichier departements.css (`className = "suggestions"`).

Et enfin, ce nouvel élément est inclus dans la balise `<div id="popups">` en dernière position grâce à la méthode `appendChild()`.

```
var liste = document.getElementById("popups").childNodes.length;
if (liste == 0) {
document.getElementById("formulaire").className = "error";
}
```

Le script prévoit, en outre, l'éventualité d'une saisie qui n'a rien à voir avec le nom des départements, par exemple le caractère @. La variable `liste` contient le nombre des nœuds enfant créés dans la balise `<div id="popups">` par la boucle for précédente (`getElementById("popups").childNodes.length`). Dans le cas d'une saisie inadéquat (`liste == 0`), la feuille de style `error` est appliquée au formulaire.

```
if (liste == 1) {
document.getElementById("formulaire").value = document.getElementById("
popups").firstChild.innerHTML;
document.getElementById("popups").innerHTML = "";
}
}
}
```

Le script prévoit également le cas où il n'y aura qu'un seul département possible à afficher à partir de la première lettre saisie dans la zone de texte. C'est le cas de la lettre J, le seul département ayant pour initiale J est le Jura, de même pour la lettre F le seul choix possible est le Finistère. Dans cette éventualité, le script peut afficher directement le département.

Voici l'explication du code : i cette variable `liste` ne contient qu'un seul élément (`liste == 1`), JavaScript peut directement encoder dans le formulaire (`get-ElementById("formulaire").value`) le contenu de ce seul élément. Il est alors inutile d'afficher ni même de faire apparaître le menu déroulant (`get-ElementById("popups").innerHTML = "";`).

Il ne reste pluq qu'à étudier la fonction `choix()`.

```
function choix(evt) {
var thisDiv = (evt) ? evt.target : window.event.srcElement;
document.getElementById("formulaire").value = thisDiv.innerHTML;
document.getElementById("popups").innerHTML = "";
}
```

La fonction `choix()` était, rappelons-le, associée à l'événement du clic de la souris correspondant au choix de l'utilisateur d'un département dans la liste des départements suggérés. La ligne `(evt) ? evt.target : window.event. srcElement;` assure la compatibilité avec Firefox et Internet Explorer : en effet les deux navigateurs ne gèrent pas de la même façon les événements. Au clic de la souris sur un élément du menu déroulant, la valeur de celui-ci (`thisDiv.inner HTML`) est retenue pour être mise dans la ligne de texte (`getElementById("formulaire").value`). Dans ce cas le menu déroulant peut alors être effacé (`getElementById("popups").innerHTML = ""`).

Le script complet devient donc :

```
window.onload = initAll;
var xhr = null;
var departements = new Array();

function initAll() {
document.getElementById("formulaire").onkeyup = searchSuggest;
if (window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject) {
try {
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e) { }
}
}
if (xhr) {
xhr.onreadystatechange = setdepartements;
xhr.open("GET", "departements.xml", true);
xhr.send(null);
}
else {
alert("Désolé, votre navigateur n'est pas compatble avec AJAX");
}
}
function setdepartements() {
if (xhr.readyState == 4) {
if (xhr.status == 200) {
if (xhr.responseXML) {
var tousdepartements = xhr.responseXML.getElementsByTagName("item");
for (var i=0; i<tousdepartements.length; i++) {
departements[i] = tousdepartements[i].getElementsByTagName("dep")
[0].firstChild;
}
```

```
}
}
else {
alert("Il y a un problème avec la requête " + xhr.status);
}
}
}
function searchSuggest() {
var str = document.getElementById("formulaire").value;
document.getElementById("formulaire").className = "";
if (str != "") {
document.getElementById("popups").innerHTML = "";
for (var i=0; i<departements.length; i++) {
var ce_departement = departements[i].nodeValue;
if (ce_departement.toLowerCase().indexOf(str.toLowerCase()) == 0) {
var tempDiv = document.createElement("div");
tempDiv.innerHTML = ce_departement;
tempDiv.onclick = choix;
tempDiv.className = "suggestions";
document.getElementById("popups").appendChild(tempDiv);
}
}
var liste = document.getElementById("popups").childNodes.length;
if (liste == 0) {
document.getElementById("formulaire").className = "error";
}
if (liste == 1) {
document.getElementById("formulaire").value =
document.getElementById("popups").firstChild.innerHTML;
document.getElementById("popups").innerHTML = "";
}
}
}
function choix(evt) {
var thisDiv = (evt) ? evt.target : window.event.srcElement;
document.getElementById("formulaire").value = thisDiv.innerHTML;
document.getElementById("popups").innerHTML = "";
}
```

Il reste à construire le fichier des propriétés de feuilles de style, soit *departement*. css.

```
.suggestions { background-color: #FFF;
          padding: 2px 2px;
          border: 1px solid #000;}
.suggestions :hover { background-color: #9cf;}
#popups { position: absolute;}
#formulaire { font: 9pt arial, helvetica, sans-serif;}
#formulaire.error { background-color: #FFC;}
```

Commentaires :

- Les suggestions du script (*.suggestions*) seront affichées avec un arrière-plan de couleur blanche et avec une fine bordure.
- Au survol de la souris, ces suggestions (*.suggestions*) apparaissent avec un arrière-plan de couleur.
- L'encodage dans la ligne de texte (#formulaire) se réalise en police Arial, 9 points.
- Si l'utilisateur saisit une lettre ou un caractère qui ne correspond pas à la liste des départements (`#formulaire.error`), la ligne de texte apparaît en jaune.

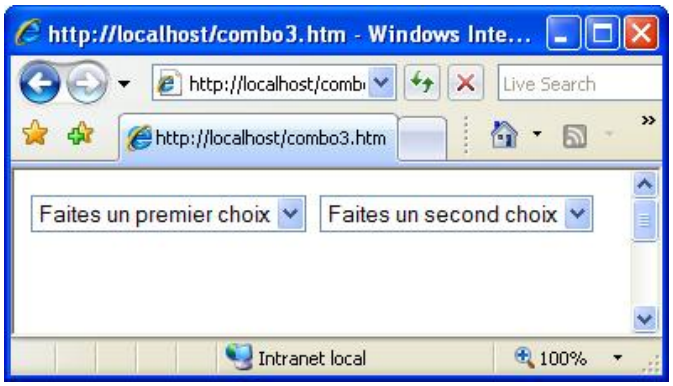
Ce script est globalement compatible entre Internet Explorer et Firefox.Cependant, les effets de style ne sont pas présents dans Firefox car, en effet, il faudrait utiliser *class* (au lieu de *className* sous Internet Explorer). Nous n'en avons pas tenu compte pour éviter de rallonger le script.

Un double menu déroulant

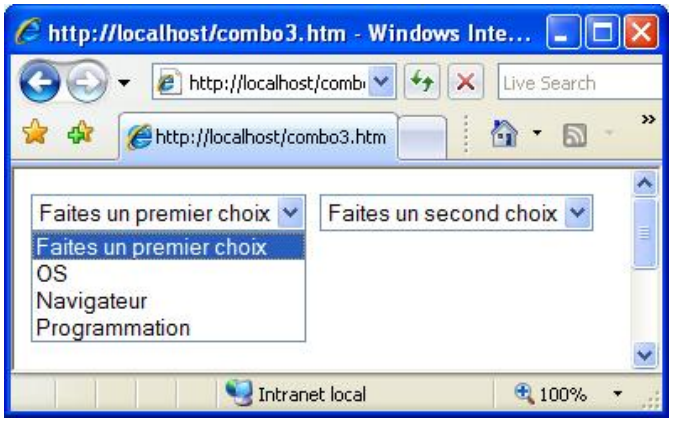
Cet exemple met en œuvre un double menu déroulant (balises `<select> ... </select>`) dans lequel, en fonction du choix effectué dans la première liste déroulante, les options de la seconde liste sont différentes, le tout, sans faire appel à un langage serveur comme PHP.

Les captures d'écran suivantes illustrent le script.

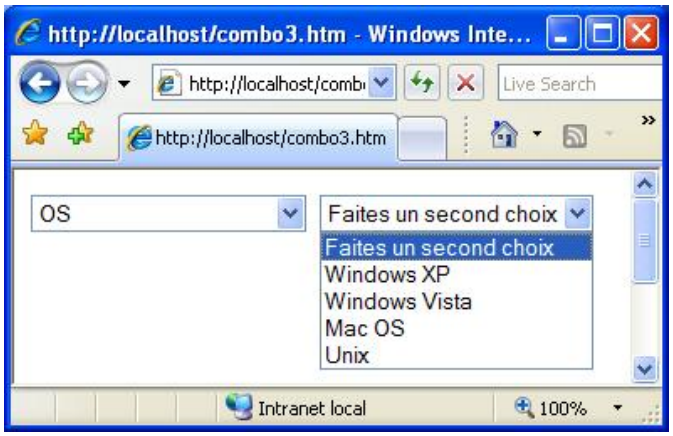
La situation de départ est la suivante :



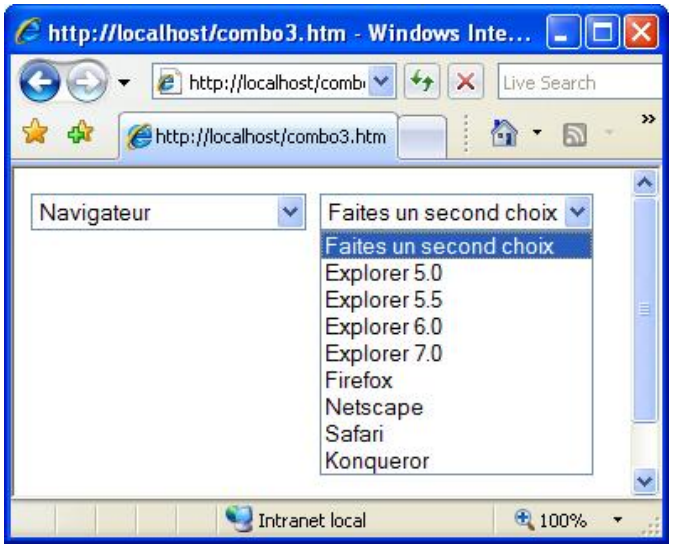
Le premier menu propose les choix : OS, Navigateur et Programmation.



