Premières applications Web 2.0 avec Ajax et PHP

Avec 40 ateliers pour réaliser facilement des moteurs Ajax-PHP avec JavaScript ou jQuery

Jean-Marie Defrance

EYROLLES

Premières applications Web 2.0 avec Ajax et PHP

CHEZ LE MÊME ÉDITEUR

Du même auteur

J.-M. DEFRANCE. – **PHP/MySQL avec Dreamweaver 8.**
N°11771, 2006, 632 pages.

J.-M. DEFRANCE. – **PHP/MySQL avec Flash 8.**
N°11971, 2006, 752 pages.

Dans la même collection

T. TEMPLIER, A. GOUGEON. – **JavaScript pour le Web 2.0.**
N°12009, 2007, 492 pages.

M. PLASSE. – **Développer en Ajax.**
N°11965, 2006, 314 pages.

C. PORTENEUVE. – **Bien développer pour le Web 2.0.**
Bonnes pratiques Ajax.
N°12028, 2006, 560 pages.

R. GOETTER. – **CSS 2.** *Pratique du design web.*
N°11976, 2e édition 2007, 310 pages.

H. WITTENBRIK. – **RSS et Atom.** *Fils et syndications.*
N°11934, 2006, 216 pages.

W. ALTMANN et al. – **Typo3.**
N°11781, 2006, 532 pages.

D. THOMAS, D. HEINEMEIER HANSSON. – **Ruby on Rails.**
N°11746, 2006, 590 pages.

M. MASON. – **Subversion.**
Pratique du développement collaboratif avec SVN.
N°11919, 2006, 206 pages.

E. DASPET ET C. PIERRE DE GEYER. – **PHP 5 avancé.**
N°12004, 3e édition 2006, 800 pages.

M. KOFLER. – **MySQL 5.**
Guide de l'administrateur et du développeur.
N°11633, 2005, 692 pages.

J. DUBOIS, J.-P. RETAILLÉ, T. TEMPLIER. –
Spring par la pratique.
Mieux développer ses applications Java/J2EE avec Spring, Hibernate, Struts, Ajax...
N°11710, 2006, 518 pages.

A. PATRICIO. – **Hibernate 3.0.**
N°11644, 2005, 336 pages.

K. DJAFAAR. – **Eclipse et JBoss.**
N°11406, 2005, 656 pages + CD-Rom.

J. WEAVER, K. MUKHAR, J. CRUME. – **J2EE 1.4.**
N°11484, 2004, 662 pages.

G. LEBLANC. – **C# et .NET 2.0.**
N°11778, 2006, 700 pages.

T. ZIADÉ. – **Programmation Python.**
N°11677, 2006, 530 pages.

O. DECKMYN, P.-J. GRIZEL. – **Zope.**
N°11706, 3e édition 2005, 810 pages.

Autres ouvrages sur le développement Web

M. NEBRA. – **Créer son site web en XHTML et CSS** (Accès libre).
N°11948, à paraître en octobre 2006.

D. MERCER. – **Créer son site e-commerce avec osCommerce** (Accès libre).
N°11932, 2006, à paraître en octobre 2006.

D. SHEA, M. HOLZSCHLAG. – **Le Zen des CSS.**
N°11699, 2005, 296 pages.

V. CARON, Y. FORGERIT et al. – **SPIP 1.8** (Les Cahiers du programmeur).
N°11428, 2005, 450 pages.

G. PONÇON. – **Best practices PHP** (Architecte logiciel).
N°11676, 2005, 490 pages.

J. MOLIÈRE. – **Cahier du programmeur J2EE.**
Conception et déploiement J2EE.
N°11574, 2005, 234 pages.

R. FLEURY. – **Cahier du programmeur Java/XML.**
Méthodes et frameworks : Ant, Junit, Eclipse, Struts-Stxx, Cocoon, Axis, Xerces, Xalan, JDom, XIndice…
N°11316, 2004, 228 pages.

Premières applications Web 2.0 avec Ajax et PHP

Jean-Marie Defrance

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Mise en page : TyPAO
Dépôt légal : janvier 2008
N° d’éditeur : 7736
Imprimé en France

Remerciements

Je remercie Matthieu Montaudouin, Thorsten Kruske et Alain Bertaut pour leur aide dans la rédaction de cet ouvrage ainsi que Matthieu Amiot et Claire Defrance pour leur collaboration dans la réalisation des illustrations.

Table des matières

PARTIE I – INTRODUCTION À AJAX

CHAPITRE 6

CHAPITRE 7

CHAPITRE 9

CHAPITRE 12

CHAPITRE 13

CHAPITRE 14

ANNEXE B

Partie I

Introduction à Ajax

Dans cette première partie, nous vous proposons de découvrir ce que représente Ajax et la place que cette technologie prend au sein du Web 2.0. Pour bien appréhender le fonctionnement d'un nouveau concept il est souvent intéressant de connaître son historique, aussi nous commencerons par un rapide récapitulatif des étapes clés de l'évolution du Web pour montrer en quoi le Web 2.0 est devenu incontournable dans l'Internet d'aujourd'hui et de demain. Savoir ce qu'on peut réaliser avec Ajax et connaître les grands acteurs d'Internet qui l'utilisent vous convaincra de l'intérêt que vous pourrez tirer de cette technologie dans vos futures applications Web. Enfin, comprendre comment fonctionne Ajax, détailler ses avantages et ses inconvénients, et présenter l'objet XMLHttpRequest qui constitue le cœur de cette technologie, nous a semblé un passage obligé avant que vous ne réalisiez vos premières applications Ajax-PHP.

1

Chronologie du Web

Depuis le début d'Internet, le Web a évolué par paliers et plusieurs phases se sont succédées avant d'obtenir les applications en ligne que l'on utilise aujourd'hui. Curieusement, les périodes charnières de ces évolutions sont espacées d'environ 5 ans et si l'on anticipe l'évolution à venir, il y a de fortes chances que 2010 soit l'année qui marquera la maturité du Web 2.0.

1990 : début du Web statique

Au début du Web, les pages HTML se limitaient à l'affichage de simples textes et à quelques illustrations (dont l'affichage était d'ailleurs souvent bloqué pour améliorer la fluidité de la navigation sur les réseaux à faible débit de l'époque). Au fil des années, avec l'avènement d'Internet tel qu'on le connaît, les exigences des utilisateurs ont évolué. En effet, la seule interactivité possible sur les pages HTML était l'affichage une nouvelle page lors d'un clic sur un lien hypertexte. Il était donc impossible d'obtenir le moindre effet visuel (comme un simple roll-over sur une image par exemple) sans avoir recours à une technologie complémentaire. De plus, l'envoi d'une requête au serveur Web, suite à un clic sur un lien hypertexte par exemple, engendrait un cycle de traitement long et fastidieux qui freinait considérablement la réactivité des applications sur des réseaux et des serveurs souvent sous-dimensionnés pour le trafic sans cesse croissant de l'époque.

Les sites statiques

Les sites statiques sont constitués d'un ensemble de pages HTML reliées entre elles par des liens hypertextes qui permettent de naviguer de l'une à l'autre. Le protocole utilisé pour transférer des informations Web sur Internet s'appelle HTTP. Une requête HTTP (*http://www.eyrolles.com/page.htm* par exemple) est envoyée vers le serveur afin d'accéder à la page désirée et de la visualiser dans le navigateur du poste client (voir étape 1 de la figure 1-1).

Lorsque le serveur Web reçoit cette requête, il recherche la page demandée parmi toutes les pages HTML présentes sur le site concerné et la renvoie ensuite au client (voir étape 2 de la figure 1-1).

☞

Les sites statiques (*suite)*

Le code HTML reçu par le poste client est alors interprété et affiché par le navigateur (voir étape 3 de la figure 1-1). C'est ce qu'on appelle l'architecture client-serveur (je demande, on me sert) : le client est le navigateur Internet (Internet Explorer, Firefox...) et le serveur est le serveur Web sur lequel sont stockées les informations du site Internet.

Ce type de site est très simple à réaliser et la plupart des premiers sites ont été conçus sur ce modèle. Cependant, ce concept est limité car il manque cruellement d'interactivité.

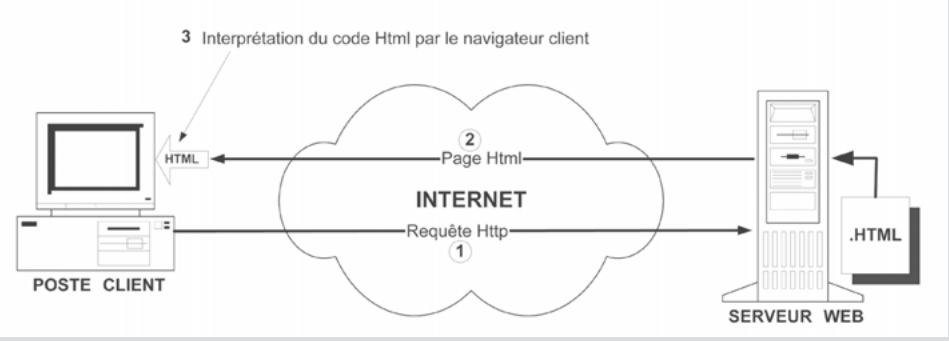


Figure 1-1

L'architecture client-serveur d'un site statique : le poste client envoie au serveur une requête HTTP ; le serveur Web recherche puis fournit au poste client la page demandée, qui est ensuite interprétée par le navigateur.

HTTP

HTTP (Hyper Text Transfer Protocol) est le protocole utilisé principalement par toutes les applications Web : celui-ci permet l'échange de données entre un serveur Web et un navigateur installé sur un poste client. Ainsi, le navigateur peut envoyer au serveur Web une requête formalisée sous la forme de l'adresse de la page ciblée (URL) et recevoir en réponse la page HTML correspondante qui remplacera complètement la page actuellement affichée dans le navigateur. Comme la plupart des protocoles applicatifs d'Internet (FTP, SMTP, POP3…) HTTP s'appuie sur un autre protocole de transport, le TCP/IP (Transmission Control Protocol/Internet Protocol) qui assure l'acheminement des données par paquets (l'ensemble des données est scindé en petits paquets pendant la phase du transfert) d'un point à un autre.

(x)HTML

HTML (HyperText Markup Language) est le langage de description d'une page Web. Ce langage s'appuie sur un ensemble de balises standards interprétées par le navigateur afin de définir le contenu et la mise en forme de la page.

Le XHTML (eXtensible HyperText Markup Language) quant à lui, est une évolution du précédent langage conforme aux contraintes du XML et impose le respect de certaines règles pour qu'une page soit bien formée (noms des balises en minuscule, attributs des balises obligatoirement encadrés par des guillemets, fermeture obligatoire de toutes les balises…).

1995 : le Web orienté client

Pour remédier au manque d'interactivité et aux problèmes d'engorgement des réseaux et de saturation des serveurs Web, les développeurs ont commencé à mettre en œuvre diverses technologies côté client afin de délester le serveur (réduisant ainsi le trafic sur le réseau) de tâches pouvant être traitées directement par le navigateur. Ainsi chaque éditeur

de navigateur Web a rapidement commencé à implémenter dans son logiciel des interpréteurs pour son propre langage. Aussi, Netscape avec JavaScript et Microsoft avec JScript permirent de pouvoir enfin exécuter des scripts côté client ; mais le non respect des standards du W3C de certains éditeurs compliquait le travail des développeurs qui devaient tenir compte des différences d'interprétation de leur code d'un navigateur à l'autre.

Ces nouvelles technologies client ont soulevé aussi un autre problème : celui de la sécurité des utilisateurs. Ainsi les éditeurs des navigateurs durent rapidement ajouter dans les options de leurs logiciels la possibilité de désactiver l'exécution des différentes technologies client pour répondre à la crainte des utilisateurs. Le fait même que certains navigateurs ne puissent plus exécuter les scripts client a constitué un frein important à leur usage car les développeurs devaient alors prévoir des alternatives en mode dégradé pour permettre à tous les utilisateurs d'utiliser leur application.

Par la suite, d'autres sociétés ont développé des programmes propriétaires (applets Java, ActiveX, Flash…) pouvant être intégrés dans une page Web et exécutés dans le navigateur grâce à un plug-in (extension du navigateur). Le Web disposait alors d'une pléthore de technologies client mais le manque de standardisation et l'hétérogénéité des navigateurs en rendaient leur usage très difficile.

Les sites interactifs côté client

La solution la plus simple pour créer de l'interactivité consiste à intégrer quelques lignes de code JavaScript dans une page HTML. Lorsqu'une requête HTTP appelle la page HTML (voir étape 1 de la figure 1-2), le serveur Web la retourne au poste client afin qu'elle puisse être interprétée comme une page HTML classique (voir étapes 2 et 3 de la figure 1-2). Le script inclus dans la page est ensuite traité par le navigateur (donc côté client) dès que survient l'événement pour lequel il a été programmé (voir étape 4 de la figure 1-2).

Les scripts côté client sont simples à mettre en œuvre car ils ne nécessitent pas une infrastructure serveur spécifique. De plus, ils sont très réactifs car le script s'exécute directement sur le poste client.

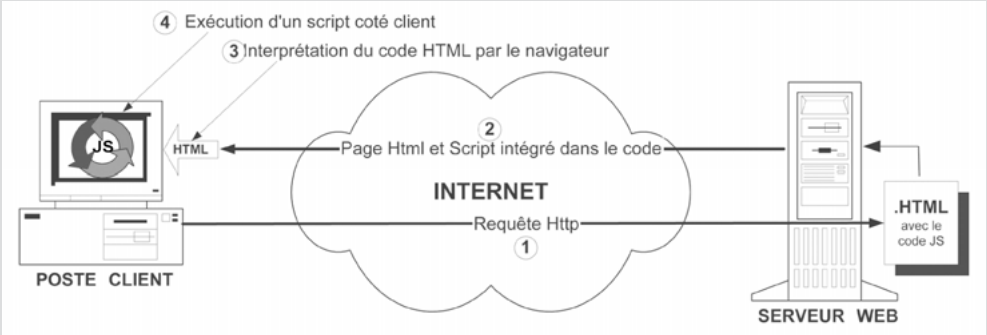


Figure 1-2

Utilisation d'un script côté client avec JavaScript : il existe une dépendance relative au navigateur mais l'interactivité est rapide.

En revanche, les programmes JavaScript souffrent de problèmes de compatibilité avec la configuration du client sur lequel ils s'exécutent et peuvent se comporter différemment selon le type d'ordinateur et la version du navigateur. Par exemple, un script en JavaScript peut parfaitement fonctionner sur Firefox mais poser des problèmes avec Internet Explorer ou créer des erreurs sous IE 5 alors qu'il fonctionne sous IE 6 ou IE 7. De même, les résultats peuvent varier selon qu'on utilise un PC ou un Mac. Tout cela impose au concepteur multimédia d'effectuer des tests importants s'il désire que sa page interactive fonctionne sur toutes les plates-formes et dans toutes les configurations.

☞

Les sites interactifs côté client *(suite)*

D'autres problèmes liés à la sécurité des données constituent aussi un frein à l'usage des technologies client. En effet, le code source des programmes étant intégré dans la page renvoyée par le serveur au client, il devient facile pour un développeur mal intentionné d'altérer le fonctionnement des scripts en consultant simplement le code source de la page HTML.

Enfin, la possibilité donnée à l'internaute d'invalider le fonctionnement de JavaScript sur son navigateur contraint le développeur à prévoir des solutions alternatives pour que ses applications puissent quand même fonctionner en mode dégradé.

JavaScript

On appelle souvent JavaScript, par abus de langage, l'ensemble des deux technologies client les plus utilisées sur le Web : le « JavaScript » développé par Netscape Communications (à ne pas confondre avec Java) et le « Jscript » développé un peu plus tard par Microsoft pour concurrencer la technologie de Netscape. En réalité, ils sont tous les deux conformes (ou censés l'être…) au ECMAScript (European Computer Manufacturers Association) qui est un standard européen, mais aussi international, de ces technologies client.

Quoi qu'il en soit, le fonctionnement des deux technologies est le même. Les instructions JavaScript ou Jscript sont incluses dans le code HTML des pages envoyées par le serveur vers le poste client, puis sont traitées directement par le navigateur grâce à un interpréteur standard implémenté dans le logiciel client.

2000 : le Web orienté serveur

À partir des années 2000, l'évolution croissante des complications rencontrées avec les technologies client a entraîné une migration progressive des applications côté serveur. Motivés par les problèmes de compatibilité et de sécurité liés aux applications côté client, bon nombre de développeurs ont adapté et installé leur programme côté serveur pour mieux satisfaire les internautes (ce qui explique en partie l'extraordinaire développement de langages serveurs comme le PHP). En quelques années la majorité des sites ont subi des refontes structurelles pour s'adapter à une infrastructure Web exploitant principalement des applications côté serveur et cela malgré une organisation du serveur Web plus complexe liée à l'usage de ces technologies serveur.

Cependant, l'utilisation intensive des technologies serveur n'est pas non plus sans inconvénient. En effet, un usage exclusif d'applications serveur (alors que certaines gagneraient à être exécutées côté client) entraîne l'échange, entre le client et le serveur, d'un grand nombre de requêtes qui ont vite fait d'engorger le réseau et de ralentir fortement la réactivité de l'application. De même, à chaque requête, le serveur envoie la page HTML complète avec tout son lot d'informations redondantes, ce qui ralentit fortement l'échange d'informations entraînant des temps d'attente importants pour l'utilisateur.

Les sites interactifs côté serveur

Lorsque l'interactivité est placée côté serveur, le serveur Web doit disposer d'un préprocesseur PHP afin de traiter les scripts PHP intégrés dans la page avant d'envoyer celle-ci au poste client qui en a fait la demande (voir étapes 1 et 2 de la figure 1-3).

Si on le compare avec un script côté client, la réaction d'un script côté serveur à un événement est beaucoup plus lente car elle nécessite l'envoi d'une requête au serveur (voir étape 1 de la figure 1-3), son exécution sur le serveur (étape 2), le retour de la réponse par le réseau Internet (étape 3) et le chargement d'une page HTML complète dans le navigateur (étape 4).

☞

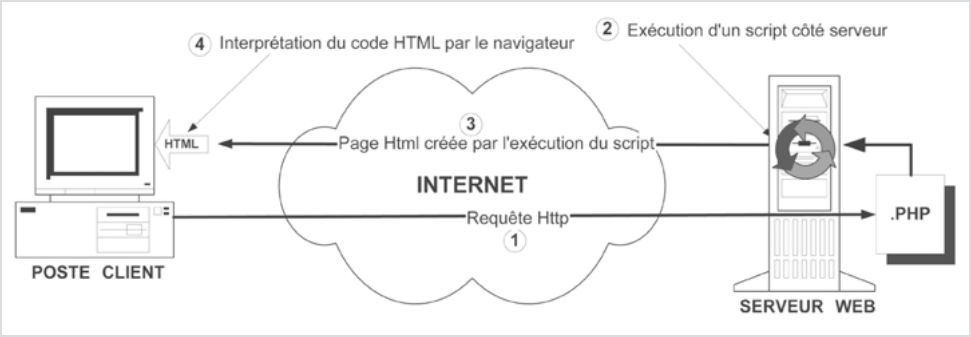


Figure 1-3

Utilisation d'un script côté serveur : il n'y pas de dépendance vis-à-vis du navigateur mais l'interactivité est plus lente.

En revanche, les langages côté serveur sont indépendants de la plate-forme du client ou de la version de son navigateur. En effet, l'interprétation du script est réalisée côté serveur et le code envoyé vers l'ordinateur du client est compatible avec le standard HTML et donc interprété de la même manière par tous.

Parmi les inconvénients des scripts côté serveur, il faut signaler que leur utilisation nécessite la disponibilité d'un serveur adapté. Même si les offres des hébergeurs qui proposent des serveurs intégrant des scripts dynamiques sont désormais très accessibles, il faut en tenir compte lors de votre choix.

Les sites dynamiques

L'exécution du script côté serveur permet de créer une page « à la volée » lors de son exécution par le préprocesseur PHP intégré au serveur. La page ainsi créée contient les mêmes informations qu'une simple page HTML. Elle peut donc être interprétée sans problème par le navigateur côté client (voir figure 1-4).

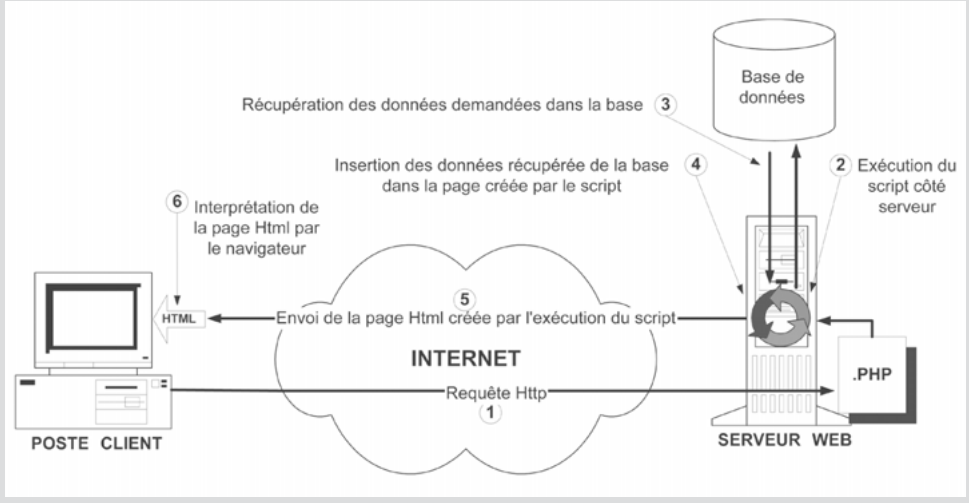


Figure 1-4

Utilisation d'un script côté serveur avec accès à une base de données lors de la création d'une page dynamique.

☞

Les sites dynamiques *(suite)*

Lors de la création de cette page, les scripts (PHP par exemple) intégrés au fichier dynamique sont exécutés et, si nécessaire, établissent une connexion à un serveur de données. Avec ce processus, la page dynamique devient un modèle de présentation des informations. Ce modèle pouvant être personnalisé par des contenus différents selon la requête du client.

Il n'est donc plus nécessaire, par exemple, de créer une page spécifique pour présenter chaque produit d'un catalogue : une seule page dynamique peut être utilisée. Il suffit de lui indiquer l'identifiant du produit demandé grâce à une variable qui lui est transmise en même temps que son appel ; la page renvoyée au client contient alors toutes les informations et photos relatives au produit concerné. L'arborescence du site est simplifiée puisque cette page dynamique remplace les nombreuses pages statiques correspondant à chaque produit.

MySQL

MySQL (My Structured Query Language) est un serveur de base de données relationnelles SQL souvent couplé avec le langage de script serveur PHP.

PHP

PHP (initialement Personal Home Page puis Hypertext Preprocessor) est un langage de script libre utilisé principalement comme langage de programmation Web côté serveur.

2005 : le compromis client-serveur tant attendu !

Heureusement, les navigateurs les plus courants se sont améliorés en attachant progressivement plus d'importance aux standards (même s'il reste encore des divergences entre certains d'entre eux…), diminuant ainsi les problèmes de compatibilité liés à l'usage de technologies côté client. De même, la valeur ajoutée résultant des applications client sans cesse plus puissantes a compensé rapidement les craintes des utilisateurs à leur égard. Le fait que bien des sites populaires exploitent désormais le JavaScript a entraîné progressivement une disparition des utilisateurs qui désactivaient les technologies client dans leur navigateur.

Ces évolutions ont eu une incidence bénéfique sur les ventilations des applications et ont permis un retour à un juste équilibre des tâches entre le client et le serveur. Désormais, l'usage du JavaScript, du DOM et des CSS est entré dans la normalité et ces technologies sont d'ailleurs fortement recommandées pour assurer l'accessibilité du site aux personnes handicapées.

Maintenant, les applications peuvent être équitablement réparties entre le client et le serveur favorisant ainsi une meilleure réactivité des systèmes même si certaines tâches comme la conservation des données (la liaison avec les bases de données est toujours réalisée côté serveur) ou la gestion de l'authentification restent encore le privilège des technologies serveur.

DOM

Le DOM (Document Object Model) est une technologie qui permet la modélisation des éléments d'une page XHTML sous forme d'une hiérarchie normalisée indépendante de tout type de langage. Couplé à la technologie JavaScript, il est ainsi possible de modifier la structure d'une page Web à la volée. Le DOM fait partie des technologies exploitées par Ajax pour interagir sur le contenu ou la forme d'une page XHTML.

CSS

Le CSS (Cascading Style Sheets) permet de décrire précisément la mise en forme d'une page HTML, XHTML ou XML. Il est ainsi possible de séparer clairement le contenu d'une page Web (matérialisé par les balises XHTML) et sa forme (décrite par la feuille de style CSS).

Les tendances du Web 2.0 de demain.

Puisque la situation semble s'être stabilisée depuis l'équilibre des tâches entre client et serveur, nous pourrions nous demander en quoi Ajax pourrait améliorer encore l'usage des applications Web sur Internet. Si la ventilation des scripts client-serveur s'est révélée fructueuse pour une meilleure réactivité des applications, le problème de l'inertie des requêtes HTTP n'en reste pas moins présent. De plus, l'envoi d'une requête bloque les actions de l'internaute jusqu'à la réponse du serveur et le fait que la réponse du serveur soit constituée du code complet de la page et que la page en cours soit obligatoirement rafraîchie sont des inconvénients dont on aimerait bien se passer.

La technologie Ajax peut alors apporter une solution à cette problématique. En effet, partant du constat que l'exécution des programmes côté client est devenue maintenant plus fiable sur la majorité des navigateurs et que l'objet XMLHttpRequest est désormais implémenté sur la plupart d'entre eux, il devient possible de mettre en œuvre des applications Ajax gérant l'envoi de la requête au serveur d'une manière asynchrone. Le mode asynchrone est possible grâce à une utilisation pertinente de l'objet XMLHttpRequest qui permet d'envoyer une requête serveur en tâche de fond sans pour autant bloquer l'activité de l'application client pendant l'attente de la réponse du serveur. À la réception de la réponse du serveur, la donnée (et non toute la page HTML comme c'était le cas auparavant) s'insérera automatiquement dans la page Web en cours sans que s'opère un rafraîchissement complet de la page (voir figure 1-5).

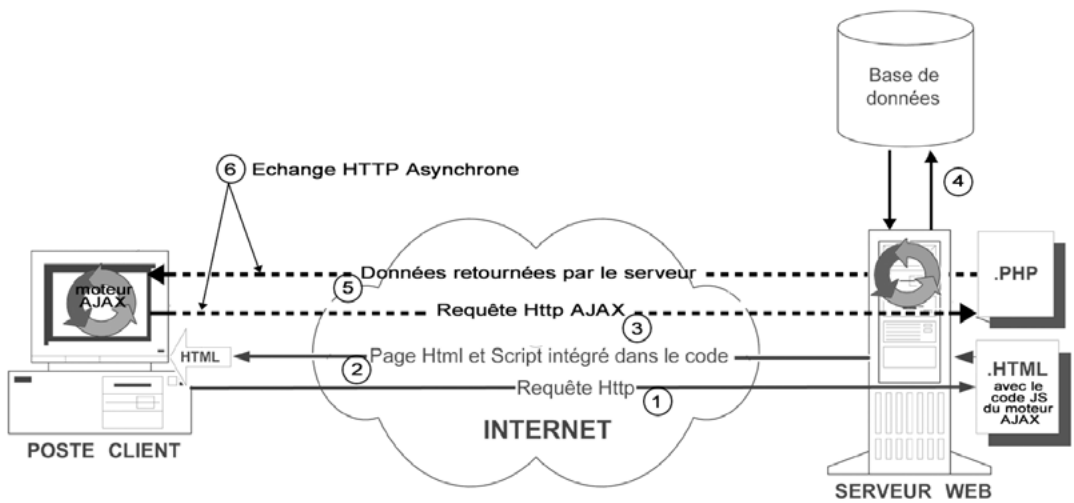


Figure 1-5

Utilisation d'un moteur AJAX chargé côté client afin d'interagir d'une manière asynchrone avec un programme serveur en PHP.

Avec ce type de processus, les applications Web deviennent plus souples à utiliser, car elles ne bloquent plus l'utilisateur pendant le temps d'attente du serveur. De même, elles permettent une meilleure réactivité de l'application car seule la donnée demandée sera retournée au client. Enfin, le fait qu'il n'y ait plus de chargement d'une nouvelle page à chaque requête, améliore considérablement le confort de l'internaute. Tous ces atouts permettent désormais de proposer aux internautes des applications riches très réactives qui préfigurent une nouvelle ère, celle du Web 2.0.

Ajax

Ajax (Asynchronous JavaScript and Xml) désigne une combinaison de technologies (XHTML, DOM, CSS, XML, JavaScript et plus particulièrement son objet XMLHttpRequest) permettant de mettre en œuvre sur le Web des applications interactives et riches comparables aux logiciels disponibles jusqu'alors sur les ordinateurs de bureau.

XMLHttpRequest

XMLHttpRequest est une classe JavaScript disposant de propriétés et de méthodes permettant de récupérer des données sur le serveur d'une manière asynchrone.

2

Ajax, un acteur du Web 2.0

Les fondements du Web 2.0

Avant de parler plus particulièrement d'Ajax, rappelons quelques notions de base sur les sites Web 2.0, ou plutôt les sites de « type Web 2.0 », car l'appellation Web 2.0 correspond plus à un concept qu'à la structure matérielle ou logicielle spécifique d'un site.

Application Internet riche (RIA)

Les sites Web 2.0 se caractérisent par leurs fonctionnalités, leur ergonomie et leur réactivité qui s'apparentent davantage à des applications d'ordinateurs de bureau qu'aux applications traditionnelles du Web.

Quand on souhaite mettre en œuvre un site de type Web 2.0 avec les avantages que nous avons énumérés précédemment, il faut faire appel à des applications Internet riches (RIA). Contrairement aux applications Web traditionnelles pour lesquelles le traitement des données est principalement réalisé côté serveur (le client ne faisant qu'en assurer la présentation), les applications Internet riches déportent les traitements sur le client (navigateur) afin de mettre à la disposition de l'utilisateur des fonctionnalités avancées et très réactives.

RIA

Les RIA (Rich Internet Application) sont des applications Web qui permettent de disposer en ligne des mêmes services que sur une application habituellement installée sur un ordinateur de bureau.
Le webmail est bon exemple de RIA simple car il permet de consulter ses messages électroniques depuis un navigateur, alors que cela nécessitait auparavant l'utilisation d'un gestionnaire de messagerie préalablement installé sur son ordinateur (Outlook par exemple).

Ajax, l'application Internet riche légère

Pour réaliser ces applications d'un genre nouveau, différentes technologies peuvent être mises en œuvre. Parmi ces technologies, il en existe une qui se distingue particulièrement

et qui fait beaucoup parler d'elle : Ajax, mais d'autres solutions permettent aussi de créer des clients riches Internet tel que Flash couplé avec Flex ou encore les applications Java déployées sur Internet à l'aide de Java Web Start.

Cependant, Ajax est souvent préféré par les développeurs car, contrairement aux autres RIA, il a l'énorme avantage de ne pas nécessiter la présence d'un plug-in puisqu'il exploite des technologies intégrées par défaut dans tous les navigateurs récents (CSS, DOM, JavaScript et son objet XMLHttpRequest, XML).

À noter que, parmi les RIA, certaines comme les applications Java sont considérées comme des clients lourds du fait de l'infrastructure logicielle nécessaire au fonctionnement de l'application à ajouter au navigateur. D'autres, comme Flash, nécessitent un plug-in moins volumineux et qui, de surcroît, est souvent pré-installé dans la majorité des navigateurs. En comparaison, Ajax ne nécessite aucun plug-in pour fonctionner, nous pouvons donc le considérer comme une application riche Internet légère.

Plug-in

Le plug-in est un programme devant être installé préalablement sur le navigateur pour qu'une application puisse fonctionner.

Ajax, dans la droite ligne du Web 2.0

Ajax se place dans la droite ligne du Web 2.0 car il permet aux internautes de disposer d'interfaces riches semblables à celles des logiciels de bureau.

En effet, les applications Ajax permettent de disposer de fonctionnalités avancées mais aussi d'améliorer l'interactivité et l'ergonomie des interfaces Web. Concrètement, l'internaute peut déclencher des traitements modifiant à la volée la structure de la page ou générant des effets graphiques avancés (réduction progressive de la taille d'une image sur un survol de la souris, disparition ou apparition fluide et progressive d'une image, déplacement instantané d'un élément de la page par un simple glisser-déplacer de la souris…). Les applications Ajax permettent aussi de faire abstraction des problèmes d'hétérogénéité du navigateur utilisé (grâce à l'utilisation de bibliothèques externes) afin d'assurer le même rendu graphique sur toutes les plates-formes mais, surtout, les mêmes fonctionnalités avancées qui font la richesse de ces nouveaux types d'interface.

Les applications Ajax se caractérisent principalement par un nouveau mode d'échange de données entre le navigateur et le serveur Web. Contrairement aux sites traditionnels pour lesquels l'envoi d'une requête vers le serveur impose au navigateur d'attendre sa réponse, le privant du même coup de tout type d'activité pendant ce délai (transfert synchrone), les applications Ajax permettent d'émettre une requête et d'en réceptionner la réponse d'une manière différée (transfert asynchrone) sans interrompre l'activité de l'utilisateur. En plus de cet avantage — qui est loin d'être négligeable — la réponse du serveur ne contient que les données sollicitées par la requête et non toute la page HTML comme c'est le cas lors d'une réponse classique. Dès son arrivée sur le poste client, un processus est déclenché qui introduit discrètement les nouvelles données dans la page active, évitant ainsi le rechargement de la page à l'origine des « trous blancs » (temps d'attente) désagréables que l'on connaît. Avec ce type de transfert, le trafic s'en trouve réduit et la réactivité de l'application renforcée.

La genèse d'Ajax

En décembre 2004, Google lance en version bêta un nouveau service en ligne : « Google Suggest » (voir figure 2-1). Ce moteur de recherche intelligent suggère une liste de dix mots en rapport avec les premières lettres saisies dans le champ de recherche. À chaque ajout d'une lettre, les suggestions du menu déroulant sont actualisées dynamiquement. Ces dernières indiquent en plus le nombre de résultats correspondant à chaque suggestion, guidant ainsi l'internaute dans son choix. Le concept des applications interactives de la nouvelle génération du Web était né.

Figure 2-1

Exemple d'une recherche d'informations sur Ajax avec Google Suggest

Quelques mois plus tard, en février 2005, le nom « Ajax » fait sa première apparition sur Internet dans un article de Jesse James Garret de l'agence Adaptative Path (voir figure 2-2). Son article, *Ajax: A New Approach to Web Applications*, expose ses expérimentations concernant l'utilisation de JavaScript pour créer des interfaces innovantes qui se distinguent des applications actuelles par une grande interactivité et dans lesquelles le chargement d'une page entière n'est plus nécessaire. Ce nouveau type d'interface permet de mettre en œuvre des fonctionnalités avancées qui s'apparentent à celles des ordinateurs de bureau. Il définit alors Ajax comme étant le raccourci de *Asynchronous JavaScript And Xml*.

Par la suite, courant 2005, Google lance de nouvelles applications à succès du même type comme Gmail, Google Map ou encore dernièrement Google Calendar. Mais Google n'était pas le seul à s'intéresser à Ajax. Yahoo! notamment a retenu en 2005 cette technologie innovante lors de la refonte de son site d'informations Yahoo! News. Depuis, de nombreux sites exploitent Ajax pour rendre leur interface plus interactive et cela laisse présager que la technologie Ajax n'en est qu'à ses balbutiements.

Figure 2-2

Article de Jesse James Garret en février 2005 qui attribue pour la première fois le terme « Ajax » à un nouveau type d'application interactive.

À quoi sert Ajax ?

Pour illustrer l'utilisation d'Ajax, rien de mieux que quelques exemples concrets. Aussi, nous vous proposons ci-dessous une liste non exhaustive (loin de là !) de quelques emplois courants d'Ajax dans le Web 2.0 d'aujourd'hui.

Actualisation d'information en tâche de fond

L'avantage d'une requête asynchrone est de pouvoir récupérer des données sans interrompre le travail de l'internaute. Il est alors très simple de mettre en place des systèmes d'actualisation d'une information spécifique d'une page HTML, déclenchés d'une manière chronique ou par un gestionnaire d'événements JavaScript. On peut alors imaginer, par exemple, qu'une zone de page Web affichant les derniers résultats d'un match de tennis ou des élections en cours puisse actualiser ses informations à intervalles réguliers sans aucun rechargement de la page et sans que l'internaute n'ait besoin de solliciter la mise à jour des résultats (voir l'atelier 10-5 qui illustre une application de ce type par un

exemple pratique). Pendant ce temps celui-ci peut utiliser les autres fonctionnalités de la page Web sans aucune perturbation, car l'application asynchrone ne bloque pas l'utilisation du navigateur pendant le traitement du serveur et actualise uniquement le résultat de la zone concernée.

Complétion automatique

Depuis le lancement de Google Suggest en 2005 (voir figure 2-1) de nombreux sites ont intégré des systèmes d'auto-complétion dans leur formulaire pour assister les internautes dans leur choix. Le système de complétion automatique permet d'afficher dans une liste déroulante des suggestions pertinentes par rapport au début de la saisie de l'internaute (voir l'atelier 15-2 qui illustre cette application par un exemple pratique). Le navigateur envoie pour cela au serveur le début de la saisie de l'utilisateur, le serveur réceptionne l'information, la traite en recherchant les réponses possibles commençant par la chaîne de caractères réceptionnée et renvoie ses suggestions au navigateur qui les affichera par exemple dans une liste déroulante. Ces suggestions peuvent être actualisées à chaque nouveau caractère saisi, mais il est souvent plus judicieux de déclencher l'actualisation de la liste selon un intervalle de temps défini afin de ne pas trop surcharger le serveur (voir par exemple le site de Google Suggest de la figure 2-1).

Contrôle en temps réel des données d'un formulaire

Dans un formulaire traditionnel, le contrôle des champs peut être réalisé par des fonctions JavaScript au fil de la saisie si l'on désire simplement s'assurer de la présence ou de l'adéquation du contenu (le contrôle de la bonne syntaxe d'un e-mail par exemple). Par contre, pour des vérifications plus poussées nécessitant de comparer le contenu d'un champ avec des informations issues d'une base de données (comme la vérification de l'existence d'un pseudonyme lors de la création d'un compte utilisateur), on est alors contraint de réaliser ces vérifications côté serveur après l'envoi du formulaire.

Ajax peut alors être utilisé judicieusement pour réaliser des tests lors de la saisie et donc avant la soumission du formulaire. Il suffit pour cela d'envoyer une requête au serveur dès que le contenu du champ est connu (voir l'atelier 13-1 qui illustre cette application par un exemple pratique). Le serveur s'occupera du traitement de ce contenu en le comparant avec les informations de la base de données pendant que l'utilisateur continuera de renseigner les autres champs. Si le contrôle s'avère négatif, le serveur renverra un message d'erreur au navigateur qui l'avertira et lui permettra de modifier sa saisie avant la soumission du formulaire.

Navigation dynamique

De nombreux menus de navigation ou onglets de pagination exploitent désormais la technologie Ajax afin d'éviter le rechargement de la page à chaque nouvelle sélection (voir l'atelier 15-1 qui illustre cette application par un exemple pratique). Le résultat est en général assez agréable à utiliser car cette technologie permet une transition fluide et continue d'une page à l'autre, mais il est souvent judicieux de la coupler avec des systèmes alternatifs comme les cadres cachés afin de conserver l'utilisation des boutons Suivant et Précédent et l'historique de navigation.

Lecture d'un flux RSS

Les flux RSS permettent de diffuser les mises à jour des sites d'information sur d'autres sites ou applications tiers (page personnelle comme Netvibes, blog, gestionnaire de messagerie…). L'utilisateur peut s'abonner aux flux d'informations de son choix et afficher ainsi des nouvelles qui s'actualiseront automatiquement sur son site (voir l'atelier 12-6 qui illustre cette application par un exemple pratique).

Ces flux sont contenus dans un document au format XML, structuré selon des items prédéfinis (titre, résumé, date…). Ajax permet de récupérer ce type de flux et de l'afficher dans un navigateur après l'avoir converti au format HTML sans recharger la page (voir la présentation du site de Netvibes ci-dessous en guise d'exemple).

Sauvegarde de documents éditables

L'édition de documents en ligne est de plus en plus courante sur les sites communautaires. Elle permet de concevoir des documents riches (textes, images…) puis de les sauvegarder directement depuis un navigateur sans avoir recours à des formulaires de téléchargement (voir l'atelier 13-2 qui présente une application de sauvegarde d'informations).

Ces systèmes caractérisent très bien les applications de type Web 2.0 qui permettent à l'internaute de disposer de services avancés proches de ceux proposés jusqu'à présent par des logiciels de bureau.

Personnalisation d'interface Web

Si la personnalisation d'une interface Web (disposition et sélection des blocs à afficher dans sa page Web personnelle, ou encore le choix de modèles de mise en page incluant la couleur, la police et bien d'autres paramètres) peut être réalisée par les technologies du DHTML (JavaScript, DOM, CSS), leur mémorisation est souvent effectuée par Ajax et notamment grâce à l'objet XMLHttpRequest qui se chargera d'envoyer au serveur les nouveaux paramètres de votre page personnalisée dès qu'une modification de celle-ci sera détectée.

Le site Netvibes illustre très bien l'utilisation de cette fonctionnalité d'Ajax (entre autres) pour permettre aux internautes d'aménager librement leur page personnelle.

Widget

Les widgets sont des petites applications Web qui ont l'énorme avantage de ne pas nécessiter la présence d'un navigateur pour fonctionner (voir l'atelier 15-3 qui illustre la mise en œuvre d'un widget de calendrier). Ils peuvent donc être placés sur le bureau de votre ordinateur et être déplacés comme bon vous semble (à l'origine, le concept des widgets vient d'Apple qui les avait déjà intégrés dans le système d'exploitation Mac OS X version 10.4).

De nombreux widgets exploitent Ajax pour récupérer des données sur un serveur d'informations soit d'une manière chronique, soit à la demande de l'utilisateur. Ainsi, si vous le désirez, vous pouvez placer un widget sur le bureau de votre ordinateur pour afficher en permanence les prévisions météorologiques ou les fluctuation d'une valeur boursière.

Le nouveau service Talk de Google est basé sur un widget Ajax. Il permet l'utilisation d'une messagerie instantanée sans nécessiter la présence d'un navigateur. Pour l'installer

sur le bureau de votre PC, il suffit de télécharger un petit logiciel gratuit proposé en téléchargement sur Google.

Chargement progressif d'information

Le chargement de données volumineuses est un problème qui freine fortement la réactivité d'un site. À l'ère du haut débit, il est maintenant difficilement concevable d'attendre plusieurs secondes pour que l'effet d'un événement déclenché par l'utilisateur puisse s'afficher sur l'interface du site. Cependant, certaines applications doivent interagir rapidement pour afficher de nouvelles images qu'il n'est pas toujours concevable de précharger (comme les applications de cartographie par exemple). Il faut alors faire appel à des méthodes prédictives couplées à des technologies client comme Ajax pour trouver une solution à ce dilemme.

L'exemple de la cartographie Google Maps illustre bien cette utilisation en permettant à l'application de conserver une bonne réactivité lors du glisser-déplacer de la carte par exemple.

Moteur de recherche sans rechargement de la page

La plupart des moteurs de recherche nécessitent un rechargement complet de la liste des résultats à chaque modification des critères de recherche. Il est cependant possible d'utiliser la technologie Ajax pour pallier ce désagrément (voir par exemple le moteur de recherche *http://www.rollyo.com* ou encore le moteur d'Amazon *http://www.a9.com*).

Qui utilise Ajax ?

Actuellement de nombreux sites de renom utilisent déjà des applications Ajax pour améliorer leur interface en augmentant l'interactivité avec les internautes et en apportant de nouvelles fonctionnalités, plus proches des applications d'ordinateur de bureau que des sites Web traditionnels. Nous vous proposons ci-dessous de faire un rapide tour d'horizon de ces sites précurseurs du Web 2.0.

Google suggest

Google suggest (voir figure 2-1) a été le premier site à exploiter le modèle Ajax pour améliorer l'interface de son outil de recherche que tout le monde connaît. Lorsque l'internaute renseigne le champ de recherche avec un mot-clé, une requête est envoyée au serveur à chaque nouveau caractère saisi afin d'afficher une liste de suggestions susceptibles de correspondre à votre recherche.

Google suggest utilise l'objet XMLHttpRequest et le format JSON (et non le XML) pour gérer les échanges de données avec le serveur.

http://www.google.com/webhp?complete=1&hl=en

Gmail

Gmail est un webmail gratuit mis à la disposition de tous les internautes par Google. À l'instar de son outil de recherche, Google doit la réussite de son webmail à la simplicité et la grande réactivité de son interface.

Gmail utilise l'objet XMLHttpRequest conjointement avec une structure de cadres cachés afin d'éviter certains défauts inhérents à l'utilisation exclusive de cet objet (boutons Précédent et Suivant inactifs). Pour la gestion du transfert des données, le format JSON a été choisi afin d'améliorer la rapidité des échanges entre le serveur et le navigateur.

http://mail.google.com

Google Maps

Google Maps est le service de cartographie proposé par Google. Le site Google Maps utilise des applications Ajax pour améliorer l'interactivité avec l'internaute en lui offrant une interface très intuitive. L'utilisateur peut par exemple faire glisser la carte dans toutes les directions de façon très fluide grâce au préchargement des images ou encore utiliser la molette de sa souris pour zoomer sans avoir à recharger la carte.

Ici aussi, Google à opté pour l'utilisation de cadres cachés. En ce qui concerne le transfert des informations retournées par le serveur, c'est le format XML qui a été retenu, celui-ci permettant d'opérer très facilement des transformations de son contenu au moyen de fonctions XSLT. La gestion du cache du navigateur est aussi exploitée pour obtenir une bonne réactivité des déplacements ou des zooms de carte permettant ainsi l'affichage rapide des multiples petites images qui constituent la carte.

http://maps.google.com

Yahoo! News

En 2005, Yahoo! News a aussi opté pour Ajax afin d'améliorer l'interactivité de son interface. L'internaute pouvait alors voir le résumé et la photo d'un article en passant simplement sa souris sur son titre. Dès que l'événement était détecté, une requête `XMLHttpRequest` était envoyée au serveur qui retournait rapidement les compléments de l'article au navigateur.

Cette application étant intégrée en complément d'une structure de page traditionnelle, elle permettait de disposer de fonctionnalités avancées tout en conservant un mode dégradé pour les internautes ne disposant pas d'un navigateur compatible avec l'application Ajax.

http://news.yahoo.com

A9.com

En 2005, Amazon a créé un méta-moteur qui permet de rechercher dans plusieurs types de médias en une seule requête (livre, vidéo, définition, article…). Pour cela, il exploite plusieurs bases de données telles que Wikipédia, Answers.com et évidemment celle de son site de commerce en ligne, Amazon.com. Les résultats sont présentés dans des colonnes différentes en fonction de leur origine. Si vous désirez modifier les critères de recherche, les boîtes contenant les résultats précédents sont alors redimensionnées automatiquement sans qu'un rechargement de la page ne soit nécessaire.

Pour les reconditionnements des boîtes des résultats, A9.com exploite des librairies DHTML. Les transferts de données entre le serveur et le navigateur sont eux réalisés grâce à la classe `XMLHttpRequest` combinée avec une structure de cadres cachés.

http://www.a9.com

Google Calendar

En avril 2006, Google complète ses services en mettant à la disposition des internautes un système d'agenda en ligne exploitant pleinement tous les artifices de la technologie Ajax. L'utilisateur peut ajouter très facilement ses rendez-vous par un simple clic, il peut aussi les partager facilement avec d'autres utilisateurs de Google Calendar ou des personnes de son choix. Les rendez-vous peuvent être déplacés ou agrandis très rapidement à l'aide de la souris. L'affichage peut être configuré sur un jour, une semaine ou un mois, et cela sans aucun rechargement de la page.

http://www.google.com/calendar

Netvibes

Netvibes est un portail personnalisable qui permet de créer et aménager sa page personnelle très simplement. L'internaute peut ainsi ajouter, supprimer, modifier ou déplacer les différentes boîtes qui composent sa page. Il pourra notamment configurer des flux RSS provenant de différents fournisseurs d'information pour en afficher le contenu dans la boîte de son choix.

Ce site illustre parfaitement l'utilisation que l'on peut faire d'Ajax et du DHTML pour mettre à la disposition de l'internaute des fonctionnalités riches du Web 2.0 tout en préservant l'ergonomie de la page Web.

http://www.netvibes.com

Google Talk

L'application Google Talk vous permet de disposer d'une messagerie instantanée et d'un service de voix IP en permanence, même si votre navigateur est fermé. L'utilisation de cette application nécessite néanmoins l'installation d'un logiciel sur votre ordinateur.

Google Talk est un widget Ajax basé sur Jabber (système standard de messagerie instantanée utilisant le protocole XMPP, basé sur le XML, développé par Google) qui permet de discuter avec des millions d'internautes.

http://www.google.com/talk

Wikipédia

Wikipédia est une encyclopédie collaborative en ligne très connue, qui regroupe des millions d'articles sur des thèmes divers et variés. Chaque internaute peut y contribuer en rédigeant un article selon sa spécialité.

Ce site exploite de nombreuses applications basées sur la technologie Ajax, comme les onglets dynamiques (le contenu de la zone d'information est modifié sans rechargement de la page) ou encore la gestion de ses ressources qui permet à chaque internaute d'éditer en ligne des documents riches (textes, images…) sans employer de formulaire de téléchargement.

http://fr.wikipedia.org

3

Comment fonctionne Ajax ?

Ajax, un amalgame de technologies

Des ingrédients déjà opérationnels

Contrairement à ce que l'on pourrait croire, Ajax n'est pas une technologie spécifique et innovante mais une conjonction de plusieurs technologies anciennes. Ainsi, les applications Ajax utilisent en général tout ou partie des technologies suivantes :

- Les feuilles de styles CSS qui permettent d'appliquer une mise forme au contenu d'une page XHTML.
- Le DOM qui représente la hiérarchie des élément d'une page XHTML.
- L'objet XMLHttpRequest de JavaScript qui permet d'assurer des transferts asynchrones (ou quelquefois synchrones) entre le client et le serveur.
- Les formats de données XML ou JSON utilisés pour les transferts entre le serveur et le client.
- Le langage de script client JavaScript qui permet l'interaction de ces différentes technologies.

L'intérêt pour Ajax d'utiliser ces différentes technologies est qu'elles sont déjà intégrées dans la plupart des navigateurs actuels. Elles sont donc immédiatement exploitables – même si quelques différences d'implémentation subsistent d'un navigateur à l'autre.

Ceci représente une véritable aubaine pour les développeurs lorsqu'on connaît les atouts d'Ajax ; et on comprend mieux pourquoi toujours plus de développeurs se rallient à cette technologie.

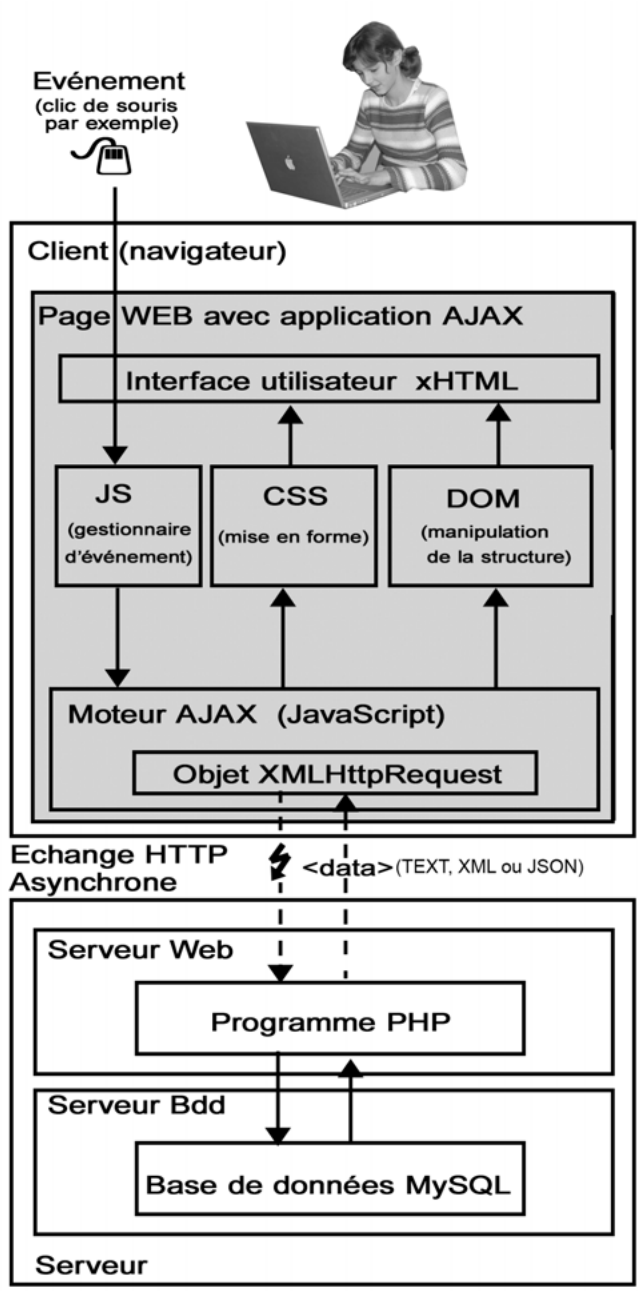
JavaScript, le ciment des fondations d'Ajax

Pour que ces différentes technologies sous-jacentes puissent être exploitées, il faut disposer d'un langage de script capable de les manipuler. Évidemment, dans ce contexte client, JavaScript est la technologie idéale pour remplir cette mission et faire interagir toutes ces technologies entre elles. Ainsi, dans chaque application Ajax, nous retrouverons un

programme JavaScript qui constituera le « moteur » du système, orchestrant à la fois les transferts de données avec l'aide de l'objet XMLHttpRequest et l'exploitation des réponses du serveur en agissant sur les CSS (pour modifier la mise en forme de la page XHTML) et sur le DOM (pour modifier le contenu ou la structure de la page XHTML) (voir figure 3-1).