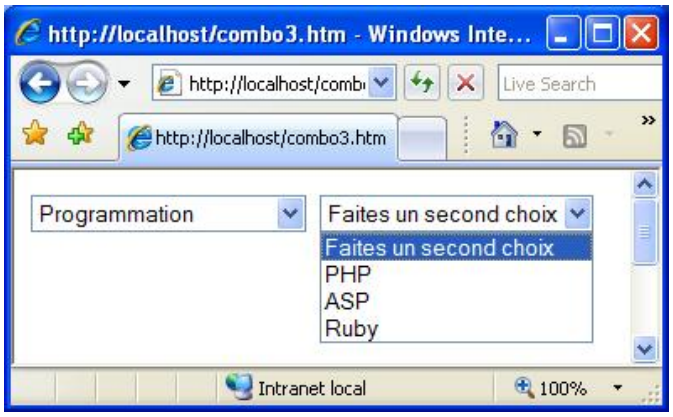
En choisissant l'option **OS** du premier menu, le second menu déroulant propose les options : **Windows XP, Windows Vista, Mac OS** et **Unix.**



Si l'option **Navigateur** est retenue, ces options deviennent : **Explorer 5.0, Explorer 5.5, Explorer 6.0, Explorer 7.0, Firefox, Netscape, Safari** et **Konqueror.**



Et enfin pour le dernier choix **Programmation**, la seconde liste se change en **PHP, ASP** et **Ruby**.



Concevons le fichier Xhtml de départ :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Script Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<form name="formulaires" action="">
<select name="form1" onChange="changement(this)">
<option>Faites un premier choix</option>
<option>OS</option>
<option>Navigateur</option>
<option>Programmation</option>
</select> 
<select name="form2">
<option>Faites un second choix</option>
</select>
</form>
</body>
</html>
```

Commentaires :

- Le formulaire (global) porte le nom suivant : `name="formulaires"`.
- Les menus déroulants portent respectivement les noms `name="form1"` et `name="form2"`.
- Le choix effectué par l'utilisateur dans le premier menu déroulant est pris en charge par la fonction `changement(this)` du script.

Les éléments de la seconde balise <select> sont présents sur le serveur sous la forme d'un fichier texte comportant les différentes options séparées par le signe étoile *.

Soit le fichier OS.txt :

```
*Windows XP*Windows Vista*Mac OS*Unix
```

Le fichier Navigateur.txt :

```
*Explorer 5.0*Explorer 5.5*Explorer 6.0*Explorer 7.0*Firefox*
Netscape*Safari*Konqueror
```

Et enfin, le fichier Programmation.txt :

```
*PHP*ASP*Ruby
```

Le script a pour mission, après avoir récupéré l'option sélectionnée au niveau du premier menu, de faire une requête sur le fichier texte concerné, de séparer les informations fournies par ce fichier et de les inclure dans le second menu.

Élaborons ce script, ligne par ligne :

```
<script type='text/JavaScript'>
//<![CDATA[
function changement(form1){
```

La fonction changement() est appelée dans le fichier Xhtml lorsque l'utilisateur choisit une option dans le premier formulaire (onChange="changement (this)).

```
var index = form1.selectedIndex;
```

La variable index identifie l'option sélectionnée par l'utilisateur dans le premier menu grâce à la propriété *selectedIndex*.

```
var valeur = form1[index].text;
```

La variable valeur retourne le texte associé à l'option (balise <option> ... </option>) sélectionnée.

```
var formulaireselements = document.formulaires.elements
var form2elements = formulaireselements["form2"];
```

Plus loin dans le script, il faudra pointer les éléments du second menu. La variable formulaireselements liste tous les éléments du formulaire global (formulaires). La variable form2elements ne reprend que les éléments du second menu (form2).

```
url = valeur + ".txt";
```

Le nom du fichier texte à utiliser lors de la requête HTTP est construit à partir de la variable valeur à laquelle on ajoute l'extension ".txt".

```
if (valeur != "Faites votre choix") {
```

Lorsque l'utilisateur effectue un choix dans le premier menu déroulant, soit lorsque la variable valeur est différente de l'option proposée par défaut (if (valeur != "Faites votre choix")), la procédure pour initialiser la requête peut être abordée.

```
var xhr = null;
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
```

L'objet xhr est créé de sorte à être compatible pour les différents navigateurs.

```
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
```

On s'assure que l'objet xhr a bien été créé (if(xhr)) et que la requête a bien abouti (xhr.readyState == 4 et xhr.status == 200).

```
var form2reponse = xhr.responseText;
```

La variable form2reponse contient le fichier retourné par la requête sous forme d'un fichier texte (xhr.responseText).

```
contenuform2(form2reponse,form2elements);
}
}
xhr.open("GET", url, true);
xhr.send(null);
}
}
}
```

La fonction `contenuform2()` qui va construire le second menu déroulant, peut alors être appelée, avec en arguments, le fichier résultant de la requête (`form2reponse`) et les éléments du second menu (`form2elements`).

```
function contenuform2(form2reponse,form2elements) {
var form2options = form2reponse.split("*");
```

La première tâche de la fonction `contenuform2()` consiste à retrouver les différents items contenus dans le fichier `form2reponse`. Pour ce faire, la méthode split()de l'objet String est utilisée : elle retourne sous forme de tableau, les éléments de la chaîne de caractères en tenant compte du séparateur étoile *.

```
form2elements = 1;
form2elements.length = form2options.length;
```

On informe le moteur JavaScript que le nombre d'éléments du second menu (`form2elements.length`) sera égal au nombre de sous-chaînes de caractères contenues dans le tableau retourné par la méthode split() (`form2options.length`).

Par l'intermédiaire d'une boucle *for*, il suffit, pour terminer le script, de transférer une par une, les sous-chaînes de caractères (`form2options[i]`) vers le texte associé aux options du second menu `<select>` (`form2elements[i].text`)

```
for (i=1; i < form2options.length; i++) {
form2elements[i].text = form2options[i];
}
}
//]]>
</script>
```

Le fichier Xhtml complet de cet exemple devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Script Ajax</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
function changement(form1){
var index = form1.selectedIndex;
var valeur = form1[index].text;
var formulaireselements = document.formulaires.elements
var form2elements = formulaireselements["form2"];
url = valeur + ".txt";
if (valeur != "Faites votre choix") {
var xhr = null;
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var form2reponse = xhr.responseText;
contenuform2(form2reponse, form2elements);
}
}
xhr.open("GET", url, true);
xhr.send(null);
}
}
}
function contenuform2(form2reponse, form2elements) {
```

```
var form2options = form2reponse.split("*");
form2elements.length = 1;
form2elements.length = form2options.length;
for (i=1; i <<+>form2options.length; i++) {
form2elements[i].text = form2options[i];
}
}
//]]>
</script>
</head>
<body>
<form name="formulaires" action="">
<select name="form1" onChange="changement(this)">
<option>Faites un premier choix</option>
<option>OS</option>
<option>Navigateur</option>
<option>Programmation</option>
</select> 
<select name="form2">
<option>Faites un second choix</option>
</select>
</form>
</body>
</html>
```

Pour rendre le script fonctionnel, il faudrait associer une action (par exemple, un lien vers une page) à l'option retenue dans le second menu déroulant.

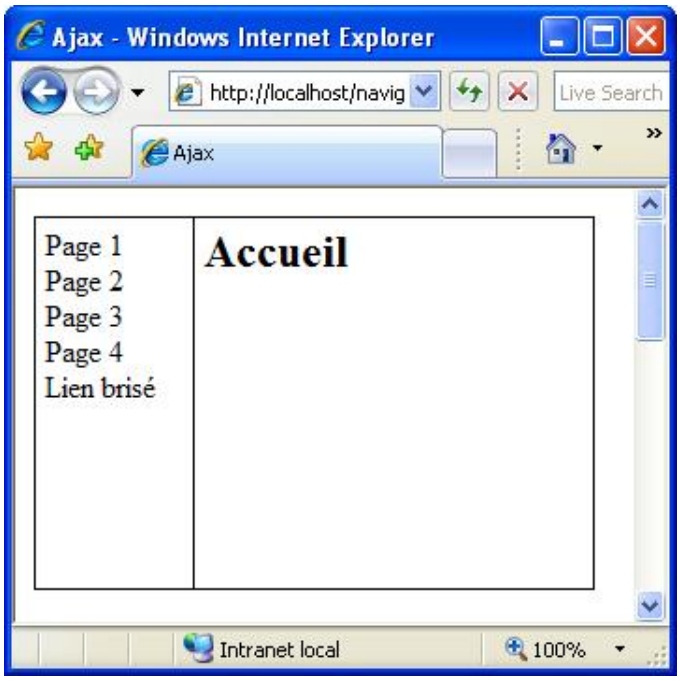
Ce script est parfaitement compatible entre Firefox et Internet Explorer.