Figure 3-1

Organisation des principaux composants d'Ajax



En ce qui concerne les données échangées, plusieurs formats peuvent être utilisés selon l'organisation et la complexité des flux d'informations. Les applications les plus simples

pourront se contenter de données au format texte (simples couples variable/valeur) alors que les systèmes plus complexes devront choisir de structurer leurs données en XML (le DOM assurant ensuite l'insertion des données XML dans la page XHTML) ou encore dans un format issu de la structure des objets JavaScript, le JSON. À noter que la plupart des requêtes envoyées vers le serveur utilisent le format texte (les couples variable/valeur suffisent dans la majorité des cas), mais sachez qu'elles peuvent éventuellement aussi exploiter les formats XML ou JSON, de la même manière que les résultats retournés par le serveur au navigateur.

Comparatif avec les applications Web traditionnelles

Pour bien comprendre le fonctionnement et connaître les avantages d'un nouveau système, une bonne méthode consiste à le comparer avec l'existant que l'on connaît déjà. Dans cette partie, nous allons utiliser cette méthode en comparant le fonctionnement d'une application Ajax avec celui d'un site Web statique et celui d'un site Web dynamique.

Fonctionnement d'une application Web statique

Avec un site Web statique, la seule interactivité dont dispose l'internaute est de pouvoir passer d'une page HTML à l'autre par un simple clic sur les liens hypertextes présents sur une page. À chaque fois que l'internaute clique sur un lien, une requête HTTP est envoyée, établissant du même coup une communication avec le serveur. Cette communication est de type synchrone, c'est-à-dire que dès l'émission de la requête, la communication reste en place jusqu'à la réception de la réponse du serveur. Pendant le temps de traitement de la requête, le navigateur reste figé, bloquant ainsi toute action possible de l'internaute.

À chaque requête, le serveur retournera une réponse sous la forme d'une page HTML complète. S'il s'agit d'une simple requête, suite à la saisie par l'internaute de l'URL spécifique d'une page dans la barre d'adresse du navigateur ou, plus couramment, lorsque l'internaute clique sur un lien hypertexte, le serveur se contentera de renvoyer la page HTML demandée, ce qui clôturera le traitement côté serveur et débloquera ainsi le navigateur.

Fonctionnement d'une application Web dynamique

Nous avons vu précédemment le traitement d'une simple requête par le serveur mais d'autre cas peuvent se produire, notamment lors de l'envoi d'un formulaire. Dans ce cas, la requête est constituée d'une ligne de requête (précisant la méthode utilisée et le protocole HTTP), d'un corps (qui contient les données envoyées au serveur dans le cas d'une requête émise avec la méthode POST) et d'une série d'en-têtes qui définissent les spécificités de la requête (nature du navigateur utilisé, type d'encodage…) qui permettront au serveur de traiter correctement les informations. En général, lors de l'envoi d'un formulaire, le traitement côté serveur est réalisé par une page contenant un programme (en PHP par exemple). Les données réceptionnées pouvant être traitées directement par le programme ou entraîner un échange avec un serveur de base de données afin de les mémoriser ou d'émettre une requête SQL. À l'issue de ce traitement, une nouvelle page HTML sera construite à la volée et renvoyée au navigateur, ce qui clôturera le processus, débloquant le navigateur de la même manière qu'avec un site statique.

Fonctionnement d'une application Ajax

Dans le cas d'une application Ajax, si la page contenant la structure XHTML et ses scripts client (moteur Ajax, gestionnaire d'événement…) est chargée de la même manière que pour un site statique, il n'en est pas de même pour les interactions qui suivent entre le navigateur et le serveur. Le moteur Ajax une fois chargé dans le navigateur restera en attente de l'événement pour lequel il a été programmé. Pour cela, un gestionnaire d'événement JavaScript est configuré pour appeler le moteur dès l'apparition de l'événement concerné. Lors de l'appel du moteur, un objet XMLHttpRequest est instancié puis configuré, une requête asynchrone est ensuite envoyée au serveur. À la réception de celle-ci, le serveur démarrera son traitement et retournera la réponse HTTP correspondante. Cette dernière sera prise en charge par la fonction de rappel du moteur Ajax qui exploitera les données pour les afficher à un endroit précis de l'écran.

Chronogrammes des échanges client-serveur

Une des grandes différences entre une application Web traditionnelle et une application Ajax est liée à l'échange asynchrone de données entre le navigateur et le serveur. Pour vous permettre de bien appréhender la différence entre ces deux applications, nous vous proposons de les comparer maintenant à l'aide de leur chronogramme.

Chronogramme d'une application Web dynamique traditionnelle

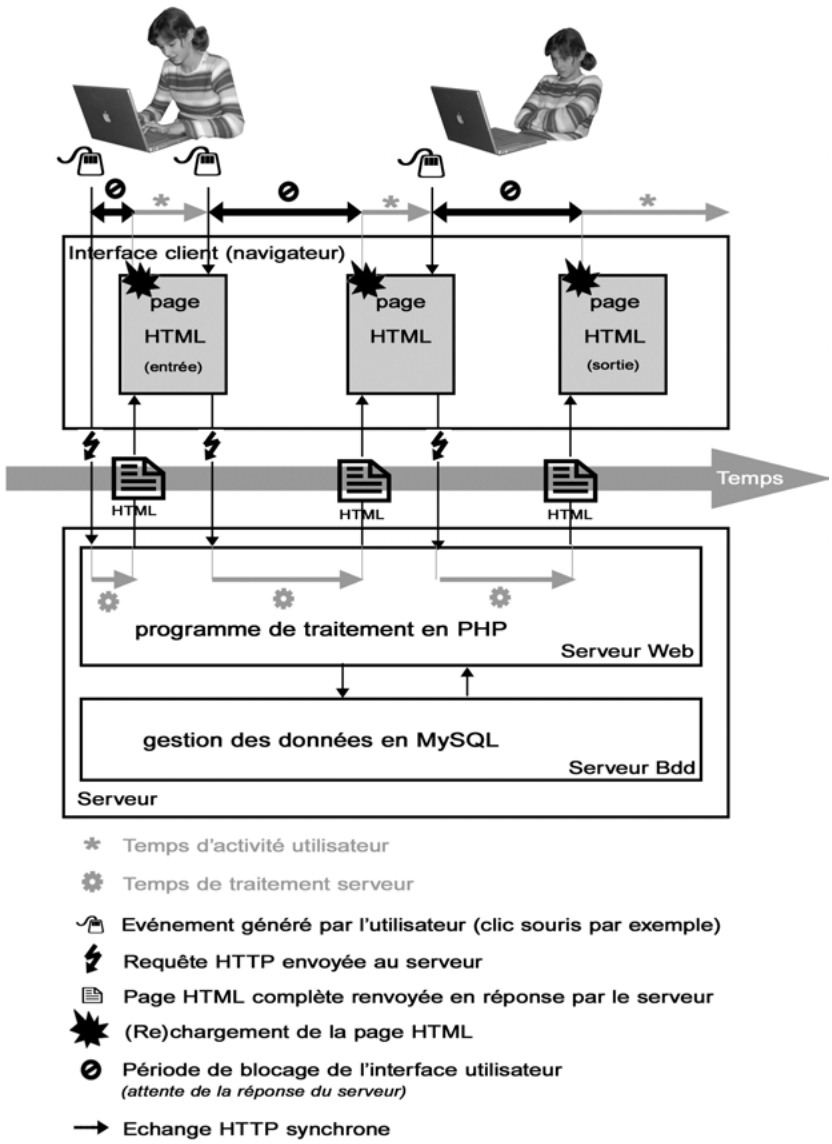
Lorsqu'un utilisateur sollicite le serveur dans une application Web dynamique traditionnelle (en envoyant un formulaire ou en cliquant sur une URL dynamique), il déclenche une requête HTTP dans laquelle sont imbriqués les paramètres de la demande. À partir de ce moment, le navigateur se fige jusqu'à la réception de la réponse HTTP du serveur, interdisant ainsi à l'utilisateur toute action pendant le temps de traitement de la requête. Dès la réception de la requête, le serveur Web analysera les paramètres et traitera la demande selon son programme. Il pourra interroger un serveur de base de données pour recueillir des données complémentaires si nécessaire. Une fois le traitement terminé, une page HTML complète sera construite à la volée, incluant les résultats du traitement après leur mise en forme. Cette page sera alors retournée au navigateur après son intégration dans le corps de la réponse HTTP. À la réception de la réponse HTTP, le navigateur interprétera la page HTML, comme lors de l'appel d'une page Web dans un site statique, et l'affichera à l'écran, entraînant le rechargement complet de la page. À la fin du chargement de la page, le navigateur est débloqué et l'utilisateur reprend la main sur l'application. Il pourra ainsi éventuellement réitérer une nouvelle demande serveur qui suivra le même cycle de traitement que celui que nous venons de décrire (voir figure 3.2).

Chronogramme d'une application Ajax en mode asynchrone

Dans le cas d'une application Ajax en mode asynchrone, le déroulement du traitement est différent. À noter que l'objet XMLHttpRequest peut aussi envoyer des requêtes synchrones, mais dans ce cas le fonctionnement serait semblable à celui d'une application Web dynamique traditionnelle comme celle que nous avons décrite précédemment.

Figure 3-2

Chronogramme des échanges client-serveur d'une application traditionnelle

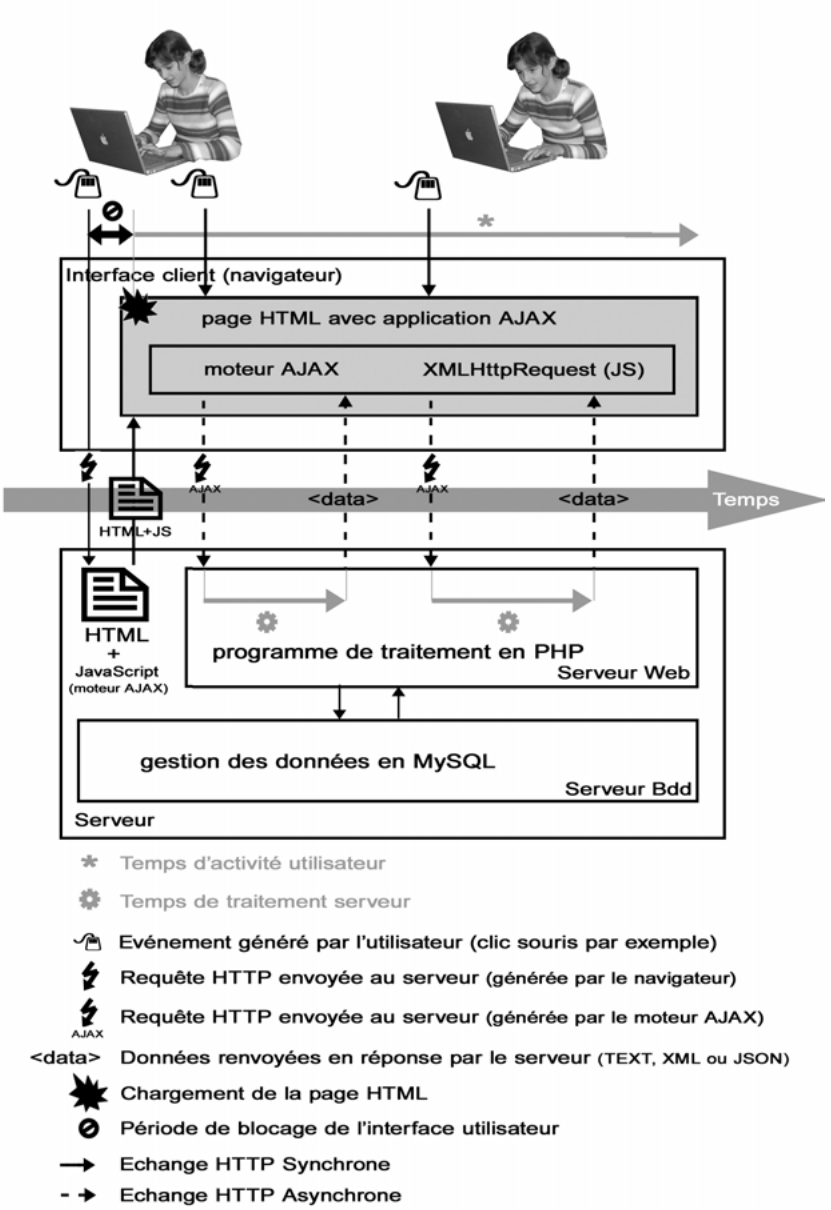


Dans une application Ajax, l'utilisateur doit commencer par appeler la page HTML contenant le moteur Ajax. Une fois la page chargée dans le navigateur, les échanges avec le serveur seront contrôlés par l'application Ajax (voir figure 3-3). L'envoi d'une requête est souvent déclenché par un gestionnaire d'événement JavaScript, mais il peut aussi être généré par un script de temporisation pour actualiser des informations à intervalles réguliers. Quel que soit le mode de déclenchement, le moteur Ajax est appelé par le biais d'une fonction JavaScript. La première action du moteur est la création d'un objet XMLHttpRequest immédiatement suivi de sa configuration (choix de la méthode de transfert GET ou POST, choix du fichier serveur sollicité, activation du mode asynchrone, désignation d'une fonction de rappel, intégration des paramètres…). Une fois l'objet configuré, l'envoi de la requête est déclenché, générant une requête HTTP semblable à celle créée avec une application dynamique traditionnelle. Toutefois, dans le

cas de l'envoi d'une requête Ajax, le navigateur n'est pas bloqué et l'utilisateur peut continuer à utiliser son interface comme bon lui semble ; le transfert est asynchrone. Côté serveur, les paramètres seront analysés et le programme pourra aussi solliciter un serveur de base de données si besoin. Mais, contrairement à une application dynamique traditionnelle, le corps de la réponse HTTP retournée au navigateur ne sera pas composé de la page HTML complète : il contiendra seulement les données réclamées par le client. Lorsque le navigateur reçoit la réponse, une fonction de rappel, programmée lors de l'envoi de la requête, se chargera de récupérer les données placées dans le corps de la réponse HTTP, de les mettre en forme et de les insérer dans une zone particulière de la page Web et cela sans nécessiter le rechargement de la page (voir figure 3-3).

Figure 3-3

Chronogramme des échanges client-serveur d'une application Ajax



Les avantages d'Ajax

Économie de la bande passante

Avec Ajax, il n'est plus nécessaire de renvoyer le contenu entier de la page HTML à chaque requête, car l'objet XMLHttpRequest assure la récupération et l'insertion dans la page en cours des seules données à modifier. Ce système permet d'éliminer le transfert de nombreuses informations redondantes, allégeant ainsi fortement le trafic réseau entre le serveur Web et le client (navigateur).

Empêche le rechargement de la page à chaque requête

Le traitement traditionnel d'une requête HTTP entraîne à chaque retour de la réponse du serveur un rechargement complet de la page en cours. Hormis le désagréable « trou blanc » que cela engendre, ce phénomène allonge le temps de traitement d'une requête aux dépens de la réactivité de l'application.

Évite le blocage de l'application pendant le traitement de la requête

Contrairement au simple échange HTTP d'une application traditionnelle, dans laquelle l'application cliente est bloquée pendant tout le temps d'attente de la réponse du serveur, l'échange XMLHttpRequest asynchrone d'une application Ajax permet à l'internaute de continuer à travailler pendant le temps de traitement de la requête. Cela ouvre des possibilités nouvelles pour le développement Web, permettant ainsi aux développeurs de créer des applications dont le mode de fonctionnement se rapproche de celui des applications disponibles jusqu'alors sur des ordinateurs de bureau.

Augmente la réactivité de l'application

Les données renvoyées par le serveur étant plus légères (le serveur retournant uniquement les données demandées et non la page HTML entière) et le rechargement de la page complète n'ayant plus lieu à chaque requête, cela améliore considérablement la réactivité du système. De plus, le chargement progressif des données couplé à une méthode prédictive permet de disposer de fonctionnalités graphiques avancées (déplacement d'une carte à l'aide de la souris dans une application de cartographie en ligne par exemple) jusqu'alors réservées aux logiciels autonomes de bureau.

Améliore l'ergonomie de l'interface

Une interface Ajax peut être composée de multiples zones ayant une gestion du contenu indépendante l'une de l'autre. Chaque zone pouvant déclencher ses propres requêtes, il est désormais possible d'avoir une mise à jour ciblée des contenus. Ainsi, grâce aux technologies DHTML associées à Ajax, l'utilisateur peut aménager librement ses différentes zones par un simple glisser-déposer et améliorer l'ergonomie de son interface Web.

Les inconvénients d'Ajax

Pas de mémorisation des actions dans l'historique

Le principal inconvénient d'une application Ajax est lié au fait que les actions de l'utilisateur ne sont pas mémorisées dans l'historique du navigateur. En effet, les différents

contenus d'une application Ajax s'affichant toujours dans la même page, ils ne peuvent pas être enregistrés dans l'historique du navigateur comme le seraient les différentes pages HTML d'une application Web traditionnelle.

Par voie de conséquence, les boutons Suivant et Précédent ne sont plus utilisables car ils s'appuient sur l'historique du navigateur pour trouver la page suivante ou précédente. Ceci est évidemment très handicapant pour les internautes qui ont l'habitude d'utiliser ces boutons pour naviguer d'une page à l'autre.

Il existe néanmoins des solutions pour remédier à ce problème en couplant l'application Ajax avec un système d'iframe comme le fait Google dans plusieurs de ses applications Ajax mais cela nécessite un traitement supplémentaire qui complexifie le développement.

Problème d'indexation des contenus

Les différents contenus d'une application Ajax s'affichant dans une seule et même page, les moteurs de recherche pourront indexer uniquement le premier contenu par défaut de la page et non tous les contenus proposés par l'application.

D'autre part, le rappel des différents contenus d'une application Ajax par le biais des favoris sera confronté au même problème. Seul le contenu de la première page pourra être mémorisé dans les signets du navigateur.

Dépendance de l'activation de JavaScript sur le navigateur

Les applications Ajax utilisant JavaScript pour interagir entre les différentes technologies exploitées côté client (CSS, DOM, XML…) sont donc dépendantes de l'activation de JavaScript sur le navigateur, au même titre que tous les autres programmes clients utilisant cette technologie.

Même si les internautes qui désactivent JavaScript se raréfient, il faut toutefois prévoir une version dégradée de l'application en prévision des navigateurs qui ne supporteraient pas ce langage de script.

Les cadres cachés, une solution alternative à Ajax

Dans le chapitre précédent, nous avons cité d'autres technologies estampillées Web 2.0 (Flash + Flex, application Java) permettant la mise en œuvre d'une application Internet riche (RIA). Nous avions cependant écarté ces solutions car elles ne pouvaient pas fonctionner sur un navigateur sans l'installation d'un plug-in.

Il existe néanmoins une technique nommée « cadre caché » (frameset HTML ou iframe) utilisée bien avant celle de l'objet XMLHttpRequest qui permet d'établir des communications en arrière plan avec le serveur et qui, comme Ajax, ne nécessite pas l'ajout d'un plug-in.

La technique du cadre caché

Cette technique exploite la structure des jeux de cadres HTML dont l'un d'entre eux est invisible et sert de pont pour établir une communication avec le serveur. Le cadre caché est rendu invisible en configurant sa largeur et sa hauteur à zéro pixel. Avec cette technique,

il est alors possible d'envoyer des requêtes serveur par le biais du cadre caché sans perturber l'écran de l'utilisateur.

Pour illustrer le fonctionnement de cette technique, nous allons détailler le cycle d'une communication complète entre le navigateur et le serveur. Pour commencer, l'utilisateur déclenche une fonction JavaScript depuis le cadre visible. Cette fonction appellera un script serveur dont le retour sera assigné au cadre caché. Le script serveur analyse alors les paramètres communiqués et traite la demande. Il renvoie ensuite en réponse au cadre caché une page HTML complète contenant le résultat dans une balise `<div>`. Dans cette même page HTML se trouve une fonction JavaScript qui sera invoquée dès que la page sera complètement chargée dans le cadre caché (avec le gestionnaire d'événement `window.onload` par exemple). Enfin, lorsque la fonction JavaScript s'exécute dans le cadre caché, elle récupère le résultat inséré préalablement dans la balise `<div>` de la même page et l'affecte à une zone définie du cadre visible. L'utilisateur peut alors voir la réponse apparaître dans la page visible du navigateur et cela tout en continuant d'utiliser l'interface pendant le traitement serveur évitant ainsi que la page ne soit rechargée.

Depuis l'apparition des iframes (introduites dans la version 4.0 du HTML), il est possible d'exploiter la même technique mais sans avoir à utiliser la structure contraignante des framesets. En effet, l'iframe peut être placé dans une page HTML traditionnelle et permet de créer ainsi un cadre dans n'importe quelle page existante. Il est même possible de créer des iframes à l'aide d'un programme JavaScript, ce qui permet de mieux contrôler la création et la suppression des flux de communication entre le serveur et le navigateur.

Avantages des cadres cachés

Fonctionne sur les anciens navigateurs

Cette technique étant pratiquée depuis longtemps, elle peut être utilisée sur des navigateurs plus anciens qui ne supportaient pas encore les objets XMLHttpRequest. Il est donc possible d'utiliser la technique des cadres cachés en tant que solution alternative à Ajax si l'on désire que l'application fonctionne sur une plus grande variété de navigateurs.

Conserve l'historique du navigateur

La technique des cadres cachés permet de conserver l'historique du navigateur. Cette caractéristique permet aux internautes de continuer à utiliser les boutons Suivant et Précédent du navigateur contrairement aux applications Ajax. À noter que certaines applications couplent Ajax à la technique des cadres cachés pour remédier au problème des boutons Suivant et Précédent inactifs (comme Gmail et Google Maps par exemple).

Inconvénients des cadres cachés

Manque d'informations sur le traitement de la requête

Le principal inconvénient de la technique des cadres cachés est lié au manque d'informations concernant le traitement de la requête HTTP en arrière-plan. Cela pose de gros problèmes dans le cas où la page du cadre caché n'est pas chargée, car l'internaute peut attendre indéfiniment la réponse sans être informé de l'incident. Même s'il est possible de programmer une temporisation pour interrompre le traitement et informer l'utilisateur au bout d'un temps déterminé, il est préférable désormais d'utiliser un objet XMLHttpRequest qui nous permet de garder le contrôle de toutes les étapes du traitement de la requête HTTP.

4

HTTP et l'objet XMLHttpRequest

Bien qu'il ne soit pas récent, l'objet XMLHttpRequest n'a pas été très utilisé avant l'apparition d'Ajax (contrairement aux autres technologies utilisées telles que CSS, DOM, XML, JavaScript usuel…), aussi nous avons décidé de lui consacrer un chapitre complet et de vous le présenter dès maintenant afin que vous puissiez mieux comprendre le fonctionnement des communications asynchrones qui caractérisent les applications Ajax.

Ressources sur les technologies utilisées par Ajax

À la fin de cet ouvrage, vous trouverez de nombreuses ressources sur les technologies sous-jacentes d'Ajax que nous vous avons présentées dans le chapitre précédent (CSS, DOM, JavaScript, XML…). Nous avons opté pour cette organisation de sorte à ne pas surcharger inutilement la progression de votre apprentissage sur Ajax tout en vous permettant d'aller puiser des compléments d'information (voire vous initier) sur certaines technologies qui pourraient vous faire défaut pour la compréhension des scripts de cet ouvrage.

Rappels sur le protocole HTTP

Pour les applications Ajax et comme pour la plupart des applications Web traditionnelles, les transferts de données entre le serveur Web et le client (le navigateur) utilisent le protocole HTTP (HyperText Transfer Protocol). Pour mieux comprendre les rouages des mécanismes de l'objet XMLHttpRequest, il est intéressant de faire un petit rappel du mode de fonctionnement de ce protocole très souvent utilisé sur Internet.

Un transfert HTTP se compose de deux éléments : les requêtes (`GET` ou `POST`) envoyées par le navigateur et la réponse retournée au navigateur par le serveur.

Les requêtes HTTP

Pour envoyer une requête HTTP au serveur depuis un navigateur, deux méthodes peuvent être utilisées : `GET` ou `POST`.

Comment indiquer la méthode choisie en Ajax ?

Dans le cas d'un simple formulaire, on précise la méthode choisie en l'indiquant en tant que valeur de l'attribut `method` de la balise du formulaire (`<form method="get">` par exemple) alors que dans le cas d'Ajax, elle sera notifiée dans le premier paramètre de la méthode `open()` de l'objet `XMLHttpRequest` (`objetXHR.open('get','monFichier.php?val='+i,true)` par exemple. La méthode `open()` et les autres méthodes et propriétés de l'objet `XMLHttpRequest` sont détaillés plus loin dans ce même chapitre.

Requête avec la méthode GET

Une requête `GET` peut être initiée par un formulaire (dont l'attribut `method` serait configuré avec la valeur `GET`) mais aussi à partir d'un simple lien hypertexte auquel les valeurs à envoyer seront associées à l'URL sous forme de couple variable/valeur après un point d'interrogation (voir code 4-1).

Code 4-1 :

```
monProgramme.php?message=coucou
```

Si plusieurs couples de variable/valeur doivent être envoyés, ils sont alors séparés par une esperluette (voir code 4-2).

Code 4-2 :

```
monProgramme.php?message=coucou&nom=toto
```

Parmi les inconvénients de la méthode `GET`, notons que la taille d'une requête `GET` étant limitée à 255 octets (ou 1024 selon les systèmes), le nombre d'informations envoyées par cette méthode se trouve du même coup réduit. En outre, les valeurs étant envoyées dans l'URL, elles sont donc visibles par tous. Les requêtes `GET` ne pourront donc pas être utilisées pour envoyer des informations volumineuses ou si la confidentialité des informations doit être préservée.

Les avantages de la méthode `GET` sont principalement sa simplicité et la possibilité de construire dynamiquement des pseudo-URL dans lesquelles sont intégrées les valeurs à envoyer (ce qui est bien pratique dans les sites catalogues pour afficher la fiche d'un produit spécifique à partir des liens hypertextes d'une liste, par exemple : `fiche.php?id=5`). À noter que la méthode `GET` est aussi souvent utilisée dans les requêtes des moteurs de recherche en raison de la possibilité offerte à l'internaute d'enregistrer la requête dans ses favoris.

Quelle que soit l'origine d'une requête `GET` (formulaire ou lien hypertexte avec paramètres), le serveur réceptionnera un ensemble d'informations composées d'une ligne de requête et de plusieurs lignes d'en-tête (pour les requêtes `GET`, le corps de la requête est vide car les valeurs sont insérées dans la ligne de requête, voir code 4-4).

Code 4-3 : Exemple d'URL absolue avec ses paramètres :

```
http://www.eyrolles.fr/monRepertoire/monProgramme.php?message=coucou
```

Code 4-4 : Exemple simplifié d'une requête HTTP avec la méthode `GET` initiée par l'URL du code 4-3 :

```
1 : Ligne de requête
GET /monRepertoire/?message=coucou HTTP/1.1
2 : Lignes d'en-tête
Host: www.eyrolles.fr
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6) Gecko/20050225
➥Firefox/1.0.1
Connection: Keep-Alive
3 : Corps de la requête (vide dans le cas d'une requête GET)
```

Requête avec la méthode POST

Contrairement à la méthode `GET`, les valeurs envoyées avec la méthode `POST` ne sont pas visibles dans la barre d'adresse (ce qui est appréciable si vous désirez envoyer des mots de passe par exemple…) mais placées dans une variable d'environnement insérée dans le corps de la requête (et donc invisible par l'internaute). Le nombre d'informations n'est plus limité comme pour la méthode `GET` et cette méthode offre ainsi la possibilité de transmettre des flux de données beaucoup plus importants (jusqu'à 2Go). Par contre, les données envoyées avec la méthode `POST` ne peuvent être transmises que par un formulaire (ou par la méthode `open()` d'un objet `XMLHttpRequest` en ce qui concerne les applications Ajax) et non plus par une URL comme avec la méthode `GET`.

À noter enfin que, contrairement à une requête `GET` pour laquelle certains navigateurs (IE par exemple) récupèrent automatiquement les précédentes données dans la mémoire cache, il n'en n'est pas de même avec la méthode `POST` qui nécessite l'autorisation de l'internaute pour renvoyer les paramètres (quel que soit le navigateur utilisé).

Dans le cas d'une requête `POST`, le serveur réceptionnera un ensemble d'informations composées d'une ligne de requête, de plusieurs lignes d'en-tête et d'un corps de requête contenant les valeurs à transmettre (voir code 4-5).

Code 4-5 : Exemple simplifié d'une requête HTTP avec la méthode `POST` initiée par un formulaire contenant un seul champ nommé « message » :

```
1 : Ligne de requête
POST /monRepertoire/ HTTP/1.1
2 : Lignes d'en-tête
Host: www.eyrolles.fr
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6) Gecko/20050225
➥Firefox/1.0.1
Content-type: application/x-www-form-urlencoded
Content-Length: 15
Connection: Keep-Alive
3 : Corps de la requête
message=coucou
```

Si l'on compare les informations d'une requête `POST` (voir code 4-5) avec celles d'une requête `GET` (voir code 4-4), on remarque que les données à transmettre sont maintenant insérées dans le corps de la requête et que deux nouveaux en-têtes ont été ajoutés. L'en-tête `Content-type` qui précise l'encodage utilisé (à noter qu'il s'agit toujours du même type MIME quel que soit le navigateur) pour le corps de la requête — qui contient maintenant les données — et l'en-tête `Content-Length` qui indique la taille en octets des données contenues dans le corps de la requête.

La réponse HTTP

Le traitement par le serveur d'une requête HTTP renvoie un ensemble d'informations au navigateur qui a initié la demande (voir code 4-6). Si on observe ces informations, on peut distinguer trois parties différentes :

La **ligne de statut** (voir le repère 1 du code 4-6) qui rappelle la version du protocole HTTP utilisé (`HTTP/1.1` dans l'exemple) suivi du statut de la réponse (information importante pour Ajax comme `200`, qui signifie le succès de l'opération) et d'un message complémentaire (`OK` par exemple).

Un nombre variable de **lignes d'en-tête** (voir le repère 2 du code 4-6) contenant entre autres l'en-tête `Content-Type` qui indique le type MIME du résultat (information importante pour Ajax). Ce type peut par exemple être `text/plain` pour du texte brut, `text/html` pour une page HTML ou encore `text/xml` pour du XML. Signalons aussi l'en-tête `Cache-Control` intéressant pour Ajax car il permet d'interagir sur le cache du navigateur.

Le **contenu de la réponse** renvoyée (voir le repère 3 du code 4-6) peut être une page HTML complète comme dans l'exemple ci-dessous mais dans le cas d'Ajax, le contenu de la réponse sera surtout constitué de données seules (au format texte ou XML), ou encore de fragments HTML pour les réponses générées suite à une requête d'une application Ajax. En effet, nous avons vu précédemment que l'un des avantages d'une application Ajax est justement de pouvoir récupérer le minimum utile du serveur et non pas la page complète comme pour les requêtes Web traditionnelles.

Code 4-6 : Exemple d'une réponse HTTP serveur :

```
1 : Ligne de statut
```

```
HTTP/1.1 200 OK
```

```
2 : Lignes d'en-tête
```

```
Cache-Control: private
Content-type: text/html
Server: GWS/2.1
Date:Tue, 15 Jun 2007 13:20:24 GMT
```

```
3 : Contenu de la réponse
```

```
<html>
<head>
<title>Réponse</title>
</head>
<body>
Hello
</body>
</html>
```

Selon la réponse du serveur, le code du statut HTTP peut changer. Nous verrons par la suite que ce code devra être testé par le script du moteur Ajax afin de s'assurer que l'opération a été effectuée avec succès. Il est donc utile de connaître les principaux codes de statut HTTP que le serveur peut renvoyer. Ces codes sont rappelés dans le tableau 4-1.

Tableau 4-1 Principaux codes de statut HTTP.

Code de statut	Signification	Exemple/Message
200 à 299	Succès de l'opération	200/OK : Requête accomplie avec succès
300 à 399	Redirection	301/Moved Permanently : Redirection permanente
400 à 499	Erreur client	404/Not Found : Document non trouvé
500 à 599	Erreur serveur	503/Service Unavailable : Service non disponible

Caractéristiques de l'objet XMLHttpRequest

La communication des applications Ajax avec le serveur repose essentiellement sur l'objet XMLHttpRequest. Cette classe JavaScript permet d'exécuter des requêtes HTTP du navigateur vers le serveur d'une manière asynchrone (réponse différée sans blocage de l'application client) sans avoir à recharger la page HTML comme c'est le cas lors d'une requête HTTP traditionnelle.

Déjà opérationnel depuis 1998

L'objet XMLHttpRequest n'est pas récent car la classe qui permet de l'instancier a vu le jour en 1998. À l'époque, Microsoft l'avait développé pour l'intégrer sous forme de contrôle ActiveX dans son navigateur Internet Explorer 5.0.

Très peu utilisé jusqu'à l'avènement Ajax, il a cependant été implémenté progressivement dans la plupart des navigateurs et même fait l'objet d'un projet de spécification du W3C depuis 2006.

Tableau 4-2 Progression de l'implémentation de la classe XMLHttpRequest dans les différents navigateurs

Année d'implementation	Navigateur(s) concerné(s)
1998	Internet Explorer
2002	Mozilla
2004	Safari
2005	Konqueror et Opera

Une instanciation en cours d'homologation

Si cette classe est implémentée dans les principaux navigateurs du moment, la procédure d'instanciation d'un objet n'est malheureusement pas encore standardisée pour tous (même si l'homogénéisation de son utilisation devrait bientôt voir le jour). En effet, comme à l'origine la classe a été créée en tant que contrôle ActiveX, la procédure d'instanciation avec Internet Explorer (utilisation du constructeur `ActiveXObject()`) est différente des autres navigateurs (utilisation du constructeur `XMLHttpRequest()`).

De plus, la valeur à passer en paramètre lors de l'instanciation d'un objet dans un navigateur Microsoft est différente entre la première version d'Internet Explorer sur lequel cette classe a été implémentée (la version IE 5.0) et les versions ultérieures. On se retrouve donc avec 3 syntaxes d'instanciation différentes selon le navigateur utilisé !

Pour contourner ce problème, une méthode judicieuse consiste à développer une fonction de création générique de l'objet XMLHttpRequest prenant en compte la détection du navigateur (et aussi sa version pour IE) afin d'utiliser la bonne syntaxe d'instanciation selon le contexte. Cette fonction peut être avantageusement intégrée dans un fichier JavaScript externe qui sera lié aux pages HTML utilisant Ajax. À noter que si vous utilisez des frameworks (comme jQuery ou Prototype par exemple) la fonctionnalité de détection du navigateur est en général intégrée dans leur bibliothèque JavaScript.

Tableau 4-3 Syntaxe à utiliser pour l'instanciation d'un objet XMLHttpRequest selon les navigateurs et leurs versions.

Navigateurs concernés	Syntaxe d'instanciation d'un objet XMLHttpRequest
Internet Explorer 5.0	new ActiveXObject("Microsoft.XMLHttp")
Internet Explorer version ultérieure à 5.0	new ActiveXObject("Msxml2.XMLHttp")
Autres navigateurs (Firefox…)	new XMLHttpRequest( )

Pour illustrer la création d'un objet XMLHttpRequest en JavaScript, vous trouverez ci-dessous un exemple de fonction générique de création d'un objet XMLHttpRequest selon le navigateur utilisé.

Code 4-7 : Fonction générique de création d'un objet XMLHttpRequest :

```
function creationXHR() {
var resultat=null;
try {//test pour les navigateurs : Mozilla, Opera...
  resultat= new XMLHttpRequest();
  }
catch (Error) {
   try {//test pour les navigateurs Internet Explorer > 5.0
   resultat= new ActiveXObject("Msxml2.XMLHTTP");
   }
   catch (Error) {
     try {//test pour le navigateur Internet Explorer 5.0
     resultat= new ActiveXObject("Microsoft.XMLHTTP");
     }
     catch (Error) {
      resultat= null;
     }
   }
 }
return resultat;
}
```

La fonction JavaScript ci-dessus peut être incorporée directement dans la page HTML mais il est plus judicieux de la saisir dans un fichier JS externe et de le lier ensuite à la page HTML concernée afin de pouvoir en disposer facilement dans les autres pages du site.

```
var objetXHR = creationXHR();
```

```
La ligne de code ci-dessus est un exemple d'utilisation de la fonction générique pour
créer un objet XMLHttpRequest.
```

La fonction du code 4-7 teste successivement la création de l'objet XMLHttpRequest avec des syntaxes d'instanciation différentes correspondant aux trois types de navigateurs. La structure de gestion des exceptions (`try`) permet de tester une fonction sans pour autant générer d'erreur. Si le test est positif, l'objet est créé et, dans le cas contraire, elle passe au test suivant et poursuit le même processus (`catch`).

Ressources sur les technologies associées

Si la syntaxe du code de cette fonction ne vous semble pas évidente, vous pouvez vous référer aux ressources sur le JavaScript (et notamment sur la gestion des exceptions : `try`, `catch`) regroupées à la fin de cet ouvrage.

Propriétés et méthodes de l'objet XMLHttpRequest

Comme toutes les classes, XMLHttpRequest possède des propriétés et des méthodes. Avant de les utiliser pour mettre en œuvre une requête serveur, nous vous proposons de les présenter dans les tableaux 4-4 et 4-5 ci-dessous.

Tableau 4-4 Propriétés de l'objet XMLHttpRequest.

Propriétés	Description
onreadystatechange	Désigne une fonction de rappel qui sera appelée à chaque fois que l'état du traitement d'une requête asynchrone (readyState) changera d'état.
readyState	Etat du traitement d'une requête asynchrone (valeur numérique de 0 à 4, voir le tableau 4-6 pour connaître les différents états et leur signification).
responseText	Contient le résultat du serveur sous forme de texte.
responseXml	Contient le résultat du serveur sous forme XML.
status	Contient le code de statut HTTP (exemple : 200, voir tableau 4-1). Ce code est disponible en lecture uniquement lorsque la réponse a été transmise.
statusText	Contient le message de statut HTTP (exemple : OK, voir tableau 4-1). Ce texte est disponible en lecture uniquement lorsque la réponse a été transmise.

Tableau 4-5 Méthodes de l'objet XMLHttpRequest

Méthodes	Description
abort()	Annule la requête en cours.
getAllResponseHeaders()	Retourne dans une chaîne de caractères tous les en-têtes HTTP contenus dans la réponse du serveur. Cette méthode n'est utilisable que lorsque l'état du traitement (readyState) est égal à 3 ou 4 (Interactive ou Completed).
getResponseHeader("*nomEnTete*")	Retourne la valeur de l'en-tête dont le nom est indiqué dans le paramètre (*nomEnTete*). Cette méthode n'est utilisable que lorsque l'état du traitement (readyState) est égal à 3 ou 4 (Interactive ou Completed).

Tableau 4-5 Méthodes de l'objet XMLHttpRequest *(suite)*

open("*methode*","*url*","*async*")	Initialise l'objet en précisant certains paramètres de la requête : la méthode utilisée (*methode* : GET ou POST), l'URL du script côté serveur (*url* : fichier.php par exemple) et un troisième paramètre en option pour indiquer si la communication doit être asynchrone (*async* : true) ou synchrone (*async* : false).
send("*contenu*")	Envoie la requête. Le paramètre (*contenu*) pourra prendre la valeur `null` dans le cas d'un requête initialisée avec la méthode GET ou une chaîne de requête dans le cas d'une méthode POST. Évidemment cette méthode ne peut être utilisée que lorsque la méthode open() a déjà été configurée et donc que lorsque l'état du traitement (readyState) est égal à 1 (Loading).
setRequestHeader("*nom*","*valeur*")	Attribue une valeur (*valeur*) à l'en-tête de la requête dont le nom est spécifié dans le premier paramètre (*nom*). Cette méthode n'est utilisable que lorsque l'état du traitement (readyState) est égal à 1 (Loading).

Tableau 4-6 Signification des différents états de traitement d'une requête asynchrone (accessible grâce à la propriété « readyState »)

Valeur	Etat	Signification
0	Unitialized	L'objet n'a pas encore été initialisé et donc la méthode `open()` n'a pas encore été appelée.
1	Loading	L'objet a été initialisé mais la requête n'a pas encore été envoyée à l'aide de la méthode `send()`.
2	Loaded	La requête a été envoyée à l'aide de la méthode `send()`.
3	Interactive	La réponse est en cours de réception.
4	Completed	La réponse du serveur est complètement réceptionnée. Les données sont disponibles dans la propriété `responseText` (s'il s'agit de données texte) ou dans `responseXML` (s'il s'agit de données XML).

Restriction d'accès aux domaines externes

Pour éviter des utilisations malveillantes d'une application Ajax qui récupérerait des données d'un autre site sans l'autorisation de son propriétaire, les navigateurs ont intégré un système de sécurité qui interdit les appels Ajax vers des domaines différents de celui dans lequel se trouve l'application Ajax.

Cependant, dans certains cas il est nécessaire d'accéder à des ressources placées sur un serveur externe ou mis à disposition par un fournisseur d'information (cas des flux RSS par exemple). Pour détourner cette contrainte, il est alors possible de mettre en œuvre un serveur mandataire (proxy) qui aura pour fonction de relayer des requêtes entre le serveur Web et le serveur fournisseur d'information.

Création de moteurs Ajax de base

Envoi d'une requête synchrone sans paramètre

Même si l'objet XMLHttpRequest est très souvent exploité en mode asynchrone, il est aussi capable de gérer des requêtes synchrones. Dans ce cas, le troisième paramètre de la méthode `open()` doit être configuré avec la valeur `false` et le traitement sera séquentiel (voir code 4-8). Ce mode de fonctionnement s'apparente plutôt au processus d'une requête Web traditionnelle (blocage de l'application client dans l'attente de la réponse du serveur) et l'utilisation de l'objet XMLHttpRequest a beaucoup moins d'intérêt que dans le mode asynchrone que nous allons traiter ci-après.

Codes didactiques

Les différents codes de ce chapitre ont été élaborés pour vous permettre de comprendre les mécanismes d'une application Ajax d'une manière progressive. Ils sont souvent incomplets et nous vous conseillons de ne pas essayer de les tester en pratique mais de vous concentrer plutôt sur leur analyse. Nous reprendrons ces scripts de base à partir du chapitre 7 dans lequel vous aurez tout loisir de mettre en œuvre vos premières applications Ajax.

Code 4-8 : Exemple de traitement d'une requête synchrone (ce code nécessite la déclaration préalable de la fonction `creationXHR()` définie dans le code 4-7) :.

```
###############-MOTEUR AJAX-################
//instanciation de l'objet XMLHttpRequest
var objetXHR = creationXHR();
//----Traitement SYNCHRONE
//Configuration de la méthode utilisée, du script serveur ciblé
//et enfin activation du type synchrone en dernier paramètre
objetXHR.open("get","script.php",false);
//envoi de la requête
objetXHR.send(null);
//récupération de la réponse du serveur
var resultat = objetXHR.responseText ;
….
//Le résultat peut maintenant être inséré dans la page HTML
//si le script serveur correspond à celui de l'encadré, la valeur
//de la variable "resultat" serait alors égale à "Bonjour"
###############-FIN DU MOTEUR AJAX-################
```

La première ligne de code crée l'objet XMLHttpRequest par instanciation de sa classe. Dès que l'objet est créé, il est alors possible d'initialiser la requête à l'aide de la méthode `open()`. Dans le cas d'une requête synchrone, le troisième paramètre de cette méthode devra être configuré avec la valeur `false`. Il suffit ensuite d'envoyer la requête avec la méthode `send(null)` (à noter que l'argument de la méthode `send()` est égal à `null` car la méthode de la requête est initialisée avec `GET`) et d'attendre la réponse dans la propriété `responseText` de l'objet.

Réponse HTTP du serveur à une requête sans paramètre

Pour que vous puissiez bien appréhender tous les aspects de ce processus de communication HTTP, nous vous proposons un exemple de script serveur très simple qui pourra être utilisé avec les exemples de moteurs Ajax sans paramètre que nous vous proposons dans cette partie. Ce script est volontairement minimaliste afin que vous puissiez concentrer votre attention sur le moteur Ajax et non sur le traitement côté serveur.

Code 4-9 : Exemple de code PHP du fichier script.php :

```
<?php
//indique que le type de la réponse renvoyée au client sera du texte
header("Content-Type: text/plain");
//retourne la réponse au client à l'aide de la commande echo
echo "Bonjour";
?>
```

☞

Réponse HTTP du serveur à une requête sans paramètre *(suite)*

Dans ce contexte, la réponse HTTP du serveur à une requête simple (sans paramètre) pourrait ressembler à l'exemple ci-dessous.

```
HTTP/1.1 200 OK
Cache-Control: private
Content-type: text/plain
Server: GWS/2.1
Date:Tue, 15 Jun 2007 13:20:24 GMT

Bonjour
```

Envoi d'une requête asynchrone sans paramètre

Contrairement au mode synchrone, le mode asynchrone nécessite l'utilisation d'une fonction de rappel. Cette fonction de rappel devra être déclarée et désignée avant l'envoi de la requête de sorte à ce que le moteur Ajax puisse ensuite l'appeler lors de la réception de la réponse du serveur. Pour désigner la fonction de rappel, son nom devra être mémorisé dans la propriété `onreadystatechange` de l'objet XHR (voir repère 1 dans le code 4-10). Ainsi, la fonction désignée comme fonction de rappel sera appelée à chaque changement de la propriété `readyState`. La propriété `readyState` correspond aux différents états de traitement d'une requête Ajax (revoir tableau 4-6), elle changera donc plusieurs fois d'état au cours du cycle de chaque requête. Comme nous ne désirons récupérer le résultat que lorsqu'il sera complètement réceptionné, il faut donc ajouter un test dans la fonction de rappel (voir repère 2 du code 4-10) afin de s'assurer que la propriété `readyState` est égale à la valeur 4 (état `Completed`) avant de copier le résultat dans une variable pour sa future exploitation (voir code 4-10).

Code 4-10 : Exemple de traitement d'une requête asynchrone (ce code nécessite la déclaration préalable de la fonction `creationXHR()` définie dans le code 4-7) :

```
##############-MOTEUR AJAX-################
//instanciation de l'objet XMLHttpRequest
var objetXHR  =  creationXHR();
//Déclaration de la fonction de rappel
function traitementResultat() {
  if(objetXHR.readyState==4) {❷
  var resultat = objetXHR.responseText ;
  …
    //####Le résultat peut maintenant être inséré dans la page HTML
    //si le script serveur correspond à celui de l'encadré,
    //la valeur de la variable "resultat" sera
    //alors égale à "Bonjour"
  }
}
//----Traitement ASYNCHRONE
//Désignation de la fonction de rappel
objetXHR.onreadystatechange = traitementResultat ; ❶
//Configuration de la méthode utilisée, du script serveur ciblé
//et enfin activation du type asynchrone en dernier paramètre
objetXHR.open("get","script.php",true);
//envoi de la requête
objetXHR.send(null);
###############-FIN DU MOTEUR AJAX-################
```

Ateliers pédagogiques sur les requêtes asynchrones

Le fonctionnement de la fonction de rappel et de la propriété `readyState` d'une requête asynchrone sera revu en détail dans les ateliers pédagogiques du chapitre 9.

Ajout d'un traitement des erreurs HTTP du serveur

Les scripts précédents fonctionnent très bien tant qu'il n'y a pas de problème lors du transfert HTTP. Imaginez maintenant que le script serveur soit supprimé ou déplacé. Dans ce cas les scripts ne fonctionneront plus et rien n'indiquera la nature du problème. Pourtant le serveur dispose d'une variable qui indique les erreurs HTTP, il s'agit bien sûr du code de statut (revoir tableau 4-1). De même, l'objet XMLHttpRequest possède une propriété `status` qui permet de lire facilement cette information. Nous allons donc modifier le script de la fonction de rappel pour y ajouter un second test afin de s'assurer que tout s'est bien passé avant d'exploiter le résultat (dans ce cas la propriété `status` doit être égale à 200). Dans le cas contraire, nous affichons un message d'erreur afin d'en informer les utilisateurs.

Code 4-11 : Exemple d'une requête asynchrone avec traitement des éventuelles erreurs HTTP (ce code nécessite la déclaration préalable de la fonction `creationXHR()` définie dans le code 4-7) :

```
################-MOTEUR AJAX-##################
//instanciation de l'objet XMLHttpRequest
var objetXHR = creationXHR();
//Déclaration de la fonction de rappel
function traitementResultat() {
 if(objetXHR.readyState==4) {
   if(objetXHR.status==200) {
   var resultat = objetXHR.responseText ;
   …
   //####Le résultat peut maintenant être inséré dans la page HTML
   //si le script serveur correspond à celui de l'encadré,
   //la valeur de la variable "resultat" sera
   //alors égale à "Bonjour"
   }else {
   alert("Erreur HTTP N°"+ objetXHR.status);
   }
 }
}
//----Traitement ASYNCHRONE
//Désignation de la fonction de rappel
objetXHR.onreadystatechange = traitementResultat ;
//Configuration de la méthode utilisée, du script serveur ciblé
//et enfin activation du type asynchrone en dernier paramètre
objetXHR.open("get","script.php",true);
//envoi de la requête
objetXHR.send(null);
################-FIN DU MOTEUR AJAX-##################
```

Envoi d'une requête asynchrone avec un paramètre GET

En général les requêtes Ajax sont accompagnées de paramètres afin d'indiquer au script serveur les données à retourner. La manière la plus simple d'envoyer un paramètre est d'utiliser la méthode `GET` et d'ajouter un paramètre d'URL à la suite de l'adresse du script ciblé comme par exemple : `scriptAvecParametre.php?id=2` (pour l'utilisation de ce paramètre côté serveur, voir code 4-12).

Ce procédé a aussi l'avantage de favoriser la mise au point du script serveur, puisqu'il suffit de l'appeler individuellement dans le navigateur en ajoutant le paramètre d'URL manuellement à la suite de son nom (pour cela, saisir son URL complète dans la barre d'adresse du navigateur comme par exemple `http://localhost/scriptAvecParametre.php?id=2`). Le script ainsi appelé avec ce paramètre doit afficher à l'écran les mêmes données qu'il renverra ensuite à l'objet XMLHttpRequest lorsque le système sera opérationnel (soit dans notre exemple « Bonjour Monsieur Dupond »). Cette technique vous permet de vérifier que le script serveur fonctionne correctement et d'aiguiller éventuellement vos recherches vers le programme JavaScript côté client si le problème subsiste.

Côté serveur, le paramètre (`?id=2` par exemple) est récupéré par le biais de la variable `$_REQUEST['id']` avant d'être exploité dans le programme de construction du résultat (voir code 4-12). Celui-ci sera ensuite affiché à l'écran pour être récupéré en différé dans la propriété `responseText` de la fonction de rappel du moteur Ajax côté client (voir la fonction `traitementResultat()` code 4-13).

Réponse HTTP du serveur à un paramètre de la requête

Pour les moteurs Ajax incluant un paramètre dans l'envoi de la requête, nous avons légèrement modifié le script serveur de sorte qu'il envoie le message d'accueil correspondant au numéro de l'identifiant de l'utilisateur concerné.