Un menu de navigation

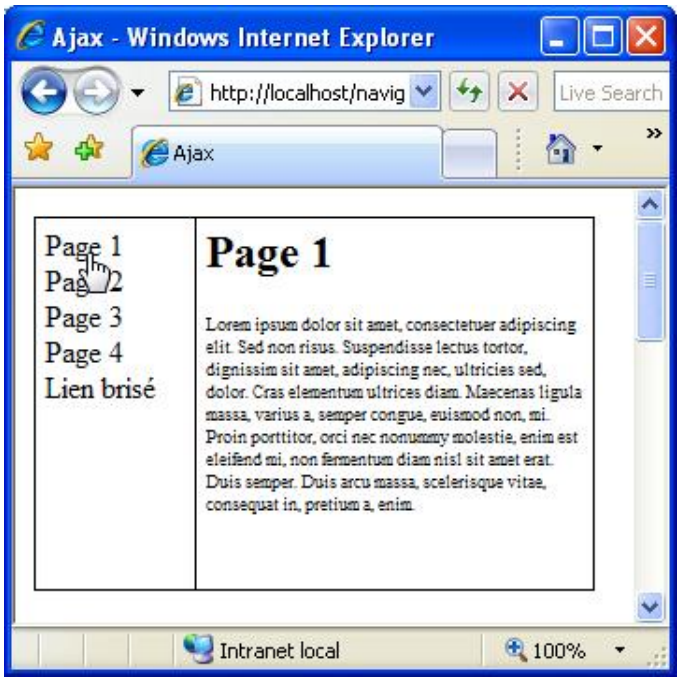
L'exemple suivant propose un tableau d'une ligne et de deux colonnes. En cliquant sur un lien du menu de navigation dans la première colonne, la page demandée s'affiche dans la seconde colonne.

Les pages affichées dans la seconde colonne sont présentes sur le serveur et sont appelées par une requête HTTP.

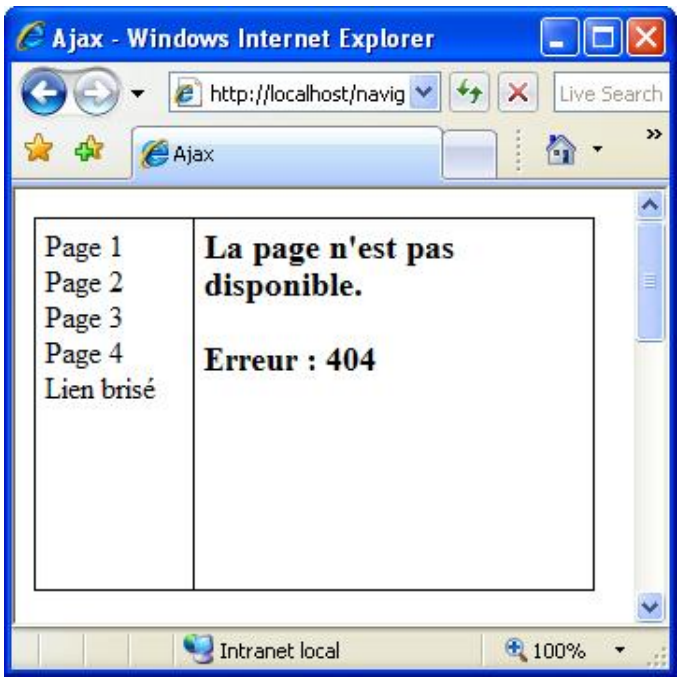
La capture d'écran illustre la situation d'origine.



Au clic sur le premier lien du menu de navigation, la page 1 s'affiche.



Le script prévoit la possibilité d'un lien brisé. Ce qui occasionne un message d'erreur illustré par la capture d'écran suivante.



Le fichier Xhtml de départ, avec quelques déclarations de style se présente comme suit :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<style type="text/css">
<!--
#tableau { border-collapse: collapse;
           border: solid black 1px;
           height : 200px;}
.td1 { border-right : solid black 1 px;
       padding: 5px;}
.td2 { padding: 5px;}
a:link { text-decoration:none; color:#000000; }
a:visited { text-decoration:none; color:#000000; }
a:hover { text-decoration:none; color:#000000; }
a:active { text-decoration:none; color:#000000; }
-->
</style>
</head>
<body>
<table width="300" id="tableau">
<tr>
<td valign="top" width="80" class="td1">
<a href="javascript:void(0)" onclick="open_url('page1.htm')">Page 1</a><br />
<a href="javascript:void(0)" onclick="open_url('page2.htm')">Page 2</a><br />
<a href="javascript:void(0)" onclick="open_url('page3.htm')">Page 3</a><br />
<a href="javascript:void(0)" onclick="open_url('page4.htm')">Page 4</a><br />
<a href="javascript:void(0)" onclick="open_url('xxxxx.htm')">Lien brisé</a>
</td>
<td valign="top" width="220" class="td2">
<div id="contenu"><h2>Accueil</h2></div>
</td>
</tr>
</table>
</body>
</html>
```

Le code de la balise de lien `<a> ... </a>` nécessite quelques commentaires :

- Habituellement, au clic d'un lien, la page spécifiée dans l'attribut *href* est chargée par le navigateur. Dans ce script, ce traitement classique du Xhtml n'est pas utilisé, ainsi nous utilisons à la place la fonction `open_url()`. Pour passer outre à toute action de l'attribut *href*, la notation `javascript:void(0)` est utilisée.

- Au clic du lien, la fonction `open_url()` est appelée et prend comme argument, l'adresse du fichier à afficher, par exemple `open_url('page1.htm')`.

Les fichiers page1.htm, page2.htm, page3.htm et page4.htm sont des fichiers Xhtml basiques. Voici le code du fichier page1.htm.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Page 1</title>
<meta http-equiv="Content-Type" content="text/html" />
</head>
<body>
<h2>Page 1</h2>
<p style="font-size: 9px">Lorem ipsum dolor sit amet, consectetuer
adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim
sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum
ultrices diam. Etc... </p>
</body>
</html>
```

Le script présente un codage dont la plupart des éléments devraient être assimilés à ce stade de l'étude.

```
<script type='text/JavaScript'>
//<![CDATA[
var xhr = null;
function open_url(url) {
```

Après avoir défini la variable `xhr` comme une variable globale, la fonction `open_url(url)`, appelée au clic sur un lien dans le code Xhtml, est initialisée.

```
var xhr = null;
if (window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject) {
try {
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e) { }
}
}
```

L'objet `xhr` est créé pour être compatible entre les navigateurs Firefox et Internet Explorer.

```
xhr.onreadystatechange = function() { response(); }
xhr.open("GET", url, true);
xhr.send(null);
}
```

La requête HTTP est effectuée. Aux changements d'états de celle-ci, la fonction `reponse()` est appelée.

```
function response() {
if (xhr.readyState == 4) {
if (xhr.status==200) {
document.getElementById("contenu").innerHTML = xhr.responseText;
}
```

La fonction `reponse()`, après avoir établi que les conditions habituelles sont remplies (`xhr.readyState == 4` et

xhr.status = 200), va écrire dans la balise , le fichier transmis sous forme d'un fichier texte. Il faut remarquer qu'il s'agit bien d'un fichier Xhtml car celui-ci répond bien à la définition de texte brut, agrémenté de balises permettant son affichage.

```
else {
document.getElementById("contenu").innerHTML = "<h3>La page n'est pas di
sponible. <br> <br> Erreur : " + xhr.status + "</h3";
}
}
}
```

Si la requête n'a pu aboutir, soit dans le cas à cause d'un lien brisé (si le fichier xxxxx.htm n'est pas présent sur le serveur), un message d'erreur est affiché.

```
//]]>
</script>
```

Fin du script.

Le fichier Xhtml complet, avec le script, se présente comme suit :

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Script Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
var xhr = null;
function open_url(url) {
if (window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject) {
try {
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e) { }
}
}
xhr.onreadystatechange = function() { reponse(); }
xhr.open("GET", url, true);
xhr.send(null);
}
function reponse() {
if (xhr.readyState == 4) {
if (xhr.status==200) {
document.getElementById("contenu").innerHTML = xhr.responseText;
}
else {
document.getElementById("contenu").innerHTML = "<h3>La page n'est pas
disponible.<br><br>Erreur : " + xhr.status + "</h3";
}
}
}
//]]>
</script>
<style type="text/css">
<!--
#tableau { border-collapse: collapse;
           border: solid black 1px;
           height : 200px;}
.td1 { border-right : solid black 1px;
       padding: 5px;}
.td2 { padding: 5px;}
```