Code 4-12 : Exemple de code PHP du fichier `scriptAvecParametre.php` :

```
<?php
//indique que le type de la réponse renvoyée au client sera du texte
header("Content-Type: text/plain");
//récupération de la variable GET
if(isset($_REQUEST['id'])) $id=$_REQUEST['id']; else $id=0;
//affectation du nom correspondant à l'identifiant
if($id==1) $nom="Dumoulin";
elseif($id==2) $nom="Dupond";
elseif($id==3) $nom="Marchand";
else $nom="Inconnu";
//mise en forme et renvoi de la réponse au client
echo "Bonjour Monsieur ".$nom;
?>
```

Dans ce contexte, la réponse HTTP du serveur pourrait ressembler à l'exemple ci-dessous si le paramètre de la requête est id=2.

```
HTTP/1.1 200 OK
Cache-Control: private
Content-type: text/plain
Server: GWS/2.1
Date:Tue, 15 Jun 2007 13:20:24 GMT

Bonjour Monsieur Dupond
```

Code 4-13 : Exemple d'une requête asynchrone accompagnée d'un paramètre transmis avec la méthode `GET` (ce code nécessite la déclaration préalable de la fonction `creationXHR()` définie dans le code 4-7) :

```
################-MOTEUR AJAX-##################
//instanciation de l'objet XMLHttpRequest
var objetXHR = creationXHR();
```

```
//Déclaration de la fonction de rappel
function traitementResultat() {
 if(objetXHR.readyState==4) {
  if(objetXHR.status==200) {
  var resultat = objetXHR.responseText ;
  …
  //####Le résultat peut maintenant être inséré dans la page HTML
  //si le script serveur correspond à celui de l'encadré,
  //la valeur de la variable "resultat" sera
  //alors égale à " Bonjour Monsieur Dupond "
  }else {
  alert("Erreur HTTP N°"+ objetXHR.status);
  }
 }
}
//----Traitement ASYNCHRONE - GET
//Désignation de la fonction de rappel
objetXHR.onreadystatechange = traitementResultat ;
//Configuration de la méthode utilisée, du script serveur avec
//les paramètres à transmettre au serveur
//et enfin activation du type asynchrone en dernier paramètre
var numero=2; //simulation du choix d'un numéro d'identifiant
objetXHR.open("get","scriptAvecParametre.php?id="+numero,true);
//envoi de la requête
objetXHR.send(null);
################-FIN DU MOTEUR AJAX-##################
```

Envoi d'une requête asynchrone avec un paramètre POST

Une autre alternative pour envoyer un paramètre au serveur consiste à configurer la requête avec la méthode `POST`. Cette technique est en général utilisée quand les données des paramètres sont importantes (supérieure à 512 octets) ce qui est assez rare en pratique. D'autre part, cette méthode est moins facile à mettre en œuvre car si vous désirez vérifier le bon fonctionnement du script PHP seul, il faut alors créer un formulaire `POST` pour simuler l'envoi de la requête Ajax.

À noter cependant que si vous désirez envoyer vos paramètres au serveur en XML et non au format texte sous forme de couple variable/valeur, il faudra dans cas utiliser la méthode `POST` pour mettre en œuvre votre application.

Le fonctionnement est néanmoins semblable (voir code 4-14) hormis le fait que le paramètre est passé en argument de la méthode `send()` (comme par exemple : `send(id=2)`) et non plus à la suite de l'URL du fichier serveur dans le second paramètre de la méthode `open()` (pour mémoire : `scriptAvecParametre.php?id=2`). En méthode `POST`, il faut indiquer le type des données envoyées par le biais de la méthode `setRequestHeader` qui affectera le type correspondant à l'en-tête `Content-Type` avant d'envoyer la requête (voir code 4-14). En effet, quand un formulaire traditionnel dans un navigateur envoie des données en méthode `POST`, le navigateur se charge d'affecter automatiquement la valeur `application/x-www-form-urlencoded` à l'en-tête `Content-Type` de la requête. Dans le cas d'une requête Ajax, il faut réaliser manuellement cette affectation, car lorsque les données `POST` arriveront sur le serveur, PHP recherchera leur type d'encodage dans cet en-tête afin de les analyser correctement.

Côté serveur il n'y a pas de différence (le fichier serveur sera donc celui du code 4-12) car pour la récupération du paramètre nous avons utilisé la variable HTTP `$_REQUEST['id']` qui

permet de récupérer des variables envoyées par un client quelle que soit leur méthode (contrairement à `$_GET['id']` et `$_POST['id']` qui sont dédiés à chacune des méthodes).

Code 4-14 : Exemple d'une requête asynchrone accompagnée d'un paramètre transmis avec la méthode `POST` (ce code nécessite la déclaration préalable de la fonction `creationXHR()` définie dans le code 4-7) :

```
###############-MOTEUR AJAX-################
//instanciation de l'objet XMLHttpRequest
var objetXHR = creationXHR();
//Déclaration de la fonction de rappel
function traitementResultat() {
 if(objetXHR.readyState==4) {
  if(objetXHR.status==200) {
  var resultat = objetXHR.responseText ;
  ….
  //Le résultat peut maintenant être inséré dans la page HTML
  //si le script serveur correspond à celui de l'encadré,
  //la valeur de la variable "resultat" sera
  //alors égale à " Bonjour Monsieur Dupond "
  }else {
  alert("Erreur HTTP N°"+ objetXHR.status);
  }
 }
}
//----Traitement ASYNCHRONE - POST
//Désignation de la fonction de rappel
objetXHR.onreadystatechange = traitementResultat ;
//Configuration de la méthode utilisée, du script serveur ciblé
//et enfin activation du type asynchrone en dernier paramètre
objetXHR.open("post","scriptAvecParametre.php",true);
//Affectation du type d'encodage de la requête envoyée
objetXHR.setRequestHeader("Content-Type","application/x-www-form-urlencoded")
//simulation du choix d'un numéro d'identifiant
var numero=2;
//envoi de la requête avec un paramètre
objetXHR.send(id=numero);
###############-FIN DU MOTEUR AJAX-################
```

Récupération du résultat de la requête avec responseText ou responseXML

Pour récupérer le résultat renvoyé par le serveur, nous avons utilisé jusqu'à présent (dans la fonction de rappel `traitementResultat()`) la propriété `responseText` de l'objet XMLHttpRequest (voir code 4-15) si la réponse est au format texte.

Code 4-15 : Récupération de la réponse du serveur dans une variable `resultat`.

```
var resultat = objetXHR.responseText ;
```

Par la suite nous verrons qu'il est aussi possible de récupérer des résultats plus complexes au format XML. Dans ce cas, il faudra utiliser la propriété `responseXML` du même objet.

Rappelons enfin que le résultat du serveur n'est disponible dans les propriétés `responseText` ou `responseXML` que lorsque le cycle de transfert HTTP est terminé et que l'état de la propriété `readyState` est égal à `4` (`Completed`). C'est d'ailleurs pour cette raison que nous avons ajouté le test du code 4-16 dans la fonction de rappel car celle-ci est appelée à chacune des 4 modifications de la propriété `readyState` d'un cycle de transfert HTTP et nous désirons récupérer la réponse du serveur uniquement au terme de ce cycle.

Code 4-16 : Test de l'état de la propriété `readyState` :

```
//---Fonction de rappel
function traitementResultat() {
 if(objetXHR.readyState==4) {
  if(objetXHR.status==200) {
  var resultat = objetXHR.responseText ;
  }
 }
}
```

Utilisation de innerHTML pour afficher le résultat de la requête

Dans les scripts précédents, nous avons récupéré la réponse du serveur dans une variable `resultat` (voir code 4-15). Si nous désirons maintenant l'afficher dans la page, la méthode la plus simple consiste à utiliser l'attribut `innerHTML`. Cet attribut permet de remplacer le contenu d'un élément par une chaîne de caractères qui lui est affectée.

Pour cela, il faut commencer par cibler un élément de la page (`<span id="message">` par exemple, voir code 4-17) en se référant à son identifiant à l'aide de la méthode `getElementById()` puis exploiter son attribut `innerHTML` comme dans l'exemple ci-dessous :

```
document.getElementById("monMessage").innerHTML
```

On pourra ainsi remplacer son contenu par la réponse du serveur en lui affectant la propriété `responseText` de l'objet XMLHttpRequest (voir code 4-17). Ainsi, la réponse du serveur (« Bonjour » par exemple) s'affichera dans la balise `span` correspondante.

Code 4-17 : Affichage du résultat dans une balise `<span>` de la page (code partiel) :

```
<script language="JavaScript">
###############-MOTEUR AJAX-################
//instanciation de l'objet XMLHttpRequest
var objetXHR = creationXHR();
//Déclaration de la fonction de rappel
function traitementResultat() {
 if(objetXHR.readyState==4) {
  if(objetXHR.status==200) {
  document.getElementById("monMessage").innerHTML=objetXHR.responseText ;
  }
 }
}
//----Traitement ASYNCHRONE - GET
//Désignation de la fonction de rappel
objetXHR.onreadystatechange = traitementResultat ;
//Configuration de l'objet XHR
objetXHR.open("get","script.php",true);
//envoi de la requête
objetXHR.send(null);
###############-FIN DU MOTEUR AJAX-################
</script>
…
//avant l'affectation :
<span id="message"></span>
//après l'affectation :
<span id="message"> Bonjour </span>
```

À noter cependant que la propriété `innerHTML` n'est pas normalisée par le W3Cw et qu'il est préférable d'utiliser les méthodes du DOM même si celles-ci sont plus complexes à mettre en œuvre (ces méthodes seront présentées par la suite dans cet ouvrage).

Utilisation d'un gestionnaire d'événement pour déclencher l'envoi de la requête

Pour que le moteur Ajax s'exécute, il faut déclencher son appel dans la page HTML de l'application. Pour illustrer cela, nous vous proposons de mettre en place un gestionnaire d'événement très simple qui appellera une fonction contenant le moteur Ajax. Dans notre exemple, nous utiliserons le gestionnaire d'événement `onkeypress` afin que la fonction `afficheMessage()` (regroupant le moteur Ajax) soit invoquée dès que l'utilisateur appuiera sur une touche quelconque du clavier. Evidemment dans vos futures applications, vous pourrez déclencher de la même manière l'appel du moteur Ajax avec l'un des nombreux gestionnaires d'événements que JavaScript met à votre disposition (voir les ressources sur le DOM à la fin de cet ouvrage pour plus d'information au sujet des gestionnaires d'événements).

Code 4-18 : Déclenchement du moteur Ajax à l'aide du gestionnaire d'événement `onkeypress` :

```
<script language="JavaScript">
//-----Gestionnaire d'événement
window.onkeypress = afficheMessage;
###############-MOTEUR AJAX-################
//Déclaration de l'objet
var objetXHR;
//Déclaration de la fonction de rappel
function traitementResultat() {
 if(objetXHR.readyState==4) {
  if(objetXHR.status==200) {
  document.getElementById("monMessage").innerHTML=objetXHR.responseText ;
  }
 }
}
//Déclaration de la fonction d'appel du moteur Ajax
function afficheMessage() {
  //Instanciation de l'objet XMLHttpRequest
  objetXHR = creationXHR();
  //Désignation de la fonction de rappel
  objetXHR.onreadystatechange = traitementResultat ;
  //Configuration de l'objet XHR
  objetXHR.open("get","script.php",true);
  //Envoi de la requête
  objetXHR.send(null);
}
###############-FIN DU MOTEUR AJAX-################
</script>
…
//avant qu'une touche ne soit actionnée :
<span id="message"></span>
//après qu'une touche a été actionnée :
<span id="message"> Bonjour </span>
…
```

Partie II

Environnement de développement

Pour développer des applications AJAX-PHP, vous devez disposer d'un serveur d'évaluation PHP-MySQL, d'un éditeur de code polyvalent (pour écrire des pages HTML ainsi que des programmes PHP et JavaScript) et d'un navigateur (qui sera aussi utilisé pour déboguer les scripts). La présente partie a pour but de vous accompagner dans la mise en œuvre de cet environnement de développement afin de vous permettre de réaliser rapidement vos premières applications.

5

Firefox, navigateur et débogueur à la fois

Pour tester le bon fonctionnement des applications Ajax, nous utiliserons bien sûr un navigateur mais ce dernier servira aussi d'outil de débogage. Pour cela, il convient d'installer des extensions *ad hoc* que nous allons détailler dans ce chapitre.

Le navigateur Firefox

Même si Internet Explorer reste encore le navigateur le plus utilisé actuellement, nous utiliserons Firefox pour tester nos applications dans le cadre de cet ouvrage.

En effet, Firefox a de nombreux avantages par rapport à son concurrent : moins vulnérable et plus conforme aux standards, il dispose aussi de nombreuses extensions qui en font aujourd'hui l'outil indispensable au développement Web. Parmi toutes ses extensions, deux d'entre elles nous intéressent particulièrement :

- La première est l'extension « firebug », elle permet de transformer votre navigateur en outil de débogage. Nous pourrons ainsi analyser le fonctionnement d'une application et facilement localiser les éventuelles erreurs qui pourront se glisser dans votre programme JavaScript en contrôlant son déroulement.
- La seconde extension se nomme « IE Tab », elle permet d'activer un émulateur du navigateur Internet Explorer tout en continuant d'utiliser Firefox. En effet les principaux problèmes dans la réalisation de scripts client sont souvent liés à des incompatibilités entre navigateurs, aussi, avec cette extension, il sera très pratique de passer de l'un à l'autre d'un simple clic lors de la phase de test.

Installation de Firefox

Mise à jour de Firefox 2.0

Si vous avez déjà une version de Firefox installée sur votre ordinateur il vous est recommandé de désinstaller l'ancienne version et de procéder à une installation complète de

Firefox 2.0 car l'installer par dessus une version existante peut quelquefois causer des problèmes imprévisibles. Pour désinstaller votre ancienne version il faut utiliser la rubrique ajout/suppression de programmes de Windows, avant de commencer à installer la nouvelle version (voir procédure ci-après). Les paramètres, préférences et autres marque-pages seront conservés dans la nouvelle version car ils sont sauvegardés directement dans votre profil utilisateur.

Première installation

Si vous n'avez pas encore eu le plaisir d'utiliser Firefox, commencez par télécharger l'installateur depuis le site de Mozilla :

http://www.mozilla-europe.org/fr/products/firefox/

Une fois téléchargé sur votre ordinateur, cliquez sur l'installateur et laissez-vous guider. Le premier écran vous invite à fermer toutes vos applications afin d'éviter d'éventuels conflits de données communes. Le second écran vous invite à préciser si vous désirez importer des paramètres, favoris et autres réglages depuis un autre logiciel déjà installé sur votre ordinateur. Vous pourrez ainsi importer de nombreux paramètres depuis Internet Explorer mais aussi depuis Netscape, Opera ou encore Mozilla selon le navigateur présent sur votre ordinateur au moment de l'installation.

Une fois l'installation terminée, un dernier écran vous demande de l'autoriser à mettre à jour automatiquement les différents modules de Firefox. Nous vous conseillons de laisser cette case cochée, afin de disposer au plus vite de la toute dernière version du navigateur.

Utilisation de Firefox

Firefox regorge de fonctionnalités très intéressantes, mais il serait bien trop long de toutes les détailler dans cet ouvrage. Nous nous contenterons simplement d'indiquer quelques procédures indispensables à connaître pour pouvoir exploiter au mieux Firefox dans le cadre de la mise au point de pages Web.

Les onglets

Précurseur par rapport à son concurrent, Firefox à intégré depuis le début l'affichage des pages dans des onglets différents. Les développeurs ayant souvent de nombreuses pages à consulter en même temps apprécieront d'utliser ce système d'onglets plutôt que d'avoir à jongler avec de multiples fenêtres.

L'ouverture d'un nouvel onglet peut être effectuée depuis le menu (fichier puis nouvel onglet) et cette fonctionnalité sera encore plus efficace si vous utilisez le raccourci clavier en rapport : Ctrl + T. Une autre alternative pour ouvrir un nouvel onglet consiste à appuyer sur la touche Ctrl en même temps qu'un clic sur un lien hypertexte d'une page. La page cible s'ouvrira alors dans un nouvel onglet et ne remplacera pas la page en cours (si vous appuyez sur la touche Maj au lieu de la touche Ctrl, la page s'ouvrira cette fois dans une nouvelle fenêtre Firefox).

À noter que dans cette nouvelle version 2.0, vous pouvez maintenant (sans extension supplémentaire à installer) ranger vos onglets dans l'ordre qui vous convient par un simple glisser-déposer de souris. Lorsque vous être en train de déplacer un onglet avec votre souris, une petite flèche indique alors à quel endroit il va être inséré si vous relâchez le bouton de la souris.

Les marque-pages

Les marque-pages (appelés aussi « favoris » dans Internet Explorer) regroupent les différentes adresses mémorisées lors de vos consultations de pages Web. Pour enregistrer une nouvelle adresse avec Firefox, il suffit de sélectionner depuis le menu Marque-pages>Marquer cette page (ou d'utiliser le raccourci clavier Ctrl + D) puis de choisir le dossier dans lequel vous désirez mémoriser le signet. Si vous désirez que le signet apparaisse dans la barre d'outils, il faudra le placer dans le répertoire Barre personnelle. Enfin, sachez qu'il est possible d'afficher les marque-pages dans le panneau latéral avec le raccourci clavier Ctrl + B.

Gestion du cache

Vous pouvez modifier la taille du cache : il s'agit de l'espace de stockage sur votre machine des éléments déjà consultés, pour un chargement plus rapide. La plupart des utilisations nécessitent entre 50 et 100 Mo de mémoire cache. Attention à ne pas trop augmenter cette valeur, c'est la plupart du temps inutile. Pour modifier la taille du cache depuis le menu de Firefox, allez dans le menu Outils>Option puis cliquez sur Avancé et choisissez l'onglet Réseau. Puis dans cette fenêtre, vous trouverez un champ qui vous permettra de modifier la taille du cache. De même, si vous désirez vider le cache, vous pouvez aussi cliquer sur le bouton Nettoyer maintenant situé à droite du champ de cette même fenêtre.

Extensions Firebug et IE Tab

L'extension Firebug permet de disposer d'outils de suivi du code JavaScript, CSS et DOM d'une page parcourue par Firefox. Cette extension permet aussi d'accéder aux données XML ainsi qu'aux requêtes XMLHttpRequest ce qui va se révéler fort pratique dans la cadre du débogage d'une application Ajax.

Voici un aperçu de ses principales fonctionnalités :

- Débogage JavaScript grâce à ses fonctions de suivi de programme et de gestion de points d'arrêts qui permettent d'avancer pas à pas pour développer et mettre au point ses scripts très rapidement.
- Analyse des accès réseau très intéressant pour connaître les données échangées et les temps de réponse des requêtes HTTP propres à Ajax.
- Inspection d'un élément HTML, XML ou CSS de la page par simple survol avec possibilité de modifier son code à la volée.
- Exploration par le clavier ou la souris des éléments les plus reculés du DOM.

L'extension IE Tab permet d'émuler le fonctionnement du navigateur Internet Explorer directement dans Firefox. Pour basculer d'un navigateur à l'autre, il suffit de cliquer sur le bouton situé en bas à droite de l'écran.

Installation des extensions

Pour installer une nouvelle extension Firefox, vous pouvez vous rendre directement sur la page Web du site officiel de Mozilla :

https://addons.mozilla.org/firefox/

Une autre alternative consiste à allez dans le menu Outils>Modules complémentaires. La fenêtre de gestion des modules complémentaires doit alors s'ouvrir (voir figure 5-1). Cliquez ensuite en bas à droite de cette fenêtre sur le lien Obtenir des extensions pour afficher la page des extensions de Firefox.

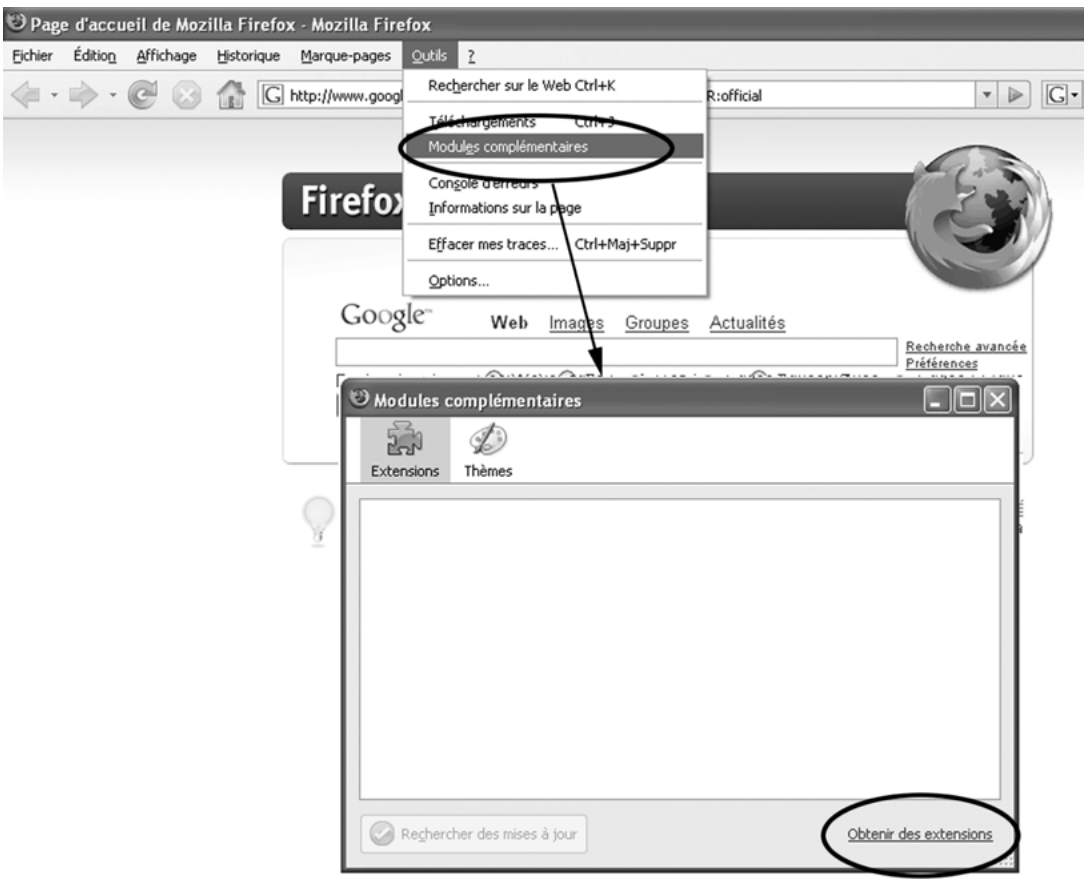


Figure 5-1

Accès à la fenêtre de gestion des extensions Firefox

Une fois dans la page des extensions, saisissez le nom de l'extension recherchée dans le champ prévu à cet effet placé en haut et à droite de l'écran. Cliquez ensuite sur le bouton Hop ! pour lancer la recherche.

Après avoir localisé la page correspondante à l'extension Firebug, cliquez sur le bouton de téléchargement Installer puis sur le bouton Installer maintenant de la fenêtre d'installation du logiciel.

Une fois l'installation effectuée, l'extension vient alors se placer dans la fenêtre des modules complémentaires dans la rubrique Installation. Un bouton placé en bas de cette fenêtre vous invite alors à redémarrer Firefox pour que l'extension soit complètement opérationnelle. Dans notre cas, comme nous désirons installer une autre extension, nous allons différer ce redémarrage après l'installation de la seconde extension.

Retournons donc sur la page d'accueil des extensions Firefox pour lancer une seconde recherche en saisissant cette fois le nom de l'extension IE Tab.

La procédure sera la même que pour l'extension précédente, cliquez sur les boutons de téléchargement pour installer la seconde extension sur l'ordinateur et pour qu'elle soit visible dans la fenêtre des modules complémentaires.

Cette fois, nous pouvons cliquez sur le bouton de redémarrage en bas de la fenêtre du module complémentaire afin d'installer complètement les deux extensions dans Firefox (voir figure 5-2).

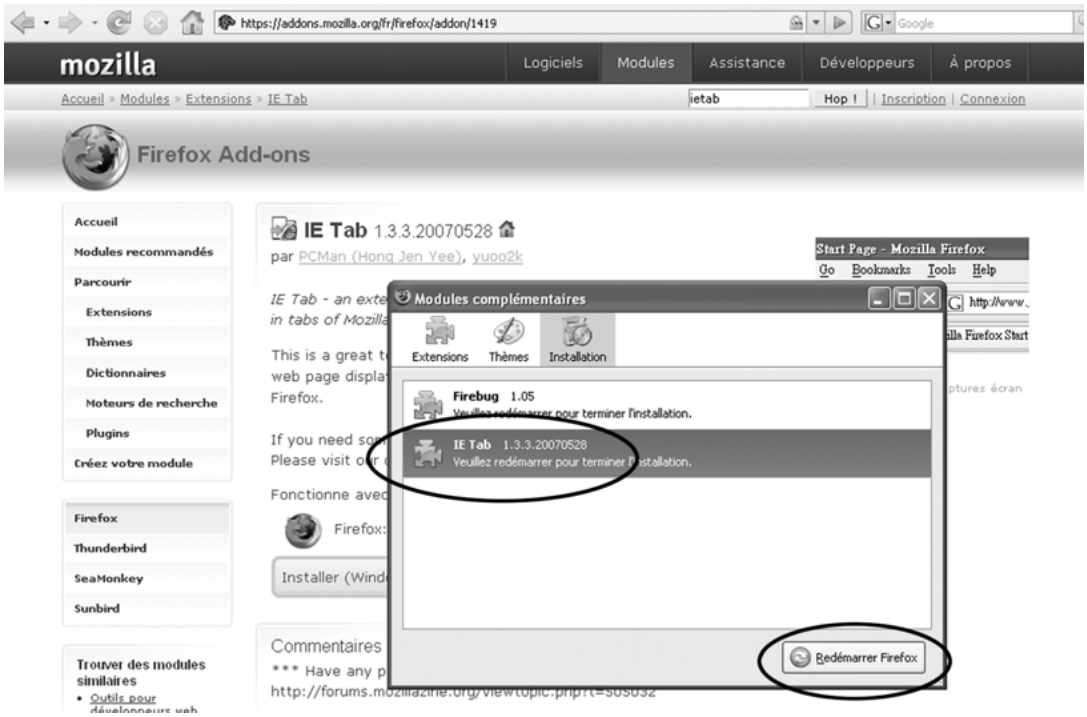


Figure 5-2

Fenêtre de gestion des extensions Firefox après l'ajout des deux modules complémentaires.

Lorsque Firefox s'ouvre de nouveau, il dispose alors des fonctionnalités de ses nouvelles extensions. Vous pouvez vous en assurer en déroulant le menu Outils, par la présence de deux nouvelles options correspondantes aux extensions précédemment installées (voir repères 1 et 2 de la figure 5-3).

De même deux nouveaux boutons sont venus prendre place en bas à droite de l'écran du navigateur (voir figure 5-3). Le premier bouton permettra d'ouvrir Firebug et le second de basculer de Firefox à l'émulateur de IE (voir repère 3 de la figure 5-3).

> **Utilisation des extensions**
>
> Pour l'utilisation des extensions Firebug et IE Tab, nous vous invitons à vous référer aux chapitres de la partie 3 dans lesquels nous illustrons par des cas pratiques comment exploiter leurs différentes fonctionnalités.

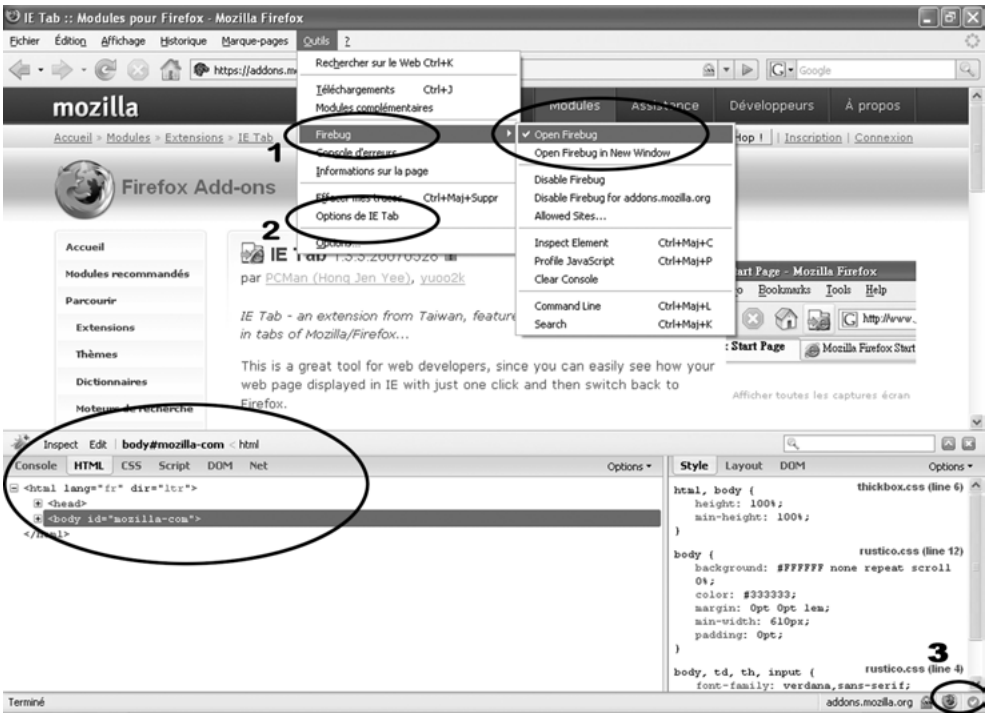


Figure 5-3

Menu et boutons de contrôle de Firebug et de IE Tab

6

Wamp5, une infrastructure serveur complète

Choix de l'infrastructure serveur

Contrairement à une simple page HTML (même si elle contient du code JavaScript traditionnel), une page Ajax devra en général communiquer avec des scripts serveur (PHP, par exemple). Cette page pourra éventuellement avoir des accès à une base de données (MySQL, par exemple). Or, l'utilisation de ces technologies nécessite une infrastructure serveur adéquate car plusieurs applications sont nécessaires à leur fonctionnement côté serveur :

- un serveur Web (le serveur Apache est le plus fréquemment utilisé) ;
- un langage de script serveur installé sur le serveur Web (dans cet ouvrage, nous utiliserons PHP) ;
- un serveur de base de données (dans cet ouvrage, nous utiliserons MySQL).

Selon les ressources matérielles dont vous disposez, plusieurs solutions peuvent être exploitées.

La première solution concerne les développeurs qui disposent d'une connexion permanente et rapide à Internet ainsi qu'un serveur Web distant équipé d'une base de données MySQL et d'un moteur de scripts PHP.

La deuxième solution est la plus exigeante. Elle concerne surtout les sociétés de développement Internet qui ont à leur disposition un serveur Web en local, avec PHP et MySQL, en plus de leur serveur distant de production.

La troisième solution est accessible à tous, puisqu'il suffit d'installer sur son poste de développement une infrastructure serveur avec PHP et MySQL qui recrée en local le même comportement que le serveur Web distant.

Nous avons retenu la troisième solution pour réaliser nos démonstrations car elle pourra être utilisée par tous les lecteurs de cet ouvrage. Cependant, les concepts développés sont identiques quelle que soit la méthode retenue.

Afin de vous accompagner dans la mise en œuvre de votre plate-forme de développement, le paragraphe suivant sera consacré à l'installation d'une infrastructure serveur locale.

Les protocoles Internet

Internet permet de relier de nombreux ordinateurs distants par un support physique. Cependant, pour que ces derniers puissent dialoguer, ils doivent utiliser un même protocole. Le protocole pour le Web d'Internet est le HTTP qui permet aux internautes de consulter (ou d'évaluer dans notre cas) des pages Web à l'aide de leur navigateur. Il existe également d'autres protocoles dédiés à des médias spécifiques ou permettant d'accéder à des services en ligne. Ainsi, les protocoles SMTP et POP3 permettent de gérer les e-mails et le protocole FTP permet le transfert (ou la publication dans notre cas) de fichiers d'un ordinateur à l'autre.

Mise en œuvre d'une infrastructure serveur

Procédure d'installation de la suite Wamp5

Pour télécharger gratuitement la dernière version de Wamp5, consultez le site *www.wampserver.com*. Cliquez sur le lien `Downloads`, puis remplissez le formulaire d'informations (voir figure 6-1). Vous serez ensuite redirigé vers la page du site Sourceforge.net dédiée à Wamp5 où vous pourrez télécharger le fichier exécutable sur votre ordinateur.

http://www.wampserver.com

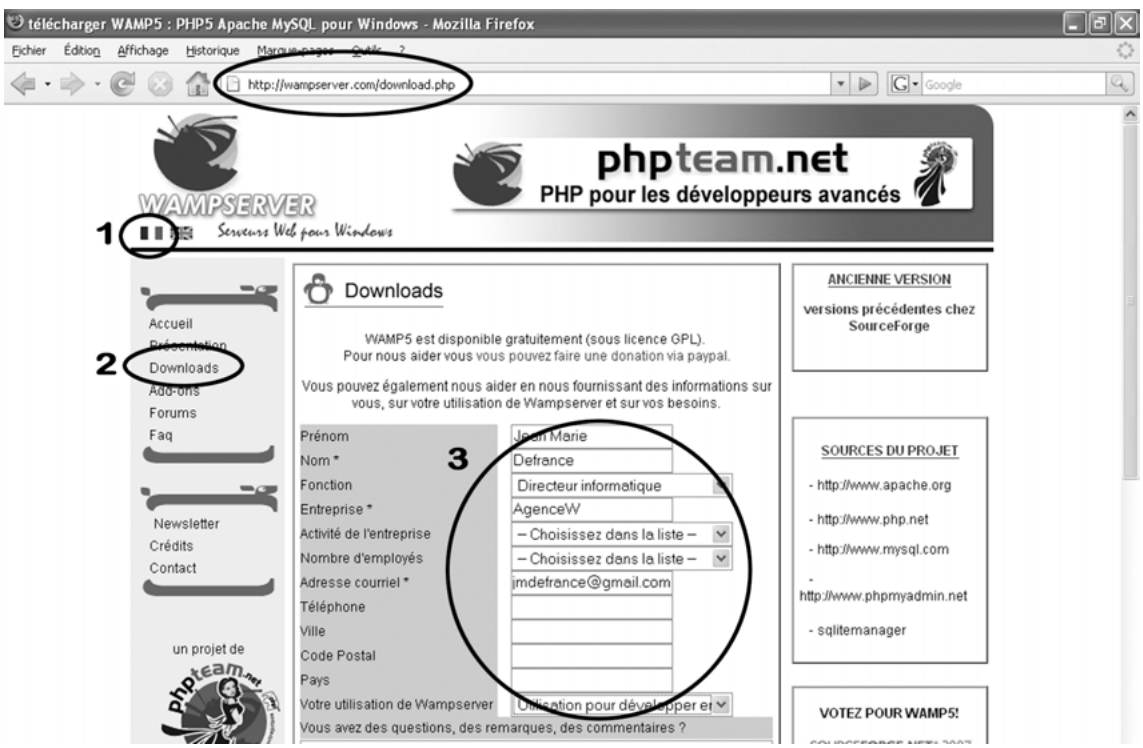


Figure 6-1

Installation de Wamp5 : cliquez sur le lien Downloads et complétez le formulaire d'information.

Une fois le fichier enregistré sur votre ordinateur, lancez l'installation en double-cliquant sur l'installeur. Une première fenêtre apparaît, vous recommandant de fermer toutes les applications actives avant de lancer l'installation. Cliquez ensuite sur `Next` pour faire apparaître les conditions d'utilisation (licence) qu'il faut valider. Dans l'écran suivant, vous pouvez choisir le répertoire dans lequel vous allez installer le logiciel. Nous vous suggérons de valider l'option par défaut (`C:\wamp`). Dans l'écran qui suit, choisissez le dossier de programme dans lequel le logiciel apparaîtra (nous vous recommandons de choisir celui suggéré par Wamp5, soit `WampServer`). L'écran suivant, récapitule les options choisies et vous invite à lancer l'installation en cliquant sur le bouton `Install`.

L'installation démarre et un indicateur affiche l'état d'avancement de la tâche. Vous aurez ensuite à choisir le répertoire dans lequel seront stockés les fichiers PHP (conserver l'option par défaut : `www`). Vous devez également renseigner un certain nombre d'informations concernant votre messagerie, à savoir le SMTP (prendre le serveur de messagerie sortant de votre fournisseur d'accès Internet : `smtp.wanadoo.fr`, par exemple) et votre e-mail par défaut. Une fenêtre s'affiche alors pour vous informer que Firefox a été détecté sur votre ordinateur et vous invite à le choisir comme navigateur par défaut. Validez cette option, afin de vous permettre d'utiliser Firefox et ses extensions précédemment installées (Firebug et IE Tab) pour mettre au point vos pages dynamiques. Enfin, un dernier écran indique que l'installation de Wamp5 s'est correctement déroulée et vous propose de démarrer le logiciel dès maintenant (case précochée).

Après validation du dernier écran, Wamp5 démarre automatiquement et une icône apparaît dans la barre des tâches de votre ordinateur. Il existe trois états possibles de cette icône : si elle est complètement blanche, cela signifie que les deux serveurs (le serveur Apache et MySQL) sont en état de marche ; si les deux premiers tiers du demi-cercle sont jaune, cela signifie qu'au moins un des deux serveurs est arrêté (ou pas encore démarré) ; enfin, si le premier tiers du demi-cercle est rouge, cela signifie que les deux serveurs sont à l'arrêt.

Arrêt et démarrage de Wamp5

Avant d'utiliser Wamp5, il est utile de rappeler la procédure de gestion des serveurs et du logiciel pour vos futures utilisations. Pour commencer, je vous invite à arrêter les serveurs de Wamp5. Pour cela, cliquez sur l'icône de Wamp5, puis dans le menu contextuel qui s'affiche (par la suite nous appellerons ce menu contextuel, le manager de Wamp5), cliquez ensuite sur `Stop All Services` (voir figure 6-2). L'icône doit alors changer d'état et apparaître avec son premier tiers rouge. Pour redémarrer les serveurs de Wamp5, vous devez parcourir le manager de ce dernier et sélectionnez l'option `Start All Services`. À noter que si l'icône de Wamp5 reste au jaune ou au rouge, cela indique que vos serveurs (ou l'un de vos serveurs) ne sont plus opérationnels. Il faudra alors accéder au manager et sélectionner l'option `Restart All Services` pour réactiver le ou les serveurs de Wamp5. Nous venons de voir la procédure pour gérer l'arrêt et le redémarrage des serveurs de Wamp5. Cependant, si vous désirez complètement arrêter l'application, il faut dans ce cas faire un clic droit sur la même icône et sélectionner `Exit` (l'icône doit alors disparaître complètement de la barre des tâches). Pour relancer le logiciel Wamp5 il faut alors dérouler le menu `Tous les programmes` du bouton `Démarrer` de Windows, puis le dossier `WampServer` et cliquer sur l'icône de `start wampserver`. La même démarche devra d'ailleurs être effectuée lors du démarrage de votre ordinateur pour

lancer Wamp5, sauf si vous avez coché la case de démarrage automatique dans la procédure d'installation que nous venons de détailler.

Figure 6-2

Utilisation de Wamp5 : dès le démarrage du logiciel Wamp5, une icône apparaît dans la barre des tâches. Cliquez sur cette icône pour afficher le manager de Wamp5.

Découverte du manager de Wamp5

Le manager de Wamp5 vous permet d'accéder aux fonctions suivantes (les différentes options seront détaillées en partant du haut du manager, voir figure 6-2) :

- **Locahost** – donne accès au Web local et permet de tester toutes les pages enregistrées sous la racine `www` (soit `http://localhost/` qui correspond à la racine située à l'emplacement `C:/wamp/www/`).
- **PhpMyAdmin** – permet d'accéder au gestionnaire de base de données MySQL nommé phpMyAdmin (soit l'alias `http://localhost/phpmyadmin/`).
- **SQLiteManager** – permet d'accéder au gestionnaire de base de données intégré à PHP nommé SQLite (soit l'alias `http://localhost/sqlitemanager/`).
- **www directory** – donne accès à un explorateur Windows configuré pour s'ouvrir automatiquement dans le répertoire racine `www` (`C:\wamp\www\`).
- **Log files** – permet d'afficher les fichiers de log du serveur Apache (`apache error log`), du préprocesseur PHP (`php error log`) et du serveur MySQL (`mysql error log`). Ces fichiers de log pourront être avantageusement consultés pour identifier un problème dans l'une des applications citées.
- **Config files** – ouvre les différents fichiers de configuration de la suite Wamp5 dans le Bloc-notes. Ces fichiers sont : `httpd.conf` pour la configuration d'Apache, `php.ini` pour la configuration de PHP et `my.ini` pour la configuration de MySQL. Vous pouvez ainsi consulter, voire modifier (à ne faire évidemment que si vous êtes sûr de vous…) les options de chaque fichier de configuration de la suite Wamp5.
- **PHP extensions** – permet d'activer ou de désactiver une extension PHP. Les extensions de PHP sont des bibliothèques de fonctions dédiées à un usage particulier qu'il faut activer sur le serveur avant de pouvoir les exploiter dans vos programmes PHP. Certaines de ces extensions sont activées par défaut (gestion des bases de données MySQL ou SQLite…) alors que d'autres doivent être activées manuellement avant leur usage (gestion des fichiers PDF ou encore XSL…). Pour activer une extension, il suffit de double-cliquer sur l'extension désirée dans la liste proposée par le manager.

- **Alias directories** – permet de créer des répertoires alias pointant vers des ressources placées dans un emplacement différent de la racine www (`C:/wamp/www/` par défaut). Par exemple, si vous désirez créer un alias pointant sur des fichiers PHP placés dans le répertoire `D:/www/SITEdemo`, il suffit de créer un alias `/SITEdemo/` ainsi, si vous appelez l'URL `http://localhost/SITEdemo/` dans votre navigateur, vous accéderez aux fichiers situés dans le répertoire `D:/www/SITEdemo` et non ceux du répertoire `C:/wamp/www/SITE-demo/` qui n'existe pas en réalité.
- **Apache** – donne accès à un sous-menu d'administration du serveur Apache. Vous pourrez ainsi arrêter le serveur (`Stop Service`) et le redémarrer (`Restart Service`). Ce sous menu vous permet aussi de désinstaller puis d'installer un autre serveur Apache d'une version différente.
- **MySQL** – donne accès à un sous-menu d'administration du serveur MySQL. Vous pourrez ainsi arrêter le serveur (`Stop Service`) et le redémarrer (`Restart Service`). Ce sous-menu vous permet aussi de désinstaller puis d'installer un autre serveur MySQL d'une version différente.
- **Start All Service** – permet de démarrer tous les services en même temps (soient les deux serveurs, Apache et MySQL).
- **Stop All Service** – permet d'arrêter tous les services en même temps (soient les deux serveurs, Apache et MySQL).
- **Restart All Service** – permet de redémarrer tous les services en même temps (soient les deux serveurs, Apache et MySQL).

D'une version à l'autre

Selon la version de Wamp5 et de votre système d'exploitation, les écrans et les procédures détaillées précédemment peuvent être très légèrement différents. En guise de référence, nous avons utilisé la version 1.7.2 de Wamp5 pour nos démonstrations. Si vous utilisez une version antérieure ou ultérieure, il est possible que le manager soit organisé différemment. Il n'en demeure pas moins que le fonctionnement de ces logiciels reste identique d'une version à l'autre et que vous n'aurez pas de difficulté à adapter les procédures détaillées dans cet ouvrage.

Test du serveur local

Pour tester le bon fonctionnement du serveur Web et du moteur PHP, nous allons commencer par créer un script PHP à l'aide d'un simple éditeur de texte. Ouvrez le Bloc-notes de Windows à partir du menu `Démarrer` (`Programmes>Accessoires>Bloc-notes`) ou TextEdit si vous utilisez un Macintosh. Saisissez ensuite les trois lignes de code suivantes dans l'éditeur :

```
<?php
echo "Bonjour, PHP fonctionne" ;
?>
```

Enregistrez ensuite ce fichier dans `C:\wamp\www\SITEajax` sous le nom `bonjour.php`, en prenant soin de sélectionner le type `Tous fichiers` et en ajoutant au nom du fichier l'extension `.php`. Le répertoire `SITEajax` sera créé sous `www` lors de l'enregistrement de ce premier fichier. Ce même répertoire sera utilisé dans les chapitres suivants pour tester les exemples d'applications Ajax qui entreront en interactions avec des fichiers PHP, c'est

pourquoi nous vous conseillons d'utiliser les mêmes conventions de nommage. De retour dans le Bloc-notes, assurez-vous que le nom du fichier apparaît bien dans la barre de titre de la fenêtre puis fermez le Bloc-notes.

Ne jamais supprimer le fichier index.php de la racine www

La page Web local qui s'affiche quand vous accédez au localhost par le manager de Wamp5, n'est autre que le fichier `index.php` qui se trouve à la racine `www`. Si vous tenez à conserver la page qui affiche les différents répertoires de vos sites, il faudra veiller à ne pas supprimer ce fichier. Enfin, côté organisation, nous vous conseillons de créer un répertoire différent sur cette même racine à chaque fois que vous ajouterez un nouveau site sur votre serveur local. Ainsi, vous pourrez accéder à vos différents sites très facilement depuis la page du Web local en cliquant sur le dossier correspondant dans la rubrique Vos projets.

Ouvrez maintenant la page `Web Local` à partir du manager de Wamp5 (option `localhost` du manager, voir repère 1 de la figure 6-3). Le répertoire `SITEajax` doit alors apparaître en bas de cette page, dans une rubrique nommée `Vos projets`. Cliquez sur le lien `SITEajax` (voir repère 2 de la figure 6-3) pour ouvrir une fenêtre qui affiche la liste de tous les fichiers contenus dans ce répertoire : dans le cas présent, nous retrouvons uniquement notre fichier `bonjour.php` (voir figure 6-3).

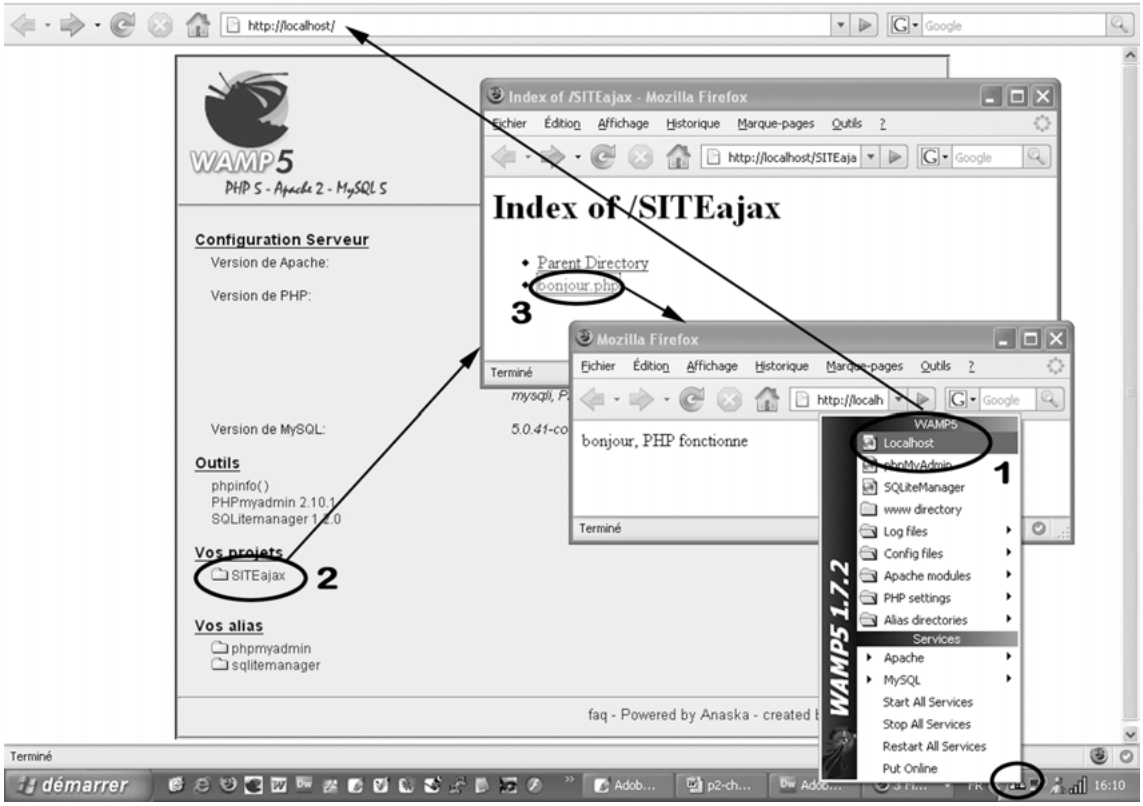


Figure 6-3

La page Web local (option localhost du manager Wamp5) permet d'accéder au répertoire SITEajax, puis d'ouvrir le fichier bonjour.php dans le navigateur.

Si vous cliquez maintenant sur le fichier `bonjour.php` (voir repère 3 de la figure 6-3), vous envoyez une requête au serveur Apache pour ouvrir le fichier dans le navigateur. Si le serveur Web et le moteur PHP fonctionnent correctement, le message `Bonjour, PHP fonctionne` doit s'afficher dans le navigateur (voir figure 6-3). Il est en outre intéressant d'observer le code source envoyé au navigateur en cliquant sur `Source` dans le menu `Affichage`. On remarque que le code ne comporte plus les balises PHP ni l'instruction `echo` saisies lors de la création du fichier, mais uniquement le message affiché dans le navigateur. En effet, lors de l'appel du fichier, celui-ci est d'abord exécuté par le moteur PHP du serveur Apache et c'est la page résultante en HTML qui est ensuite envoyée au navigateur pour son interprétation finale.

Configuration du fichier php.ini

Le fichier `php.ini` permet de configurer de nombreux paramètres et options d'exécution de PHP. Ce fichier est lu à chaque démarrage du serveur Apache, il suffit donc de redémarrer le serveur Apache pour que les nouvelles options soient prises en compte. Pour vos premiers tests, nous vous conseillons de l'utiliser avec ses options par défaut mais, par la suite, vous pourrez facilement le modifier à l'aide du manager de Wamp5. Pour accéder à ce fichier, il suffit de cliquer sur l'option `Config files` du manager de Wamp5 puis de sélectionner `php.ini`. Une fois ce fichier ouvert dans le Bloc-notes, vous découvrirez un grand nombre de paramètres accompagnés de nombreux commentaires qui vous guideront dans leur configuration. Parmi ces paramètres, nous avons choisi de vous en présenter trois, dont il conviendra de vérifier leur configuration :

- `magic_quote_gpc` : s'il est initialisé avec la valeur `On`, ce paramètre permet de préfixer automatiquement par une barre oblique inverse (`\`) les apostrophes et les guillemets d'un texte envoyé par un formulaire ou issu d'un cookie avant de l'enregistrer dans la base MySQL. Il évite d'avoir à utiliser les fonctions `addSlashes()` et `stripSlashes()` à chaque insertion. Cependant, cette option est maintenant déconseillée car elle nécessite de mettre en place une gestion différente des données selon leur origine et entraîne une légère baisse des performances du système. Vous pouvez cependant l'activer pour assurer la compatibilité avec d'anciens scripts.
- `register_globals` : s'il est initialisé avec la valeur `On`, ce paramètre permet d'utiliser les variables globales (variables simples comme `$var1`) lors d'un passage d'une variable d'une page à l'autre (`GET`) ou d'une récupération de valeur d'un champ de formulaire (`GET` ou `POST`). Cette option est configurée par défaut à `Off` depuis la version 4.2 de PHP, ce qui contraint à utiliser les tableaux des variables serveur (`$_POST['var1']`, `$_GET['var1']`...). Vous pouvez configurer ce paramètre à `On` si vous utilisez des anciens scripts et que vous ne souhaitez pas les modifier. Cependant, nous vous conseillons vivement de laisser sa valeur à `Off` si vous développez de nouveaux scripts afin qu'ils soient exploitables quelle que soit la version du PHP.
- `error_reporting` : cette option peut être paramétrée selon le niveau de contrôle désiré de vos scripts. Dans les dernières versions de PHP, cette option est configurée par défaut avec la valeur `E_ALL` qui est le niveau maximal de contrôle. Avec ce paramétrage, toutes variables non initialisées provoqueront automatiquement un *warning* (une alerte) (`Undefined variable`). Si vous désirez éviter ces fréquents warning, vous pouvez remplacer la valeur actuelle par `E_ALL & ~ E_NOTICE` mais l'idéal est bien sûr de programmer proprement et d'initialiser toutes les variables avant de les utiliser.

Gestion des extensions PHP

Extensions installées par défaut

Les extensions PHP sont des bibliothèques de fonctions dédiées à une utilisation spécifique. Il existe par exemple des extensions dédiées à MySQL (`php_mysql`), à la gestion des images (`php_gd2`) ou encore aux fonctions XML (`php_domxml`).

Lorsque vous avez installé Wamp5, certaines extensions PHP ont été installées par défaut et sont donc immédiatement disponibles (`php_mysql` par exemple) mais il est aussi très simple d'en installer d'autres par le biais du manager de Wamp5.

Installation d'une extension

L'installation d'une extension sur Wamp5 est très simple. Depuis le manager de Wamp5, sélectionnez `PHP settings` (voir repère 2 de la figure 6-4) puis `PHP extensions` (voir repère 3 de la figure 6-4) et recherchez l'extension à installer dans la liste (voir repère 4 de la figure 6-4). Si le nom de l'extension n'est pas précédé d'une petite flèche, cela signifie que l'extension n'est pas encore installée. Cliquez alors sur le nom de l'extension pour l'activer. Redémarrez ensuite Wamp5 en cliquant sur `Restart All services` depuis le manager. Vous pouvez ensuite réafficher cette liste d'extension afin de vous assurer que l'extension qui vient d'être installée est maintenant précédée d'une petite flèche.