```
a:link { text-decoration:none; color:#000000; }
a:visited { text-decoration:none; color:#000000; }
a:hover { text-decoration:none; color:#000000; }
a:active { text-decoration:none; color:#000000; }
-->
</style>
</head>
<body>
<table width="300" id="tableau">
<tr>
<td valign="top" width="80" class="td1">
<a href="javascript:void(0)" onclick="open_url('page1.htm')">Page
1</a><br />
<a href="javascript:void(0)" onclick="open_url('page2.htm')">Page
2</a><br />
<a href="javascript:void(0)" onclick="open_url('page3.htm')">Page
3</a><br />
<a href="javascript:void(0)" onclick="open_url('page4.htm')">Page
4</a><br />
<a href="javascript:void(0)" onclick="open_url('xxxxx.htm')">Lien
brisé</a>
</td>
<td valign="top" width="220" class="td2">
<div id="contenu"><h2>Accueil</h2></div>
</td>
</tr>
</table>
</body>
</html>
```

Le script est compatible entre Internet Explorer et Firefox.

Un script de login

Les utilisateurs autorisés à accéder à un site ont chacun un identifiant (utilisateur) et un mot de passe qui leur est propre dans un document XML. ces deux chaînes de caractères sont listées. Si les données encodées sont valides, l'accès au site est accordé. Dans le cas contraire, un message d'erreur apparaît.

Dans le cadre de l'exemple, les binômes (utilisateur/mot de passe) sont (Admin Admin), (User1 Pass1) et (User2 Pass2). Pour augmenter le nombre d'utilisateurs, il suffit d'ajouter leurs identifiants dans le fichier XML.

Les scripts de login et de mot de passe en JavaScript posent des problèmes de sécurité car le code source est accessible. Le script n'échappe pas à la règle malgré la complexité du code. Un login n'est efficace que s'il fait appel à un langage de programmation côté-serveur comme PHP ou ASP.

Le fichier Html de départ comporte deux lignes de texte et un bouton.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<form action="">
<table border="0">
<tr>
<td align="right">Utilisateur : </td>
<td><input type="text" id="idlogin" size="15" /></td>
</tr>
<tr>
<td align="right">Mot de passe : </td>
<td><input type="text" id="idpass" size="15" /></td>
</tr>
<tr>
<td align="center" colspan="2">
<br /><input type="button" onclick="log()" value="Me connecter" />
</td>
</tr>
</table>
</form>
</body>
</html>
```

Le fichier login.xml contient les utilisateurs autorisés :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<utilisateurs>
<user><login>Admin</login><pass>Admin</pass></user>
<user><login>User1</login><pass>Pass1</pass></user>
<user><login>User2</login><pass>Pass2</pass></user>
</utilisateurs>
```

La structure est la suivante :

```
<utilisateurs>
  <user>
    <login></login>
    <pass></pass>
  </user>
</utilisateurs>
```

Le script devient :

```
<script type='text/JavaScript'>
//<![CDATA[
var xhr = false;
function log() {
login=document.getElementById("idlogin").value;
pass=document.getElementById("idpass").value;
```

La variable `xhr` est définie (variable globale). La fonction log() est appelée dans le fichier Xhtml par le clic sur le bouton **Me connecter**, puis récupère les valeurs encodées dans les lignes de texte du document Xhtml, soit respectivement les variables `login` et `pass`.

```
if (window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject) {
try {
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e) { }
}
 }
```

L'objet correspondant à une requête HTTP est créé.

```
if (xhr) {
xhr.onreadystatechange = verif;
xhr.open("GET", "login.xml", true);
xhr.send(null);
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX");
}
}
```

La requête vers le fichier login.xml s'effectue en mode asynchrone. Au changement d'état de la requête (`xhr.onreadystatechange`), la fonction `verif()` est appelée.

```
function verif() {
if (xhr.readyState == 4) {
if (xhr.status == 200) {
if (xhr.responseXML) {
var login_el = new Array();
var logins = xhr.responseXML.getElementsByTagName("login");
```

La fonction `verif()`, après s'être assurée que la requête s'est effectuée de façon correcte (`xhr.readyState == 4` et `xhr.status == 200`), traite le fichier reçu comme un objet XML (`xhr.responseXML`).

La variable `login_el` est définie comme une variable indexée en vue de son utilisation ultérieure. Du fichier XML reçu (`xhr.responseXML`), le tableau *logins* liste tous les éléments (`getElementsByTagName("login");`) comportant la balise `<login>`.

```
for (var i=0; i<logins.length; i++) {
login_el[i] = logins[i].firstChild;
var celogin = login_el[i].nodeValue;
```

Une boucle *for* passe en revue, un par un, les éléments de la variable `logins` (soit `logins[i]`) et plus précisément le premier nœud enfant de celui-ci (`firstChild`). La variable celogin stocke le texte de ce noeud enfant (`nodeValue`). Elle contient donc le texte compris entre les balises `<login> ... </login>` du fichier XML.

```
if (login.toLowerCase()==celogin.toLowerCase()){
var cepass=logins[i].nextSibling.firstChild.nodeValue;
if (pass.toLowerCase()==cepass.toLowerCase()) {
window.location.href="xxx.htm";
}
```

On teste alors par la condition (`if (login==celogin)`) si la variable `celogin` est égale au texte saisi par l'utilisateur (pour rappel la variable login). Cette comparaison de variables s'affectue sur des chaînes en minuscules (méthode `toLowerCase()`*, rendant ainsi l'encodage *case insensitive*.

Il faut à présent s'occuper du mot de passe. La propriété nextSibling permet d'accéder aux balises `<pass>` au même niveau que la balise `<login>` dans l'arborescence du fichier XML. Le texte compris entre les balises `<pass> ... </pass>` est alors obtenu par `firstChild.nodeValue`. Celui-ci est stocké dans la variable cepass. Si cette dernière est égale à la saisie de l'utilisateur, soit à la variable `pass` (`if (pass==cepass)`), l'accès au site peut être accordé (`window.location.href`) vers une page quelconque, ici xxx.htm.

```
else {
alert("Login ou mot de passe incorrect");
}
}
}
}
}
```

Si à ce stade, le login ou le mot de passe ne correspond pas aux informations du fichier XML, un message d'erreur s'affiche dans une boîte d'alerte.


```
else {
alert("Il y eu un problème " + xhr.status);
}
}
}
//]]>
</script>
```

Le script est terminé.

Le fichier login.html complet devient :

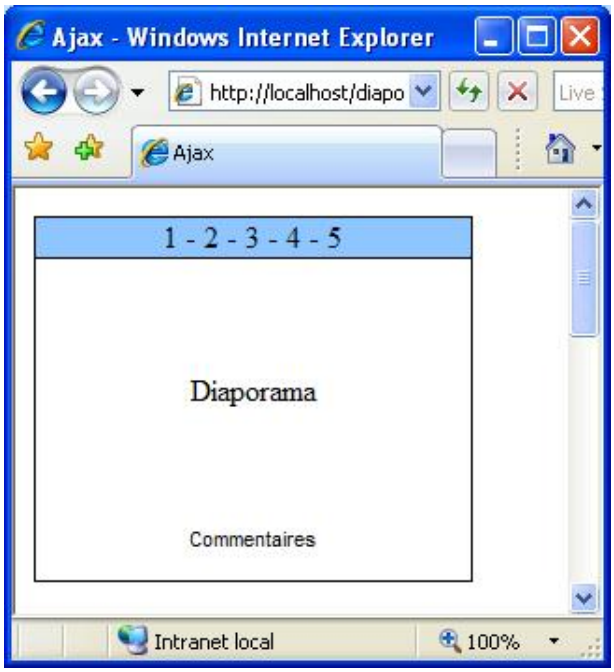
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script type='text/JavaScript'>
//<![CDATA[
var xhr = false;
function log() {
login=document.getElementById("idlogin").value;
pass=document.getElementById("idpass").value;
if (window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject) {
try {
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e) { }
}
}
if (xhr) {
xhr.onreadystatechange = verif;
xhr.open("GET", "login.xml", true);
xhr.send(null);
}
else {
alert("Votre navigateur n'est pas compatible avec AJAX");
}
}
function verif() {
if (xhr.readyState == 4) {
if (xhr.status == 200) {
if (xhr.responseXML) {
var logins = xhr.responseXML.getElementsByTagName("login");
var login_el = new Array();
for (var i=0; i<logins.length; i++) {
login_el[i] = logins[i].firstChild;
var celogin = login_el[i].nodeValue;
if (login.toLowerCase()==celogin.toLowerCase()){
var cepass=logins[i].nextSibling.firstChild.nodeValue;
if (pass.toLowerCase()==cepass.toLowerCase()) {
window.location.href="carnet.htm";
}
else {
alert("Login ou mot de passe incorrect");
}
}
}
}
}
else {
alert("Il y eu un problème " + xhr.status);
}
}
}
//]]>
</script>
</head>
<body>
<form action="">
<table border="0">
<tr>
<td align="right">Utilisateur : </td>
<td><input type="text" id="idlogin" size="15" /></td>
```

```
</tr>
<tr>
<td align="right">Mot de passe : </td>
<td><input type="text" id="idpass" size="15" /></td>
</tr>
<tr>
<td align="center" colspan="2">
<br /><input type="button" onclick="log()" value="Me connecter" />
</td>
</tr>
</table>
</form>
</body>
</html>
```

En l'état, le script ne fonctionne que sous Internet Explorer.

Un diaporama en AJAX

Soit un diaporama dont les éléments sont situés sur le serveur et appelés par une requête HTTP. Il faut noter que les éléments du diaporama comportent des images mais également du commentaire.