Si vous êtes sur un serveur distant, sachez qu'il est possible de vérifier la présence d'une extension en affichant un fichier `phpinfo.php` (fichier contenant une fonction `phpinfo()`) et en recherchant l'entrée correspondant à l'extension dans les tableaux de cette page. À noter que sur Wamp5, un fichier `phpinfo` est directement disponible depuis la page `localhost` accessible depuis l'entrée du même nom dans le manager.

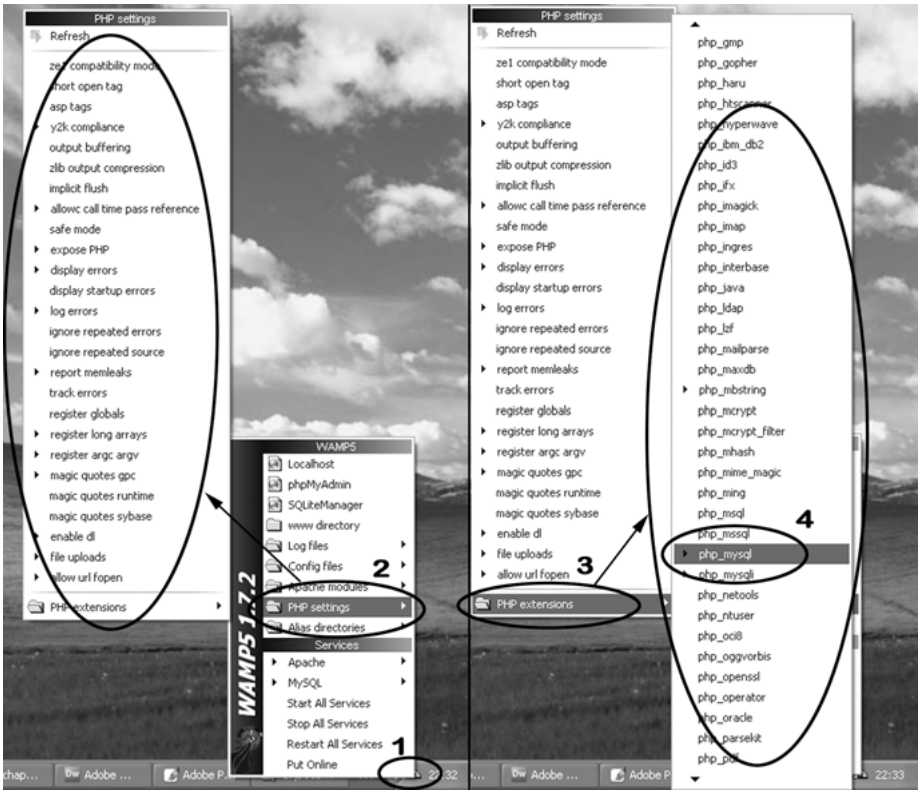


Figure 6-4

Pour installer une extension, affichez la liste des extensions à partir de l'entrée PHP extension du manager, puis cliquez sur l'extension à installer.

Gestionnaire phpMyAdmin

La suite Wamp5 comprend également une base de données MySQL et son gestionnaire phpMyAdmin. Pour accéder au gestionnaire phpMyAdmin et administrer la base de données, il faut cliquer sur l'entrée de même nom dans le menu du manager (voir figure 6-5).

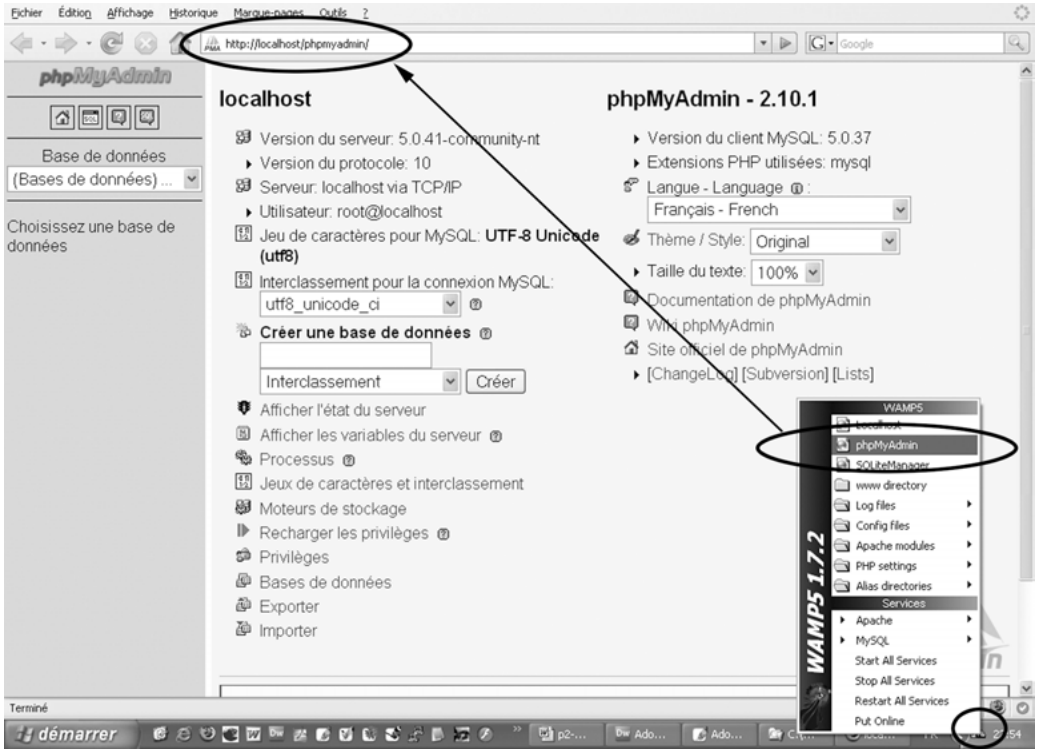


Figure 6-5

Pour accéder au gestionnaire de la base de données MySQL, il faut cliquer sur l'entrée phpMyAdmin du menu du manager de Wamp5.

L'interface du gestionnaire phpMyAdmin s'affiche alors dans le navigateur (voir figure 6-5). Les différentes fonctionnalités de ce gestionnaire seront présentées en détail dans le chapitre 22 dédié à MySQL à la fin de cet ouvrage et dans les ateliers du chapitre 13.

7

Dreamweaver, un éditeur polyvalent

Pourquoi utiliser Dreamweaver ?

Dans le chapitre consacré à l'environnement serveur, vous avez créé un petit script pour tester le fonctionnement du serveur (revoir le fichier `bonjour.php` dans le chapitre 6). Pour créer ce premier fichier PHP, vous avez utilisé le bloc-notes de votre PC (ou TextEdit si vous avez un Macintosh). Par la suite, nous allons devoir créer des pages HTML, mais surtout de nombreux programmes PHP ou JavaScript. Même s'il est possible de créer tous vos scripts avec un simple bloc-notes, il est très intéressant d'utiliser un éditeur plus évolué et surtout mieux adapté à la création de scripts PHP ou JavaScript (avec des fonctions de coloration syntaxique ou des assistants à la saisie de code par exemple).

Il existe de nombreux éditeurs polyvalents qui pourraient vous assister dans la création de pages HTML et dans la rédaction de programmes PHP et JavaScript, mais nous avons choisi l'éditeur Dreamweaver car c'est l'éditeur Web le plus utilisé actuellement et que vous êtes déjà très nombreux à être initié à son usage ce qui facilitera d'autant plus la création de vos applications Ajax-PHP.

Sachez évidemment, que si vous optez pour un autre éditeur, cela ne change en rien la lecture de ce livre car vous pourrez créer de la même manière tous les scripts des applications Ajax-PHP avec votre éditeur préféré.

Pour ceux qui n'ont jamais utilisé Dreamweaver, nous vous proposons de découvrir son interface et les principales fonctions de son éditeur de code. Les utilisateurs de Dreamweaver, quant à eux, peuvent passer directement au chapitre suivant. Nous leur conseillons quand même de lire la partie sur la définition d'un site et notamment celle concernant la configuration du serveur d'évaluation pour disposer de la fonctionnalité d'aperçu des pages. Cette dernière est bien pratique pour les ateliers qui vont suivre.

Enfin, pour cette démonstration, nous utiliserons la version CS3 mais soyez rassurés, les fonctionnalités que nous utilisons sont semblables dans les versions antérieures et vous pourrez facilement adapter ces présentations à la version de votre Dreamweaver.

Présentation de l'interface de Dreamweaver

L'interface de Dreamweaver CS3 est composée d'une fenêtre Document (voir figure 7-1, repère 1) en haut de laquelle on trouve la barre d'outils documents comprenant, entre autres, trois boutons (voir figure 7-1, repère 2) qui permettent de sélectionner le mode (mode Code, Fractionner ou mode Mixte et mode Création). Une barre d'outils nommée Insérer (voir figure 7-1, repère 3) permet d'accéder à plusieurs catégories de boutons classés par thèmes, à partir d'onglets. De nombreux panneaux disposés à droite et en bas de la fenêtre de document permettent de paramétrer les éléments de la page ou d'accéder aux fonctionnalités de l'éditeur. Le panneau Propriétés (voir figure 7-1, repère 4) affiche les caractéristiques de l'élément sélectionné dans la page. Le panneau Fichiers (voir figure 7-1, repère 5) permet d'ouvrir les différents fichiers du site.

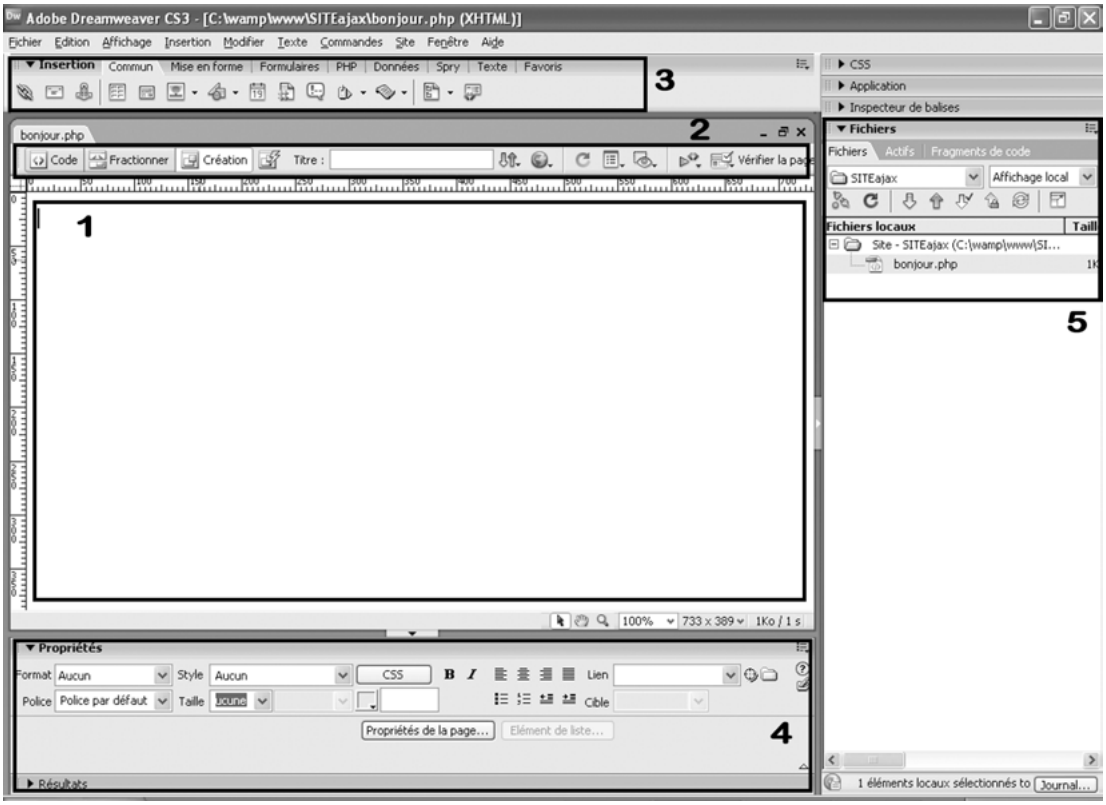


Figure 7-1

L'interface de Dreamweaver CS3 et ses principaux panneaux

Définition d'un site

Avant de créer un fichier avec Dreamweaver, il est fortement recommandé de définir la configuration du site. Nous allons configurer Dreamweaver avec le répertoire racine de Wamp5 afin que nos fichiers PHP et JavaScript soient accessibles depuis le serveur Web Local.

Depuis le menu, sélectionnez Site puis Gérer les sites. Cette fenêtre Gérer les sites permet de créer, modifier ou supprimer un site, par la suite vous pourrez ainsi afficher cette même fenêtre pour apporter des modifications à la configuration initiale d'un site. À noter que pour créer un site rapidement, vous auriez aussi pu sélectionner le menu Site puis Nouveau site pour arriver directement à la fenêtre Définition d'un site que nous allons vous présenter ci-après. Dans la fenêtre Gérer les sites, cliquez sur le bouton Nouveau puis sélectionnez Site. La fenêtre Définition d'un site s'affiche à l'écran (voir figure 7-2). Pour définir un nouveau site, vous pouvez utiliser l'assistant (onglet Elémentaire) ou le mode Avancé (onglet Avancé). Cliquez sur l'onglet Avancé car nous allons saisir nos paramètres sans utiliser l'assistant de configuration.

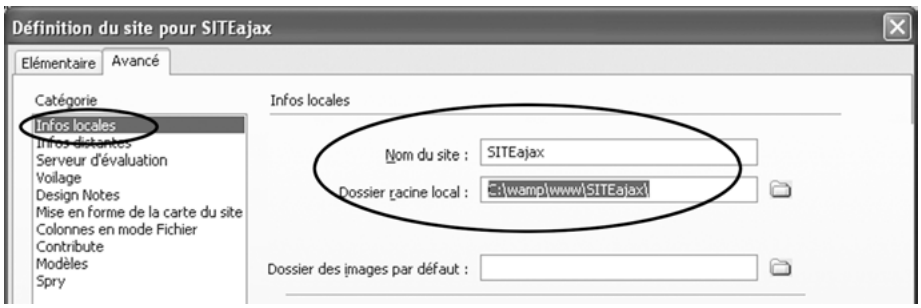


Figure 7-2

La catégorie Infos locales de la fenêtre Définition du site permet de définir le nom du site et le dossier racine sous lequel seront stockés les fichiers du projet.

Informations locales

La première catégorie sélectionnée affiche la page dédiée aux informations locales (voir figure 7-2). Cette fenêtre contient de nombreux champs mais seuls les deux premiers pourront être renseignés pour une configuration minimale d'un site.

- Le premier champ permet de renseigner le nom du site, soit `SITEajax` dans notre exemple.
- Le second champ permet d'indiquer le chemin vers le répertoire dans lequel seront stockés les fichiers du site. Dans notre exemple, nous sélectionnerons le chemin qui correspond au répertoire `SITEajax` déjà créé lors de l'installation de l'infrastructure serveur :

```
C:\wamp\www\SITEajax\
```

En dessous de ces deux champs, d'autres options peuvent être configurées. Dans le cadre de cet ouvrage, il n'est pas nécessaire de configurer ces options mais nous détaillerons cependant leur usage ci-dessous :

- Un dossier spécial permet de stocker les fichiers images : il est d'usage de séparer les médias dans la structure d'un site Internet ; les fichiers HTML, images, sons, vidéos, etc., sont toujours enregistrés dans des répertoires différents.
- Un champ Adresse HTTP permet d'indiquer l'URL pour laquelle votre site sera consultable en ligne. Ainsi, Dreamweaver peut vérifier la validité des hyperliens que vous avez intégrés dans le site.

- Une case à cocher Cache permet d'activer la mémorisation des informations du site afin d'accélérer les différents traitements de l'éditeur. Vous pouvez cocher cette option, mais elle sera surtout indispensable pour les sites de grande envergure.

Informations distantes

Vous pouvez aussi sélectionner la catégorie `Infos distantes` dans le cadre de gauche (voir figure 7-3). Elle permet de configurer les paramètres pour transférer vos fichiers sur un serveur distant. Dans le cadre de cet ouvrage, vous utiliserez uniquement le serveur local précédemment installé avec Wamp5 et vous n'aurez pas à utiliser de serveur distant. Cependant, nous détaillons la procédure de paramétrage de cette catégorie afin que vous puissiez par la suite transférer vos applications sur un serveur de production.

Dans la partie de droite, sélectionnez l'option FTP dans le menu déroulant. Saisissez ensuite les paramètres de votre compte FTP dans les champs appropriés de la fenêtre. Les informations concernant l'hôte FTP, le répertoire de l'hôte, le nom de l'utilisateur et le mot de passe doivent vous être fournis par votre hébergeur (à titre d'illustration, vous trouverez dans le tableau 7-1 des exemples de paramètres possibles pour un compte FTP).

Tableau 7-1 Description des paramètres d'un compte FTP et exemples de configuration (ces paramètres doivent vous être communiqués par votre hébergeur)

Adresse Internet (nom ou IP) du serveur FTP	ftp.monsite.com (ou 213.185.36.111)
Répertoire distant dans lequel vous devez télécharger vos fichiers. (champ optionnel chez certains hébergeurs)	/web/ /httpdocs/ ou encore /html/...
Nom du compte FTP	ajax
Mot de passe FTP	1234

Au terme de votre saisie, vous pouvez vérifier l'exactitude de vos informations en cliquant sur le bouton Test. Cliquez ensuite sur `OK` puis sur le bouton Terminer de la fenêtre Modifier les sites pour confirmer vos modifications.

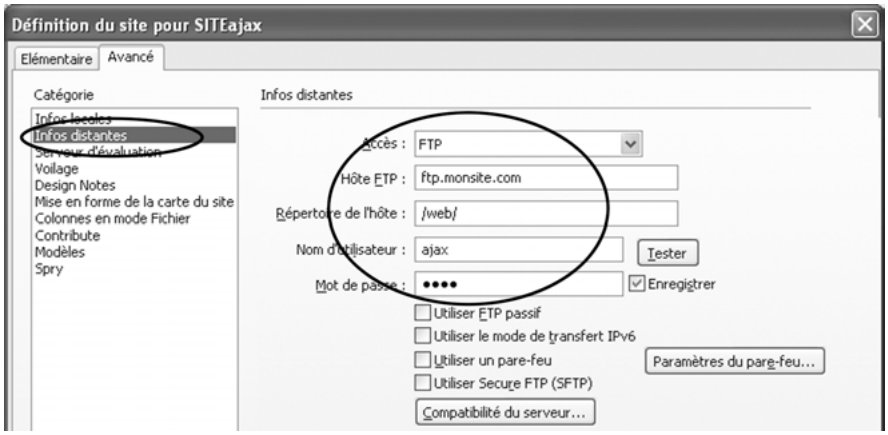


Figure 7-3

La catégorie Infos distantes de la fenêtre Définition du site permet de définir les différents paramètres pour la publication de votre site sur le serveur distant en FTP.

Dans la partie inférieure de la fenêtre, d'autres options peuvent également être configurées. Selon les particularités de votre connexion à Internet ou de votre pare-feu, vous pouvez utiliser l'option FTP passif ou indiquer les paramètres de votre pare-feu dans la fenêtre Préférences de Dreamweaver (si votre réseau local est équipé d'un pare-feu). L'option SSH permet de vous connecter en mode sécurisé codé si votre site distant a été configuré en conséquence.

Serveur d'évaluation

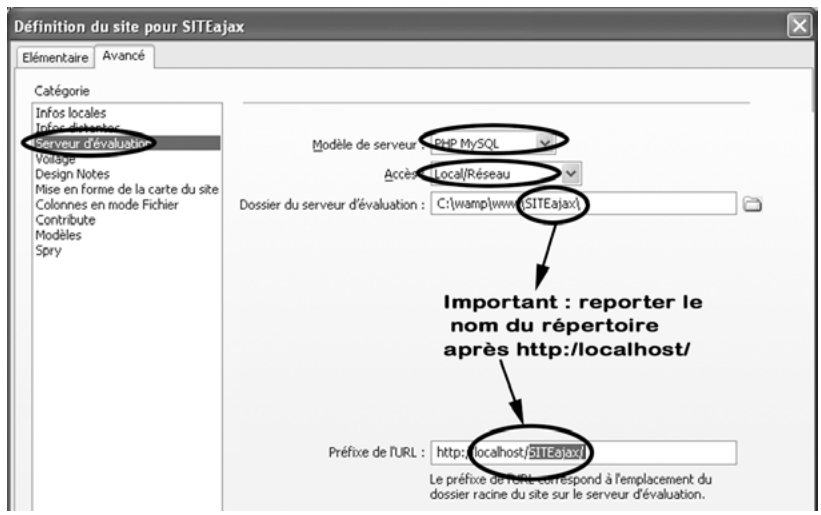
La page de la catégorie `Serveur d'évaluation` regroupe les paramètres destinés à configurer le serveur de test : ceux-ci permettront ensuite de pouvoir utiliser certaines fonctionnalités de Dreamweaver comme l'aperçu dans le navigateur d'une page PHP ou encore d'utiliser les comportements serveur de Dreamweaver (voir l'encadré de la page suivante pour en savoir plus sur les comportements serveur). Dans le cadre de cet ouvrage, nous n'utiliserons pas les comportements serveur, par contre vous apprécierez certainement d'avoir rapidement un aperçu de votre page PHP dans le Web Local de votre serveur de développement, aussi nous vous conseillons de suivre la procédure ci-dessous pour configurer un serveur d'évaluation pour Dreamweaver.

Pour configurer la catégorie Serveur d'évaluation commencez par choisir le modèle de serveur en sélectionnant le couple PHP/MySQL dans la liste déroulante. En dessous de ce premier champ, il faut choisir l'accès en mode Local/Réseau puisque le serveur Web, le moteur PHP et le serveur MySQL (installés avec la suite Wamp5) sont disponibles sur l'ordinateur de développement où est installé Dreamweaver. Dans le troisième champ, vous n'avez rien à changer car le dossier du serveur d'évaluation est le même que celui de la racine locale (revoir page Infos locales) : `C:\wamp\www\SITEajax\`. Par contre, l'URL sous laquelle sera disponible le site sur le serveur d'évaluation est `http://localhost/SITEajax/` car la racine du serveur `http://localhost/` se trouve au niveau du dossier `C:\wamp\www\`. Il convient donc d'ajouter le nom du répertoire à la suite de l'URL racine dans le champ Préfixe d'URL (voir figure 7-4).

Vous avez maintenant terminé la configuration de votre site et il ne vous reste plus qu'à cliquer sur le bouton OK en bas de la fenêtre pour valider votre site.

Figure 7-4

La rubrique Serveur d'évaluation de la fenêtre Définition du site permet de définir les différents paramètres du serveur qui testera les scripts dynamiques PHP.



Les comportements serveur de PHP

Les comportements serveur permettent de générer automatiquement des scripts PHP en interaction avec une base de données. Dans le cadre de cet ouvrage, nous n'utiliserons pas les comportements serveur de Dreamweaver pour créer nos scripts PHP, mais si vous êtes intéressé par ces fonctionnalités, nous vous invitons à vous référer à un autre ouvrage Eyrolles du même auteur entièrement dédié aux comportements de serveur de Dreamweaver : *PHP/MySQL avec Dreamweaver*.

Éditeur HTML

L'éditeur de Dreamweaver permet de travailler selon trois modes différents : mode Création, mode Code ou mode Mixte. Le mode Création permet de créer facilement des pages HTML en Wysiwyg (*What You See Is What You Get* ou encore « Ce que vous voyez est ce que vous obtenez ») ainsi l'utilisateur voit directement à l'écran à quoi ressemblera le résultat final. Ce mode est très bien adapté à la création de pages HTML car grâce à cette interface intuitive l'utilisateur peut mettre en forme très rapidement les éléments qui composent sa page.

Avant d'utiliser l'éditeur, nous allons créer une nouvelle page HTML. Pour cela, cliquez depuis le menu sur Fichier puis sélectionnez Nouveau (ou utilisez le raccourci clavier Ctrl + N). Dans la fenêtre Nouveau document, sélectionnez Page vierge dans la première colonne puis HTML dans la seconde puis validez en cliquant sur le bouton Créer. À noter que Dreamweaver permet aussi de choisir de nombreuses mises en forme pré-configurées (troisième colonne de la fenêtre Nouveau document) mais que dans le cadre de cet ouvrage nous n'utiliserons pas cette option.

Par défaut le nouveau document créé s'affiche en mode Création, mais vous pourrez passer facilement du mode Code au mode Création en cliquant simplement sur le bouton correspondant situé dans la barre d'outils document (si cette barre d'outils n'apparaît pas, cliquez sur Affichage>Barre d'outils et cochez Document).

La barre d'outils Insertion

Dans une page HTML, la barre d'Insertion de Dreamweaver comporte sept onglets qui permettent d'accéder chacun à un panneau différent. Chaque panneau comporte des boutons classés par thème (Commun, Mise en forme, Formulaire…) qui permettent d'insérer des éléments dans la page.

L'insertion peut être effectuée par un clic sur un simple bouton placé dans le panneau, ou à l'aide d'un menu déroulant correspondant à une sous-catégorie (la présence de ce menu est signalée par une petite flèche située à côté du bouton). Dans la majorité des cas, une boîte de dialogue s'affiche afin de renseigner les paramètres de l'élément à insérer.

Cette barre d'outils Insertion est très pratique en mode Création car elle permet d'ajouter des éléments dans la page Web par un simple clic ou à l'aide d'un glisser-déposer mais elle peut aussi être exploitée en mode Code (dans ce cas il faudra placer au préalable le pointeur au point d'insertion dans le code de la page, voir la figure 7-5).