Le fichier Xhtml de départ est le suivant :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Script-Type" content="text/css" />
<style type="text/css">
<!--
```

```
#tableau { border-collapse: collapse;
           border: solid black 1px;}
.top { text-align:center;
       background-color: #9cf;}
.diapo { height: 140px;
      text-align: center;
      border-top: solid black 1px;
      vertical-align: middle;}
.comment {font-family: sans-serif;
          font-size: 11px;
          text-align: center;
          paddind: 3px;}
a:link { text-decoration:none; color:#000000; }
a:visited { text-decoration:none; color:#000000; }
a:hover { text-decoration:none; color:#000000; }
a:active { text-decoration:none; color:#000000; }
-->
</style>
</head>
<body>
<table width="235" id="tableau">
<tr>
<td class="top">
<p>
<a href="javascript:void(0)" onclick="go(0)">1</a> -
<a href="javascript:void(0)" onclick="go(1)">2</a> -
<a href="javascript:void(0)" onclick="go(2)">3</a> -
<a href="javascript:void(0)" onclick="go(3)">4</a> -
<a href="javascript:void(0)" onclick="go(4)">5</a>
</p>
</td>
</tr>
<tr>
<td class="diapo"><div id="image">Diaporama</div></td>
</tr>
<tr>
<td class="comment"><div id="commentaires">Commentaires
<br /> </div></td>
</tr>
</table>
</body>
</html>
```

Les images sont contenues dans les fichiers Xhtml, *image1.htm*, *image2.htm*, *image3.htm*, *image4.htm* et *image5.htm*.

Soit à titre d'exemple, pour le fichier image1.htm :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Image 1</title>
<meta http-equiv="Content-Type" content="text/html" />
</head>
<body>
<p>
<img border="0" src="html.jpg" width="108" height="120" />
</p>
</body>
</html>
```

Les commentaires sont repris dans de simples fichiers de texte soit *description1.txt*, *description2.txt*, *description3.txt*, *description4.txt* et *description5.txt*.

Le fichier description1.txt contient ainsi :

```
HTML 4 - Maîtrisez le code source (2ème édition)
```

Pour éviter les problèmes d'encodage des caractères, nous avons, sous Notepad, enregistré le fichier texte sous un codage UTF-8.

Voici le script pas à pas :

```
<script type='text/JavaScript'>
//<![CDATA[
galerie = new Array();
galerie[0] = "image1.htm";
galerie[1] = "image2.htm";
galerie[2] = "image3.htm";
galerie[3] = "image4.htm";
galerie[4] = "image5.htm";
description = new Array();
description[0] = "description1.txt";
description[1] = "description2.txt";
description[2] = "description3.txt";
description[3] = "description4.txt";
description[4] = "description5.txt";
var index;
```

Le script commence par la déclaration de deux tableaux (`Array`), l'un pour recueillir les images (galerie) et un autre pour le texte explicatif (`description`).

On déclare aussi la variable globale `index`.

```
function requetehttp() {
if (window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject) {
try {
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e) { }
}
}
}
```

Comme il faut effectuer, pour chaque écran du diaporama, deux requêtes (une pour l'image et une autre pour le texte), la création de l'objet `xhr` est confiée à la fonction `requetehttp()`. Ce qui allège le code du script.

```
function go(index) {
var url1 = galerie[index];
var url2 = description[index];
```

L'adresse du fichier contenant l'image est fournie par l'argument de la fonction `go()` au clic de la souris dans le fichier Html. Cet index est fourni au tableau `galerie`. On procède de même pour le texte descriptif.

```
requetehttp();
```

On effectue une première requête (requetehttp()) en vue de récupérer le fichier Xhtml contenant l'image.

```
xhr.onreadystatechange = function() { reponseimg(); }
xhr.open("GET", url1, true);
xhr.send(null);
```

Dès que des informations sont disponibles sur l'évolution de la requête (`xhr.onreadystatechange`), le script passe la main à la fonction `reponseimg()`.

```
requetehttp();
```

Une seconde requête (`requetehttp();`) est envoyée en vue de récupérer cette fois le fichier texte contenant le commentaire.

```
xhr.onreadystatechange = function() { reponsecom(); }
xhr.open("GET", url2, true);
xhr.send(null);
```

```
}
```

Dès que des informations sont disponibles sur l'évolution de cette requête (`xhr.onreadystatechange`), le script passe la main à la fonction `reponsecom()` pour traiter les commentaires.

```
function reponseimg() {
if ((xhr.readyState == 4) && (xhr.status==200)) {
document.getElementById("image").innerHTML = xhr.responseText;
}
}
```

La fonction `reponseimg()`, après s'être assuré que tout s'est déroulé correctement (`xhr.readyState == 4` et `xhr.status==200`), affiche dans le fichier Xhtml (`innerHTML`), à l'emplacement prévu à cet effet dans le document pour les images (`getElementById("image")`), la réponse à la requête sous forme de texte (`xhr.responseText`).

```
function reponsecom() {
if ((xhr.readyState == 4) && (xhr.status==200)) {
document.getElementById("commentaires").innerHTML = xhr.responseText;
}
}
```

La fonction `reponsecom()` procède de même pour les commentaires à l'emplacement prévu à cet effet dans le fichier Xhtml soit dans la balise `<div id="commentaires">` (`getElementById("commentaires")`).

Le script se termine par :

```
//]]>
</script>
```

Le fichier Xhtml complet est le suivant :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<script type='text/JavaScript'>
//<![CDATA[
galerie = new Array();
galerie[0] = "image1.htm";
galerie[1] = "image2.htm";
galerie[2] = "image3.htm";
galerie[3] = "image4.htm";
galerie[4] = "image5.htm";
description = new Array();
description[0] = "description1.txt";
description[1] = "description2.txt";
description[2] = "description3.txt";
description[3] = "description4.txt";
description[4] = "description5.txt";
var index;
function requetehttp() {
if (window.XMLHttpRequest) {
xhr<+>= new XMLHttpRequest();
}
else {
if (window.ActiveXObject) {
try {
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e) { }
}
}
}
function go(index) {
```

```
var url1 = galerie[index];
var url2 = description[index];
requetehttp();
xhr.onreadystatechange = function() { reponseimg(); }
xhr.open("GET", url1, true);
xhr.send(null);
requetehttp();
xhr.onreadystatechange = function() { reponsecom(); }
xhr.open("GET", url2, true);
xhr.send(null);
}
function reponseimg() {
if ((xhr.readyState == 4) && (xhr.status==200)) {
document.getElementById("image").innerHTML = xhr.responseText;
}
}
function reponsecom() {
if ((xhr.readyState == 4) && (xhr.status==200)) {
document.getElementById("commentaires").innerHTML = xhr.responseText;
}
}
//]]>
</script>
<style type="text/css">
<!--
#tableau { border-collapse: collapse;
           border: solid black 1px;}
.top {text-align:center;
      background-color: #9cf;}
.diapo {height: 140px;
        text-align: center;
        border-top: solid black 1px;
        vertical-align: middle;}
.comment {font-family: sans-serif;
          font-size: 11px;
          text-align: center;
          padding: 3px;}
          a:link { text-decoration:none; color:#000; }
          a:visited { text-decoration:none; color:#000; }
          a:hover { text-decoration:none; color:#000; }
          a:active { text-decoration:none; color:#000; }
-->
</style>
</head>
<body>
<table width="235" id="tableau">
<tr>
<td class="top">
<p>
<a href="javascript:void(0)" onclick="go(0)">1</a> -
<a href="javascript:void(0)" onclick="go(1)">2</a> -
<a href="javascript:void(0)" onclick="go(2)">3</a> -
<a href="javascript:void(0)" onclick="go(3)">4</a> -
<a href="javascript:void(0)" onclick="go(4)">5</a>
</p>
</td>
</tr>
<tr>
<td class="diapo"><div id="image">Diaporama</div></td>
</tr>
<tr>
<td class="comment"><div id="commentaires">Commentaires<br />
 </div></td>
</tr>
</table>
</body>
</html>
```

De nombreuses améliorations ou évolutions de ce script de diaporama sont possibles, citons :

- Un préchargement des images.
- Un effet d'affichage progressif pendant le chargement de l'image.
- Un affichage aléatoire des images et des commentaires.
- Etc.

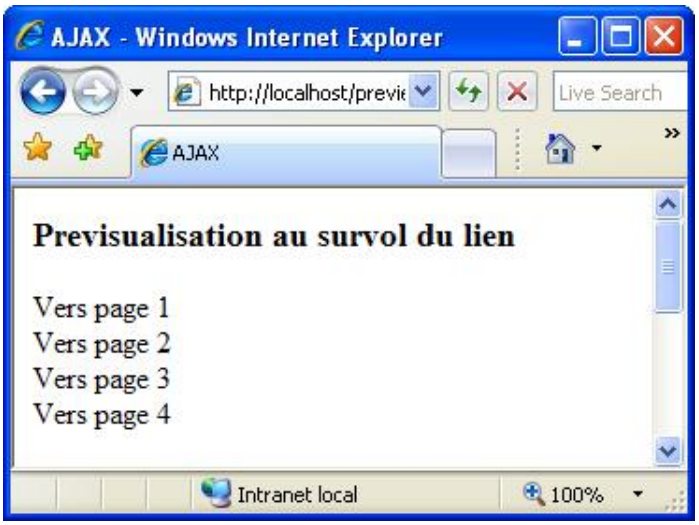
En l'état, le script ne fonctionne convenablement que sous Internet Explorer.

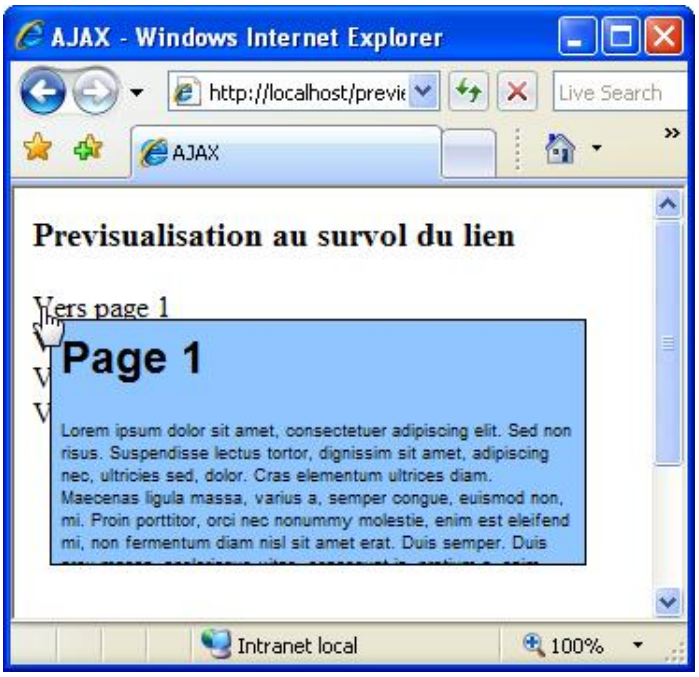
Une prévisualisation de la page

L'exemple suivant propose la prévisualisation de la page dans une boîte popup apparaissant au survol d'un lien par la souris.

Cet exemple illustre bien le confort de navigation et l'interactivité que peut apporter le concept AJAX à l'utilisateur.

Illustrons le script par des captures d'écran.





Le fichier Xhtml de départ avec la feuille de style qui gère la fenêtre de prévisualisation est le suivant :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Script-Type" content="text/css" />
<style>
<!--
#popup { position: absolute;
         top: 10px;
```

```
         left: 10px;
         background-color: #9cf;
         width: 275px;
         height: 120px;
         font: .9em Arial, sans-serif;
         padding: 5px;
         visibility: hidden;
         border: 1px black solid;
         clip: auto;
         overflow: hidden;}
a { color: black;
    text-decoration: none;}
-->
</style>
</head>
<body>
<h3>Previsualisation au survol du lien</h3>
<p>
<a href="page1.htm">Vers page 1</a><br />
<a href="page2.htm">Vers page 2</a><br />
<a href="page3.htm">Vers page 3</a><br />
<a href="xxxxx.htm">Vers page 4</a>
</p>
<div id="popup"> </div>
</body>
</html>
```

Les fichiers page1.htm, page2.htm et page3.htm sont des fichiers Xhtml quelconques.

Nous avons repris les fichiers du point Un menu de navigation du présent chapitre. Voici le code de la page1.htm :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Page 1</title>
<meta http-equiv="Content-Type" content="text/html" />
</head>
<body>
<h2>Page 1</h2>
<p style="font-size: 9px">Lorem ipsum dolor sit amet, consectetuer
adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim
sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices
 diam. Etc... </p>
</body>
</html>
```

Entamons l'étude du script.

```
<script type='text/JavaScript'>
//<![CDATA[
window.onload = initial;
var xhr = null;
var xpos;
var ypos;
```

Au chargement de la page (`window.onload`), la fonction `initial()` est appelée. Différentes variables sont initialisées : la variable `xhr`, la variable `xpos` qui détermine la position horizontale d'un point et la variable `ypos` qui détermine la position verticale d'un point.

```
function initial() {
var liens = document.getElementsByTagName("a");
for (var i=0; i< liens.length; i++) {
liens[i].onmouseover = previsualisation;
}
}
```

La fonction `initial()` a pour mission d'associer la fonction `previsualisation()` au survol par la souris des liens de la page. La variable `liens` liste toutes les balises `<a>` du document (`getElementsByTagName("a")`). Tous les liens sont

traités par une boucle *for* qui associe la fonction `previsualisation()` à l'événement survol (`onmouseover`) des liens (`liens[i]`).

```
function cacher() {
document.getElementById("popup").style.visibility = "hidden";
}
```

La fonction `cacher()` a pour but de cacher la fenêtre popup. Ceci se fait en affectant la déclaration de visibilité CSS (`style.visibility`) de la fenêtre de prévisualisation (`getElementById("popup")`) à la valeur hidden.

```
function previsualisation(evt) {
if (evt) {
var url = evt.target;
}
else {
evt = window.event;
var url = evt.srcElement;
}
```

La fonction `previsualisation()` possède comme argument l'événement `evt` de la fonction `initial()`, soit le survol de la souris sur un lien.

Les navigateurs Internet Explorer et Firefox n'ont pas la même gestion des événements, spécialement pour faire référence à l'élément d'où provient l'événement. Avec Firefox et les navigateurs de la famille Mozilla, il s'agit de la propriété `evt.target` qui fait référence à l'objet origine de l'événement. Avec les navigateurs de type Internet Explorer, il faut d'abord accéder à l'objet-événement par `window.event` puis à l'élément dans lequel a eu lieu l'événement par `srcElement`.

L'adresse (`var url`) associée au survol du lien par la souris est donnée par `evt.target` pour Firefox et par `window.event.srcElement` pour Internet Explorer. Le script est ainsi compatible.

```
xpos = evt.clientX;
ypos = evt.clientY;
```

La variable `xpos` note la position horizontale du pointeur de la souris au survol du lien et la variable `ypos`, la position verticale de celui-ci.

```
if (window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject) {
try {
xhr = new ActiveXObject ("Microsoft.XMLHTTP");
}
catch (e) { }
}
}
```

L'objet requête HTTP est créé de façon compatible avec les navigateurs Firefox et Internet Explorer.

```
if (xhr) {
xhr.onreadystatechange = afficher;
xhr.open("GET", url, true);
xhr.send(null);
}
else {
alert("Votre navigateur n'est pas compatible AJAX");
}
}
```

La requête est effectuée. Au premier changement d'état de celle-ci, la fonction `afficher()` est appelée.

```
function afficher() {
var previsu = document.getElementById("popup");
if (xhr.readyState == 4) {
if (xhr.status==200) {
previsu.innerHTML = xhr.responseText;
previsu.style.top = yPos+10 + "px";
```

```
previsu.style.left = xPos+10 + "px";
previsu.style.visibility = "visible";
previsu.onmouseout = cacher;
}
```

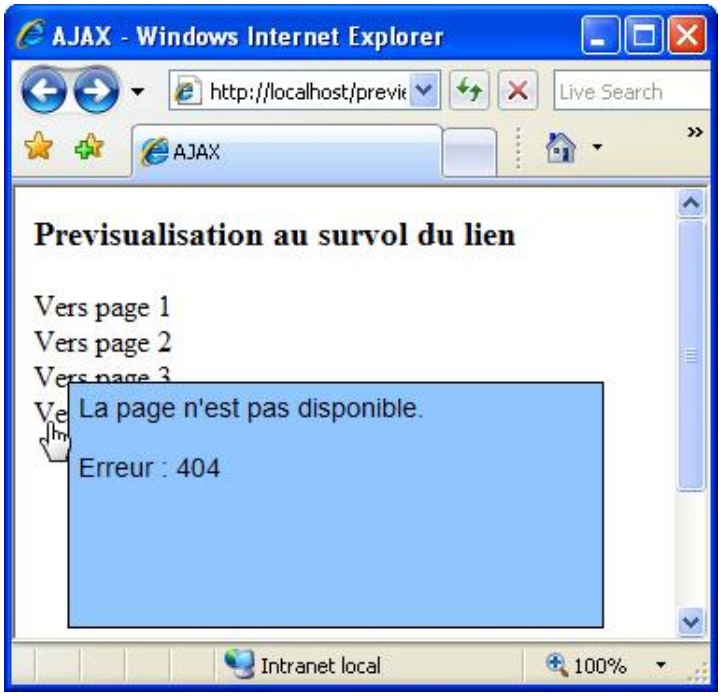
Dans la fonction `afficher()`, la variable `previsu` est d'abord définie comme étant la fenêtre popup (`getElementById ("popup")`). Si la requête s'est déroulée correctement (`xhr.readyState == 4` et `xhr.status == 200`), le contenu de celle-ci (`xhr.responseText`) est affiché dans la fenêtre popup (`previsu.innerHTML`).

On crée un léger décalage vertical (`previsu.style.top = (yPos+10) + "px"`) puis horizontal (`previsu.style.left = xPos+10 + "px"`).

La fenêtre popup devient alors visible (`previsu.style.visibility = "visible"`). Lorsque le pointeur de la souris quitte la fenêtre popup (`previsu.onmouseout`), la fonction `cacher()` est appellée.

```
else {
previsu.innerHTML = "La page n'est pas disponible. <br> <br> Erreur : "
 + xhr.status;
}
}
}
```

Si la requête ne s'effectue pas, un message d'erreur est affiché dans la fenêtre popup (`previsu.innerHTML`) avec le code (`xhr.status`) de l'erreur.




```
//]]>
</script>
```

Fin du script

Le fichier Xhtml complet devient :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>Ajax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Script-Type" content="text/css" />
<script type='text/JavaScript'>
//<![CDATA[
window.onload = initial;
```

```
var xhr = null;
var xPos;
var yPos;
function initial() {
var liens = document. getElementsByTagName("a");
for (var i=0; i< liens.length; i++) {
liens[i].onmouseover = previsualisation;
}
}

function cacher() {
document.getElementById("popup").style.visibility = "hidden";
}

function previsualisation(evt) {
if (evt) {
var url = evt.target;
}
else {
evt = window.event;
var url = evt.srcElement;
}

xPos = evt.clientX;
yPos = evt.clientY;

if (window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else {
if (window.ActiveXObject) {
try {
xhr = new ActiveXObject ("Microsoft.XMLHTTP");
}
catch (e) { }
}
}
if (xhr) {
xhr.onreadystatechange = afficher;
xhr.open("GET", url, true);
xhr.send(null);
}
else {
alert("Votre navigateur n’est pas compatible AJAX");
}
}
function afficher() {
var previsu = document.getElementById ("popup");
if (xhr.readyState == 4) {
if (xhr.status==200) {
previsu.innerHTML = xhr.responseText;
previsu.style.top = (yPos+10) + "px";
previsu.style.left = (xPos+10) + "px";
previsu.style.visibility = "visible";
previsu.onmouseout = cacher;
}
else {previsu.innerHTML = "La page n’est pas disponible. <br> <br>
Erreur : " + xhr.status;}
}
}
//]]>
</script>
<style>
<!--
#popup { position: absolute;
         top: 10px;
         left: 10px;
         background-color: #9cf;
         width: 275px;
```

```
         height: 120px;
         font: .9em arial, sans-serif;
         padding: 5px;
         visibility: hidden;
         border: 1px black solid;
         clip: auto;
         overflow: hidden;}
a {color: black;
   text-decoration: none;}
-->
</style>
</head>
<body>
<h3>Previsualisation au survol du lien</h3>
<p>
<a href="page1.htm">Vers page 1</a><br />
<a href="page2.htm">Vers page 2</a><br />
<a href="page3.htm">Vers page 3</a><br />
<a href="xxxxx.htm">Vers page 4</a>
</p>
<div id="popup"> </div>
</body>
</html>
```

Comme signalé plus haut, le script est compatible entre Internet Explorer et Firefox.

Un menu vertical dynamique

Arrivé au terme de votre apprentissage d'Ajax, il semble intéressant d'aborder un exemple, peut-être un peu plus complexe, qui mêle non seulement les requêtes asynchrones mais également le traitement dynamique des feuilles de style CSS et la création par le DOM, élements de la page.

Cet exemple est directement inspiré du script *AJAX Switch Menu* de Martial Boissonneault. Nous livrons ici une version simplifiée et personnalisée afin de rester dans la ligne pédagogique que nous nous sommes fixés dans le cadre de cet ouvrage.La version complète est téléchargeable sur le site www.getelementbyid.com/.

L'idée du script est de "confectionner" par le DOM, un menu de navigation vertical à partir des données reprises dans un simple document XML.

Ce script permet donc la réalisation, à la commande, de multiples menus de navigation par la seule modification du document XML. La création et la mise à jour d'un menu de navigation nécessite toujours un travail long et périlleux, pour contourner ce problème, ce script rend parfaitement modulable, et simple d'utilisation, la mise en place de ces menus.

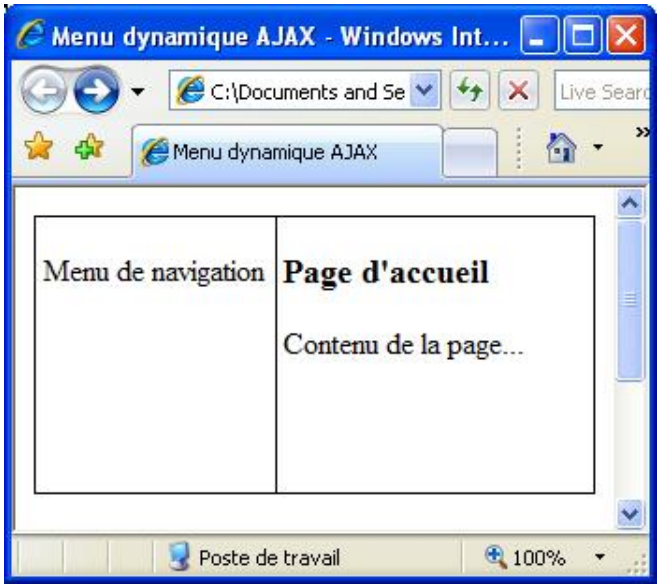
Soit notre page Html qui sert également de modèle pour les autres pages du site (menuAjax.htm).

```
<html>
<head>
<title>Menu dynamique AJAX</title>
```

```
<meta http-equiv="Content-Type" content="text/html;
charset="iso-8859-1" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Script-Type" content="text/css" />
<link rel="stylesheet" href="menuAjax.css" type="text/css"/>
<script type="text/javascript" src="menuAjax.js"></script>
</head>
<body onload="menuAjax('menuAjax.xml');">
<div id="menu" style="float:left;width:130px;">
<div id="affichageMenu"></div>
</div>
<div id="contenu">
<h3>Page d'accueil</h3>
<p>
Contenu de la page...
</p>
</div>
</body>
</html>
```

Plusieurs remarques s'imposent :

- Il n'y a pas de déclaration de *doctype*. Cette astuce est destinée à contourner les problèmes liés à l'interprétation des feuilles de style CSS par les navigateurs de la famille Internet Explorer.
- Il y a des liens vers des fichiers JavaScript et de feuilles de style CSS externes. Ce qui allège le code de la page Xhtml et facilite notre étude.
- La balise `<body>` appelle la fonction `menuAjax()` au chargement de la page.
- La page de départ comporte uniquement deux balises `<div>` dont la balise `<div id="affichageMenu"> ... </div>` qui contient le menu de navigation. À ce stade, elle est complètement vide ainsi que la balise `<div id="contenu"> ... </div>` qui accueille le texte de la page Web.



Le fichier XML (menuAjax.xml) :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<menus>
<menu texte="Home">
<sousmenu texte="Accueil" url="accueil.htm"></sousmenu>
</menu>
<menu texte="Produits">
<sousmenu texte="Produit 1" url="produit1.htm"></sousmenu>
<sousmenu texte="Produit 2" url="produit2.htm"></sousmenu>
<sousmenu texte="Produit 3" url="produit3.htm"></sousmenu>
```

```
</menu>
<menu texte="Services">
<sousmenu texte="Service 1" url="service1.htm"></sousmenu>
<sousmenu texte="Service 2" url="service2.htm"></sousmenu>
<sousmenu texte="Service 3" url="service3.htm"></sousmenu>
<sousmenu texte="Service 4" url="service4.htm"></sousmenu>
<sousmenu texte="Service 5" url="service5.htm"></sousmenu>
</menu>
</menus>
```

Ces balises comportent des attributs. Les items principaux (*Home, Produits* et *Services*) possèdent l'attribut `texte`. Les sous-menus, outre l'attribut `texte`, disposent de l'attribut `url` qui définit le lien correspondant.

Pour modifier ou mettre à jour le menu de navigation, il suffit simplement de modifier ce fichier XML.

Après chargement et traitement de ce fichier, notre fichier menuAjax.htm prend la forme suivante :

Le fichier externe de feuilles de style (menuAjax.css) :

```
#affichageMenu{
width: 100px;
background-color: #F3F9FE;
}
.menuOut {
cursor:pointer;
color: #21536A;
border: 1px solid;
background-color: #EAEEEE;
padding: 4px;
font-family:Verdana;
font-size: 11px;
border-color: #FEFEFE #C3C8CB #C3C8CB;
margin: 0px;
text-decoration: none;
padding-left:3px;
}
.menuOver {
cursor:pointer;
color: #21536A;
border: 1px solid;
background-color: #DDEEFF;
padding: 4px;
font-family:Verdana;
font-size: 11px;
border-color: #C3C8CB #C3C8CB #FEFEFE #C3C8CB;
```

```
margin: 0px;
text-decoration: none;
padding-left:3px;
}

.menuSelected {
cursor:pointer;
color: #21536A;
border: 1px solid;
background-color: #DDEEFF;
padding: 4px;
font-family:Verdana;
font-size: 11px;
border-color: #FEFEFE #C3C8CB #C3C8CB;
margin: 0px;
text-decoration: none;
padding-left:3px;
}

.sousmenu {
font-family:Verdana;
font-size: 10px;
padding-top:3px;
padding-bottom:4px;
padding-left:20px;
border-right: 1px solid #C3C8CB;
border-left: 1px solid #C3C8CB;
border-bottom: 1px solid #C3C8CB;
}

.sousmenu a {
color:#21536A;
text-decoration:none;
font-family:Verdana;
font-size: 10px;
}
.sousmenu a:hover {
color:#0000FF;
text-decoration:none;
font-family:Verdana;
font-size: 10px;
font-weight: bold;
}
```

Commentaires

- Les déclarartions `menuOut`, `menuOver` et `menuSelected` gèrent la présentation des items principaux. Au survol de ceux-ci par la souris, un petit effet visuel (un arrière-plan de couleur différente) est mis en place comme illustré par la capture d'écran suivante.

- La déclaration de style `sousmenu` prend en charge l'affichage des sous-menus.
- Un petit effet au survol des éléments de `sous-menu` est également prévu avec `sousmenu a` et `sousmenu a:hover`.

Étudions le fichier JavaScript externe (*menuAjax.js*) pas à pas.

```
var ie5 = (document.getElementById && document.all);
```

Comme il faut tenir compte d'incompatibilités notoires entre Internet Explorer et Firfox, la variable `ie5` permet de distinguer les navigateurs de la famille Microsoft.

Le script doit avant tout, initier une requête HTTP compatible entre Firefox et Internet Explorer afin de rapatrier les données du fichier menuAjax.xml vers le client.

```
var xhr = null;
function menuAjax(url){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n'est pas compatible avec AJAX...");
```

```
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
setXML(xmldocument);}
}
xhr.open("GET",url,true);
xhr.send(null);
}
}
```

Une fois la requête réussie, la fonction setXML() prend en charge les données du document XML et procède à l'affichage dynamique du menu de navigation.

Cette fonction est (bien entendu) assez complexe à mettre en place.

```
function setXML(xmldocument){
```

La fonction setXML() récupère le fichier XML (`xmldocument`) sous forme d'argument.

```
var menu = xmldocument.getElementsByTagName('menu');
```

La variable menu liste l'ensemble des balises `<menu>` de ce document XML.

```
for (var i=0;i<menu.length;i++){
```

Un boucle *for* passe en revue les différents éléments de la variable menu.

```
var p = document.createElement('p');
```

La variable p crée un nouveau paragraphe (createElement voir chapitre Le DOM (Document Object Model) - Modifier la Hiérarchisation) .

```
var attributs=menu[i].attributes;
var txt = attributs.getNamedItem("texte").nodeValue;
```

Le script accède aux attributs de la présente balise `<menu>` et extrait son contenu au moyen de l'attribut `texte` (voir chapitre Le DOM - Accéder aux attributs et chapitre L'approche AJAX - Récupérer et traiter du XML).

```
p.appendChild(document.createTextNode("> " + txt));
```

Le script crée un nœud texte (`createTextNode` voir chapitre Le DOM (Document Objet Model - Modifier la Hiérarchisation - La méthode createTextNode), avec en paramètre le texte identifié à la ligne de code précédente (variable `txt`) et ajouté comme dernier élément à la variable p.

```
if(ie5){p.style.cssText= 'width: 100%;';}
```

Cette ligne corrige un bug d'Internet Explorer et force l'élément à prendre tout l'espace disponible dans l'élément déterminé.

```
ie5 ? p.setAttribute("className", "menuSelected") :
p.setAttribute("class", "menuSelected");
```

La méthode `setAttribute()` permet d'ajouter ou de modifier à un élément, un attribut et sa valeur. La syntaxe en est `setAttribute("attribut","valeur")`.

Comme la gestion en JavaScript des styles est différente avec Internet Explorer et Firefox, la notation *className* est utilisée pour Internet Explorer et la notation *class* est utilisée pour Firefox. L'attribut prend la valeur *menuSelected*.

```
document.getElementById('affichageMenu').appendChild(p);
```

Après avoir accédé à l'identifiant de la balise `<div>` consacrée à l'affichage du menu (`affichageMenu`) dans le document Xhtml, l'élément p lui est ajouté comme dernier élément enfant (`appendChild`).

```
p.setAttribute("id", "menu" + i);
```

Un attribut *id* ayant pour valeur "menu" + i (ième itération) est affecté à l'élément p.

```
ie5 ? p.setAttribute("className", "menuOut") : p.setAttribute("class",
"menuOut");
```

L'attribut *className* ou *class* est modifié pour lui donner la valeur *menuOut*.

```
p.onmouseover = new Function("changerClasse('menu" + i + "','menuOver');");
p.onmouseout = new Function("changerClasse('menu" + i + "','menuOut');");
```

La déclaration de style `menuOver` est associée à l'événement `onmouseover` par la fonction `changerClasse()` (voir plus bas dans le script). La déclaration de style `menuOut` reprend la main à l'événement `onmouseout`.

La partie consacrée à la construction de l'item dans le menu de navigation est maintenant terminée. Le script passe maintenant à l'élaboration des sous-menus.

```
var d = document.createElement('div');
```

La variable d crée (createElement voir chapitre Le DOM (Document Object Model) - Modifier la Hiérarchisation) une nouvelle division (balise).

```
d.setAttribute("id", "sub" + i);
```

Un attribut *id* avec comme valeur *subi* lui est affecté par la méthode `setAttribute()`,

```
ie5 ? d.setAttribute("className", "sousmenu") : d.setAttribute("class",
"sousmenu");
```

La déclaration de style sousmenu lui est affectée (voir fichier de style externe menuAjax.css).

```
d.style.cssText = 'display:block;';
```

La division ainsi créée, est affichée par la propriété CSS de style *display*.

```
document.getElementById('affichageMenu').appendChild(d);
```

Et la division est jointe comme dernier enfant au menu de navigation en construction.

```
var sousMenu = menu[i].getElementsByTagName('sousmenu');
```

La variable `sousMenu` liste par item, les différentes balises <sousmenu> (voir le fichier menuAjax.xml).

```
for (var j=0;j<sousMenu.length;j++){
```

Une boucle *for* traite alors en revue, un par un, les différents éléments de la variable `sousMenu`.

```
var a = document.createElement('a');
```

Une balise de lien (createElement) est créée.

```
var attributs= sousMenu[j].attributes;
var Url = attributs.getNamedItem("url").nodeValue;
```

L'attribut url est recherché parmi les attributs de la balise <sousmenu> et sa valeur est déduite.

```
a.href = Url;
```

Cette valeur constitue le lien (`href`) attaché à l'élément a.

```
var txt = attributs.getNamedItem("texte").nodeValue;
```

L'attribut `texte` est cherché parmi les attributs de la balise <sousmenu> et sa valeur est reprise.

```
var spn = document.createElement('span');
a.appendChild(spn);
```

Une balise (variable spn) est créée : elle est jointe comme dernier enfant à l'élément a.

```
spn.appendChild(document.createTextNode("- " + txt));
```

Un nœud texte est créé prenant pour valeur la variable txt : il est joint à l'élément spn.

```
d.appendChild(a);
```

Le tout est ajouté comme dernier enfant à la variable d (pour rappel, les divisions du sous-menu).

```
var b = document.createElement('br');
d.appendChild(b);
}
}
}
```

Une balise
 (createElement) est créée et ajoutée à l'élément d, ce qui permettra au sous-menu suivant de s'afficher sur une nouvelle ligne.

```
function changerClasse(menu, newClass) {
if (document.getElementById) {
document.getElementById(menu).className = newClass;
}
}
```

Le fichier JavaScript menuAjax.js complet :

```
var ie5 = (document.getElementById && document.all);
function setXML(xmldocument){
var menu = xmldocument.getElementsByTagName('menu');
for (var i=0;i<menu.length;i++){
var p = document.createElement('p');
var attributs=menu[i].attributes;
var txt = attributs.getNamedItem("texte").nodeValue;
p.appendChild(document.createTextNode("> " + txt));
if(ie5){p.style.cssText= 'width: 100%;';}
ie5?p.setAttribute("className", "menuSelected"):p.setAttribute("class",
"menuSelected"); document.getElementById('affichageMenu').appendChild(p);
p.setAttribute("id", "menu" + i);
ie5?p.setAttribute("className", "menuOut"):p.setAttribute("class", "menuOut");
p.onmouseover = new Function("changerClasse('menu" + i + "','menuOver');");
p.onmouseout = new Function("changerClasse('menu" + i + "','menuOut');");
var d = document.createElement('div');
d.setAttribute("id", "sub" + i);
ie5?d.setAttribute("className", "sousmenu"):d.setAttribute("class", "sousmenu");
d.style.cssText = 'display:block;';
document.getElementById('affichageMenu').appendChild(d);
var sousMenu = menu[i].getElementsByTagName('sousmenu');
for (var j=0;j<sousMenu.length;j++){
var a = document.createElement('a');
var attributs= sousMenu[j].attributes;
var Url = attributs.getNamedItem("url").nodeValue;
a.href = Url;
var txt = attributs.getNamedItem("texte").nodeValue;
var spn = document.createElement('span');
a.appendChild(spn);
spn.appendChild(document.createTextNode("- " + txt));
d.appendChild(a);
var b = document.createElement('br');
d.appendChild(b);
}
}
}
var xhr = null;
function menuAjax(url){
if(window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
```

```
}
else if(window.ActiveXObject){
xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else{
alert("Votre navigateur n’est pas compatible avec AJAX...");
}
if(xhr) {
xhr.onreadystatechange = function(){
if(xhr.readyState == 4 && xhr.status == 200){
var xmldocument = xhr.responseXML;
setXML(xmldocument);}
}
xhr.open("GET",url,true);
xhr.send(null);
}
}
function changerClasse(menu, newClass) {
if (document.getElementById) {
document.getElementById(menu).className = newClass;
}
}
```

Ce script est compatible entre Internet Explorer et Firefox.

Mots réservés JavaScript

abstract	final	protected
boolean	finally	public
break	float	return
byte	for	short
case	function	static
catch	goto	super
char	if	switch
class	implements	synchronized
continue	import	this
const	in	throw
debugger	instanceOf	throws
default	int	transient
delete	interface	true
do	label	try
double	long	typeof
else	native	var
extends	new	void
enum	null	volatile
export	package	while
false	private	with

Outre les mots réservés énumérés ci-dessus, il est préférable d'éviter les mots suivants dans les noms de variables. Ces noms sont utilisés pour les objets, méthodes ou propriétés par le langage JavaScript.

alert all anchor anchors area array assign blur button checkbox clearTimeout clientInformationclose closed confirm crypto date defaultStatus document element elements embed embeds escape eval event	fileUpload focus form forms frame frames frameRate function getClass hidden history image images isNaN java JavaArray JavaClass JavaObject JavaPackage innerHeight innerWidth layer layers length link location	Math mimeTypes name navigate navigator netscape Number Object offscreen onblur onerror onfocus onload onunload open opener option outerHeight outerWidth packages pageXOffset pageYOffset parent parseFloat parseInt password	plugin prompt prototype radio reset screenX screenY scroll secure select self setTimeout status String submit sun taint text textarea top toString unescape untaint valueOf window