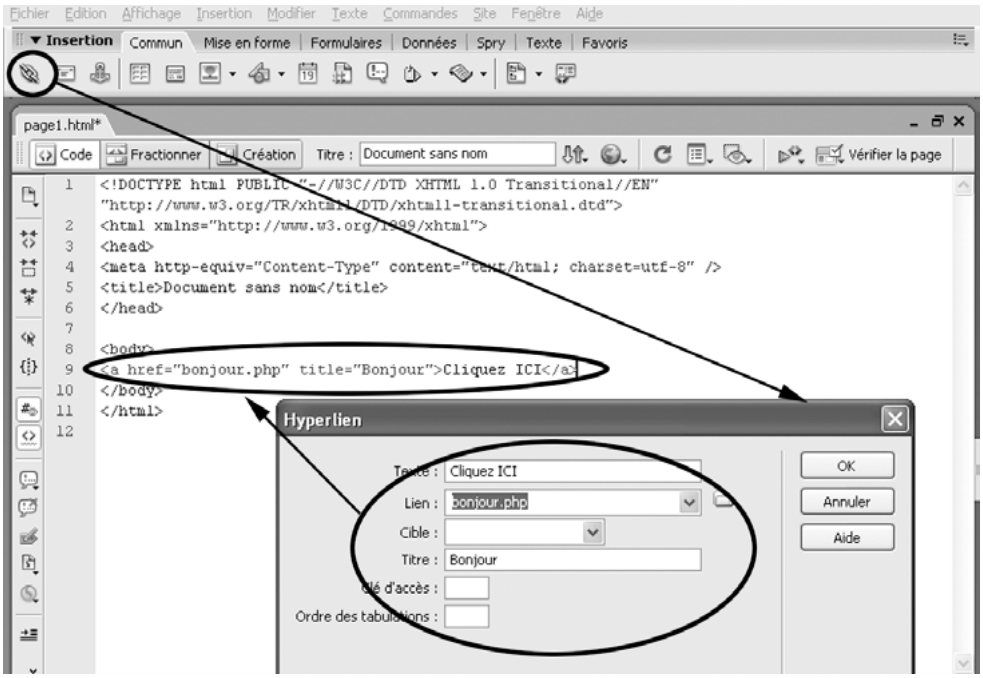


Figure 7-5

Exemple d'insertion d'un lien hypertexte dans une page HTML

Le panneau des Propriétés

Le panneau Propriétés est contextuel et affiche toutes les informations ou paramètres liés à l'objet sélectionné dans le document. Par exemple si vous choisissez une image, il liste tous les attributs de l'image, comme sa hauteur, sa largeur, l'emplacement du fichier image ou encore l'URL vers laquelle elle établit un lien, s'il s'agit d'une image cliquable. Ces attributs peuvent être ainsi facilement modifiés grâce aux champs correspondants dans le panneau Propriétés.

Sélecteur de balise

Le sélecteur de balise est une zone horizontale placée en bas de la fenêtre Document. Selon la position du pointeur dans la fenêtre Document, elle fait apparaître toutes les balises parents ordonnées hiérarchiquement de la gauche vers la droite.

Cette fonction est très appréciable lorsqu'on désire sélectionner précisément un élément HTML de la page Web en mode Création. En effet, il suffit pour cela de cliquer sur la balise concernée dans le sélecteur (voir figure 7-6) pour que tout le code correspondant apparaisse en surbrillance. Ce sélecteur est également intéressant pour identifier la partie de code correspondant à un élément graphique ou à un tableau, car celle-ci apparaît en surbrillance dans la zone de code.

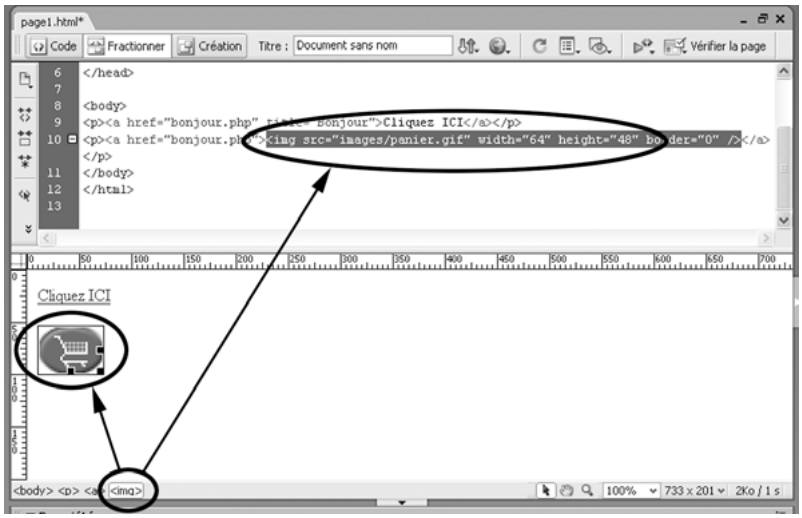


Figure 7-6

Le sélecteur de balise permet de sélectionner d'une manière précise et rapide une balise HTML du code source ou dans la fenêtre Document. Dans l'exemple ci-dessus, la balise <img> a été sélectionnée à l'aide du sélecteur de balise.

Indicateur de code HTML

L'indicateur de code vous guide dans la saisie de votre code. Vous pouvez l'utiliser depuis la fenêtre du document en mode Code.

Pour le configurer, cliquez depuis le menu de Dreamweaver sur l'entrée Edition puis l'option Préférences et sélectionnez la rubrique Indicateur de code. Vous pouvez le désactiver ou augmenter son temps de réaction si vous le trouvez trop envahissant. De même, depuis cette fenêtre, vous pouvez neutraliser la création automatique de la balise de fin (décocher l'option Activer l'achèvement automatique) ou réduire son action à certains types de codes (cochez les cases correspondant aux actions désirées : noms des balises, noms des attributs, valeurs des attributs, valeurs des fonctions…).

Pour illustrer son fonctionnement, nous vous proposons de commenter son utilisation pour la création d'une balise d'un tableau HTML.

Création d'une balise de tableau HTML avec l'indicateur de code (voir figure 7-7) :

Depuis l'éditeur de code (fenêtre mode Code), saisissez le début d'une balise de tableau, par exemple `<t`. L'indicateur apparaît et vous propose les différentes balises HTML commençant par t (si la première lettre n'est pas suffisante pour afficher la bonne balise, saisissez une deuxième lettre et ainsi de suite jusqu'à l'affichage correct de la balise désirée). Dès que vous pouvez sélectionner la balise désirée dans le menu déroulant, validez-la en appuyant sur la touche Entrée. Le début de la balise est complété automatiquement. Appuyez sur la touche `Espace` pour afficher de nouveau l'indicateur de code, cette fois en mode Attribut (notez qu'une petite icône précède chacun des attributs afin de vous rappeler le type de fonction auquel il appartient). De la même manière que pour les balises HTML, il suffit de valider l'attribut désiré pour qu'il apparaisse dans le code. Cette fois, le pointeur de la souris est positionné entre les guillemets de la valeur de l'attribut. Si les valeurs attendues sont standards, un nouveau menu déroulant propose les choix possibles. Dans le cas contraire, saisissez la valeur (par exemple 1 pour l'attribut `border`). Ensuite, appuyez sur la touche du clavier Fin pour vous placer après les guillemets, puis sur la touche Espace pour saisir un autre argument. Renouvelez la même opération pour

le deuxième argument et terminez votre saisie par un caractère >, afin que l'inspecteur affiche automatiquement la balise de fermeture correspondante.

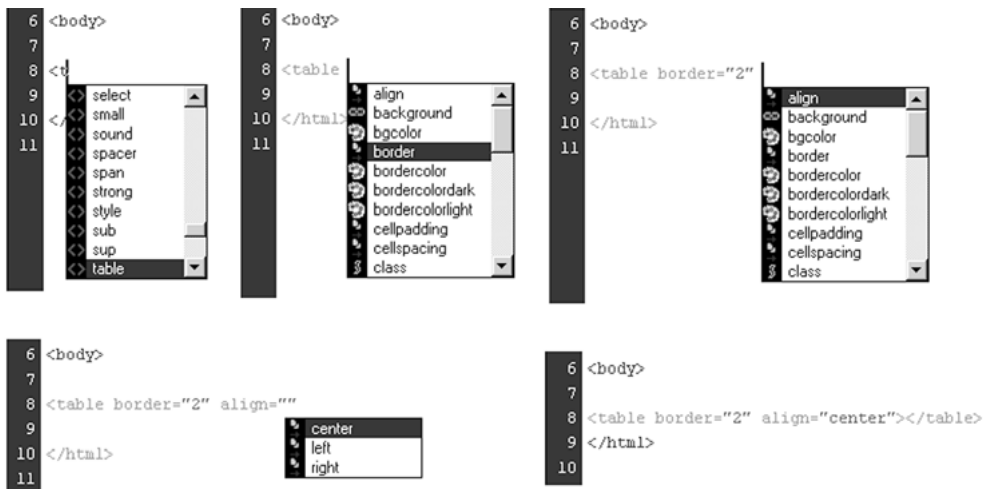


Figure 7-7

L'indicateur de code vous permet de saisir facilement vos balises HTML sans avoir à vous référer à la documentation.

Éditeur PHP

Pour la création des fichiers PHP nous allons nous intéresser plus particulièrement à l'utilisation du mode Code et à ses fonctions qui nous assisteront dans l'écriture de nos scripts PHP.

Afin d'utiliser l'éditeur PHP, nous allons créer une nouvelle page PHP. Pour cela, cliquez dans le menu sur Fichier puis sélectionnez Nouveau (ou utilisez le raccourci clavier Ctrl + N). Dans la fenêtre Nouveau document, sélectionnez Page vierge dans la première colonne, PHP dans la seconde, puis validez en cliquant sur le bouton Créer.

Une fois le nouveau document créé, vous pouvez passer en mode Code (si cela n'est pas déjà le cas) en le sélectionnant dans le menu Affichage>Code ou en cliquant sur le bouton Code situé à gauche de la barre d'outils Document.

Options de l'éditeur de code

Avant de saisir la première ligne de code, assurez-vous que toutes les options de l'éditeur sont configurées correctement. Pour cela, cliquez sur le bouton d'options à droite de la barre d'outils Document (voir figure 7-8 ; si cette barre d'outils n'apparaît pas, cliquez sur Affichage>Barre d'outils et cochez Document).

Parmi les différents choix du menu Option nous vous conseillons de valider les options qui correspondent aux définitions suivantes :

- **Retour automatique à la ligne** – Renvoie le code à la ligne lorsqu'il excède la largeur de la fenêtre de l'éditeur. Nous vous conseillons de cocher cette option car elle rend la lecture plus confortable lors de la rédaction des scripts.

- **Numéro de ligne** – Permet la numérotation des lignes dans la fenêtre Document en mode Code ou dans l'inspecteur de code. Cette option est fort utile lors de vos premiers dépannages de scripts, pour repérer la ligne signalée par le message d'erreur PHP.
- **Surligner le code non valide** – Pour éviter que Dreamweaver mette en surbrillance certaines balises HTML lors de la rédaction de vos programmes PHP, il est indispensable de respecter l'équilibre du code HTML (une balise d'ouverture doit toujours être fermée) si vous émulez des balises HTML à l'aide de scripts PHP (par exemple : `echo "<TABLE>"`). Cela permet notamment aux graphistes de pouvoir afficher correctement le document en mode Création, même après l'ajout d'un script PHP dans la page.
- **Coloration de la syntaxe** – La coloration syntaxique est très utile pour la recherche de pannes et pour la saisie. Dreamweaver permet d'utiliser la coloration syntaxique du langage PHP et il serait dommage de s'en priver. Il est possible de configurer chaque couleur selon le type de code représenté (pour accéder à cet écran, affichez la fenêtre Préférences, rubrique Coloration syntaxique et PHP, puis cliquez sur le bouton Modifier le modèle de coloration). Pour une bonne lisibilité de votre code source, nous vous conseillons de conserver les couleurs attribuées par défaut.
- **Retrait auto** – Automatise la mise en retrait du code dès que vous appuyez sur la touche Entrée lors de la rédaction du code.

Figure 7-8

Le mode Code de Dreamweaver permet de saisir vos scripts avec le même confort qu'avec un éditeur dédié au développement de code source. Pour faciliter votre saisie des fonctions PHP, nous vous conseillons d'utiliser le raccourci clavier Ctrl + Espace. De même, le bouton d'options de la barre d'outils Document active différentes fonctions de la fenêtre Document, très appréciables pour une utilisation en mode Code.

Indicateur de code PHP

L'indicateur de code utilisé en PHP est le même que celui que nous avons déjà présenté et utilisé pour nous assister dans la création de balise HTML (revoir la partie dédiée à l'éditeur HTML pour plus de détails sur son paramétrage).

Comme pour le HTML, l'indicateur de code permet de saisir du code PHP facilement et rapidement, sans avoir à se référer à la documentation. Pour illustrer son fonctionnement, nous vous proposons de commenter son utilisation pour le paramétrage des arguments d'une fonction PHP.

Aide à la saisie des fonctions PHP

Dreamweaver propose un système d'aide à la saisie des fonctions PHP qui se matérialise par un menu déroulant suggérant les différentes fonctions commençant par les premiers caractères déjà saisis. Pour faire apparaître ce menu, il suffit d'utiliser la combinaison de touches Ctrl + Espace.

Création d'une fonction PHP avec l'indicateur de code :

L'indicateur de code peut aussi gérer les différentes fonctions PHP, ainsi que les variables HTTP (par exemple, si vous saisissez `$_`, il vous propose les différents types de variables serveurs disponibles dans un menu déroulant : `$_GET, $_POST`…). En guise d'illustration, voici la démarche à suivre afin d'exploiter pleinement les possibilités de cet outil pour déclarer une constante insensible à la casse (nous utilisons pour cela la fonction `define()`, avec comme troisième argument la valeur `1`, voir figure 7-9). Commencez par saisir le début de la fonction dans une zone PHP de votre page (zone encadrée par les balises `<? php` et `?>`), soit « `define` »). Une zone d'assistance s'affiche alors en dessous de la fonction et vous rappelle le type et le rôle des différents arguments attendus pour la fonction. Commencez la saisie en suivant ces indications (il est à noter que, dès que la saisie du premier argument commence, la zone d'assistance disparaît). Si vous ajoutez ensuite une virgule, la zone d'assistance apparaît de nouveau et vous informe sur les arguments restants à saisir. Procédez de la même manière pour tous les arguments attendus et terminez votre saisie par une parenthèse fermante. N'oubliez pas le point-virgule si vous êtes à la fin de l'instruction.

Figure 7-9

L'indicateur de code vous permet de connaître les différents arguments attendus par les fonctions PHP.

La barre d'outils Insérer, option PHP

L'onglet PHP de la barre d'outils `Insérer` permet d'ajouter rapidement des balises PHP dans la fenêtre Document en mode Code. Pour que l'onglet PHP apparaisse, il faut se trouver dans une page dynamique PHP (menu Fichier>Nouveau>Page vierge + PHP). Hormis le bouton d'insertion de commentaire, les différents boutons du panneau insèrent des balises PHP de début et de fin (`<?php` et `?>`). Ce panneau est donc destiné à insérer du code PHP isolé dans le code source d'une page HTML et non à vous assister dans l'édition d'un script PHP de plusieurs instructions. Les fonctions des boutons de l'onglet PHP sont les suivantes :

- Ajout de variables de formulaires :

```
<?php $_POST[]; ?>
```

- Ajout de variables d'URL :

```
<?php $_GET[]; ?>
```

- Ajout de variables de session :

```
<?php $_SESSION[]; ?>
```

- Ajout de variables de cookie :

```
<?php $_COOKIE[]; ?>
```

- Ajout de la fonction `include` :

```
<?php include(); ?>
```

- Ajout de la fonction `require?` :

```
<?php require(); ?>
```

- Ajout d'un bloc de code PHP :

```
<?php ?>
```

- Ajout de la fonction `echo` (affichage) :

```
<?php echo ?>
```

- Ajout d'un commentaire dans le code :

```
/* */
```

- Ajout de la fonction `if` (test d'une condition) :

```
<?php if ?>
```

- Ajout de la fonction `else` (complémentaire de la fonction `if`) :

```
<?php else ?>
```

- Ajout d'une balise par le biais du sélecteur de balise

Test d'une page PHP

Pour vous initier à l'enregistrement et à la procédure de test d'une page PHP depuis le serveur Web Local, nous vous proposons de créer un premier document PHP avec Dreamweaver.

Le document que nous allons créer devra afficher une page phpinfo. Cette page utilisera la fonction `phpinfo()` pour afficher à l'écran toutes les informations utiles sur la version et la configuration du PHP installé sur votre serveur.

À noter

Une page phpinfo est déjà disponible sur votre serveur local par l'intermédiaire de la suite Wamp5 dans la page du localhost. Cependant, la page que nous allons créer vous sera certainement utile si vous possédez un hébergement Web car elle vous permettra de connaître les caractéristiques du PHP installé sur votre serveur distant. Nous rappelons à ce sujet qu'il faut choisir un serveur de développement local dont les caractéristiques sont proches de celles de votre serveur distant (évidemment, si vous n'avez pas encore choisi votre hébergeur, l'inverse est aussi valable…).

Commencez par créer un nouveau document PHP en suivant la démarche détaillée au début de cette partie. Enregistrez tout de suite votre document sous le nom `phpinfo.php` à la racine de `SITEajax`. À la racine de `SITEajax`, vous devez retrouver le fichier `bonjour.php` créé lors du test de l'infrastructure serveur.

Test avec l'aperçu de Dreamweaver

Assurez-vous que vous êtes bien en mode Code (si besoin, cliquez sur le bouton Code placé en haut à gauche de la fenêtre Document). Dans la barre des outils Insérer, cliquez sur l'onglet PHP pour disposer des boutons dédiés à PHP détaillés précédemment. Placez votre pointeur entre les balises `<body>` et `</body>` puis cliquez sur le bouton Bloc de code afin d'ajouter automatiquement les balises ouvrante et fermante d'un script PHP (`<?php` et `?>`). Ajoutez ensuite entre ces deux balises l'instruction suivante : `phpinfo()` ; puis enregistrez votre fichier en utilisant le raccourci clavier Ctrl + S (voir le code dans l'éditeur de figure 7-10).

Avant de tester le fonctionnement de cette page PHP, assurez-vous que le serveur Web Local (Wamp5) est bien démarré. Cliquez ensuite sur le bouton Aperçu/Débogage (icône en forme de globe dans la barre d'outils de la fenêtre Document, vous pouvez aussi utiliser la touche F12). Une fenêtre du navigateur par défaut doit alors s'ouvrir pour vous permettre de tester votre page (voir figure 7-10). Il est important pour la suite que la page à tester

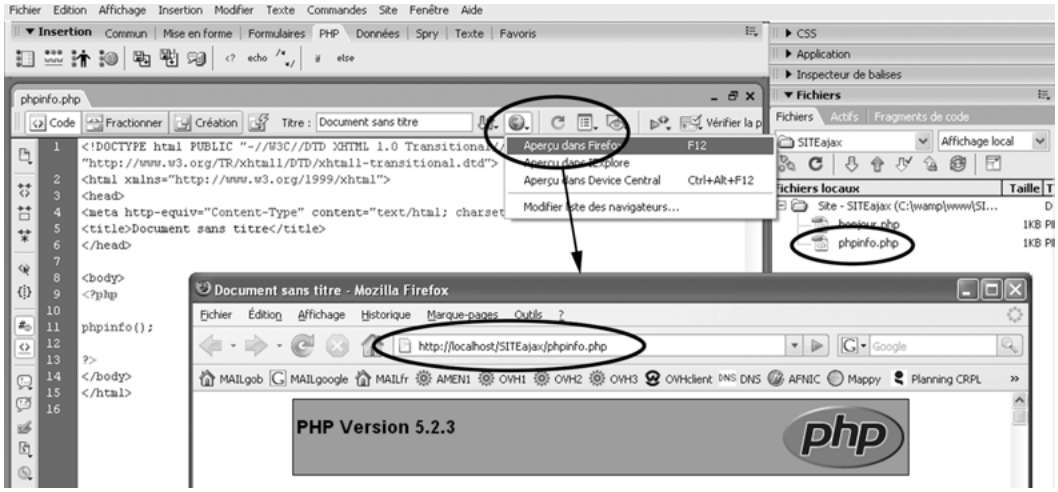


Figure 7-10

Test d'une page PHP dans le Web Local

s'ouvre bien depuis le serveur Web Local `http://localhost`. Pour que cela puisse se faire, il faut au préalable avoir configuré correctement le serveur d'évaluation du site Dreamweaver (revoir si besoin la partie `Définition d'un site / serveur d'évaluation` détaillée précédemment) et que le serveur Wamp5 soit en marche.

Test avec les raccourcis clavier de Windows

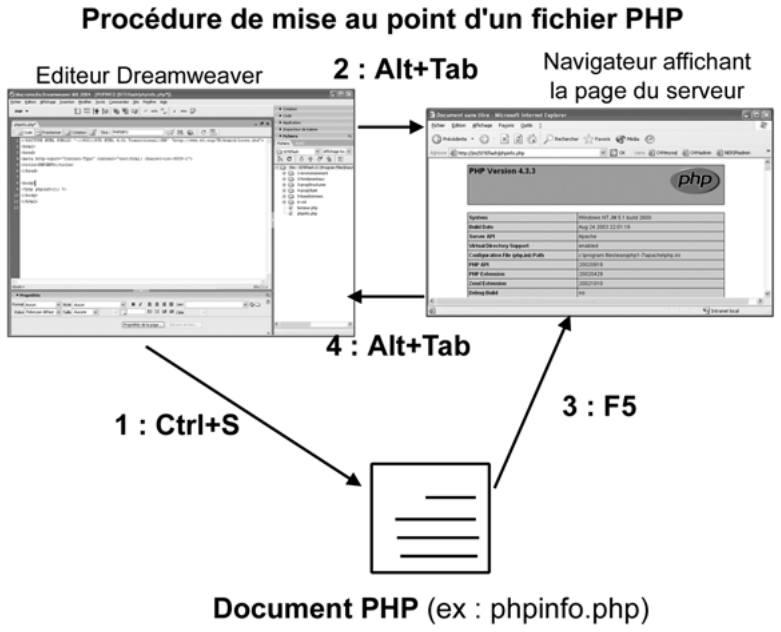
Une fois que vous aurez testé une première fois votre page dans le Web Local, la barre des tâches de votre ordinateur (zone située au centre et en bas de l'écran du PC) affiche deux boutons. L'un correspond à l'éditeur Dreamweaver et l'autre au navigateur qui affiche la page `phpinfo.php`. Vous pouvez maintenant passer d'une fenêtre à l'autre en cliquant sur le bouton correspondant à la page désirée pour mettre au point progressivement votre programme, mais il est beaucoup plus rapide d'utiliser le raccourci clavier Alt + Tab.

Imaginons maintenant que vous êtes dans le Web Local (le navigateur) et que le résultat de la page que nous venons de créer ne vous convienne pas. Il vous faut alors basculer de nouveau dans l'éditeur (en utilisant évidemment le raccourci clavier Alt + Tab) afin de modifier le contenu du code source de la page PHP. Pour illustrer cette démarche, nous allons ajouter le titre PHPINFO à la page. Une fois la modification effectuée, enregistrez votre page (en utilisant le raccourci clavier Ctrl + S) puis basculez de nouveau dans le navigateur (Alt + Tab) afin de voir le résultat. Une fois dans le navigateur, vous devrez actualiser la page à l'aide du raccourci F5 (ou en cliquant sur le bouton Actualiser de votre navigateur) pour voir apparaître l'écran modifié. À noter que pour actualiser la page, vous pouvez aussi utiliser le raccouci Ctrl + R qui évite d'avoir à appuyer sur la touche F5 située en haut du clavier.

Même si cette démarche peut paraître un peu rébarbative au début, vous allez vite apprécier le gain de temps dans vos développements dès que vous aurez acquis un peu de dextérité dans l'usage répété de cette série de raccourcis clavier (voir figure 7-11).

Figure 7-11

Enchaînement des différents raccourcis clavier utilisés en phase de développement



Cette même méthode pourra aussi être utilisée par la suite pour tester vos programmes Java Script.

En phase de développement, vous devrez très souvent effectuer des tests répétés et si la méthode de l'aperçu de Dreamweaver peut paraître plus rapide au début, elle devient moins intéressante dès que vous aurez de multiples modifications à faire. De plus, s'ajoute à cela le fait que l'aperçu créé à chaque test une nouvelle fenêtre dans le navigateur, ce qui peut devenir vite envahissant pour des aperçus répétés.

Configuration du navigateur de test par défaut

Le navigateur utilisé par l'aperçu de Dreamweaver peut être défini dans la fenêtre des préférences. Par la suite nous utiliserons toujours Firefox et son extension Firebug pour déboguer les programmes JavaScript et nous vous suggérons dès maintenant de vérifier que Firefox est bien configuré comme navigateur par défaut. Pour cela, cliquez depuis le menu de Dreamweaver sur l'entrée Edition puis l'option Préférences et sélectionnez la rubrique Aperçu dans le navigateur. Sélectionnez Firefox dans la liste de la fenêtre puis cocher l'option Navigateur principal dans la rubrique Navigateur par défaut. La touche F12 (touche de raccourci pour déclencher un aperçu) doit alors apparaître à droite du navigateur Firefox dans la liste de la fenêtre.

Les références PHP de poche

Dans le panneau Résultats, onglet Référence, Dreamweaver met à votre disposition un dictionnaire de poche regroupant toutes les syntaxes des fonctions PHP. Pour y accéder, ouvrez le panneau Code et cliquez sur l'onglet Référence (n'hésitez pas à élargir la fenêtre afin de pouvoir lire les différentes options des menus déroulants). Dans le haut du panneau, sélectionnez `O'REILLY - Référence PHP de poche` dans le premier menu déroulant `Livre` puis la famille de fonction que vous désirez consulter dans le deuxième menu déroulant et enfin la fonction dans le troisième menu déroulant. N'hésitez pas à consulter fréquemment ces informations si vous avez un doute sur la syntaxe d'une fonction.

Les références du langage SQL

Dans le même panneau Résultats onglet Référence, Dreamweaver met à votre disposition un second dictionnaire dédié au langage SQL. Vous pourrez ainsi vous assurer de la validité de la syntaxe de vos requêtes SQL avant de les intégrer dans vos scripts PHP. Le chapitre sur le MySQL à la fin de cet ouvrage présente en détail les requêtes et les clauses SQL couramment utilisées.

Pour accéder à ce dictionnaire, vous devez au préalable ouvrir le panneau Résultat et cliquer sur l'onglet Référence. Dans le haut du panneau, sélectionnez `O'REILLY - Référence du langage SQL` dans le premier menu déroulant Livre puis la rubrique que vous désirez consulter dans le deuxième menu déroulant (sélectionnez, par exemple, la rubrique Référence des commandes) et enfin la sous-rubrique dans le troisième menu déroulant (dans notre exemple, ce troisième menu affiche une liste de commandes parmi lesquelles nous sélectionnons SELECT).

Éditeur JavaScript

L'intégration d'un programme JavaScript dans une page HTML peut être effectué très simplement à l'aide d'une balise `<script>` qui comportera un attribut `type` précisant qu'il

s'agit d'un programme JavaScript, soit `<script type="text/javascript">`. Dans ce cas, le code JavaScript saisi dans cette balise sera reconnu par le navigateur et interprété en rapport. Cette balise peut être placée dans la tête (balise `<head>`) mais aussi dans le corps de la page (balise `<body>`) selon l'usage du programme.

Exemple de script placé directement dans une page HTML :

```
<script type="text/javascript">
alert("BONJOUR");
</script>
```

En pratique, il est souvent plus judicieux de placer son programme dans un fichier externe de sorte à mutualiser les scripts du programme avec les autres pages du site. Dans ce cas, la balise `<script>` sera vide mais comportera un second attribut `src` qui indiquera la localisation du fichier externe (dans l'exemple ci-dessous le fichier externe se trouve dans le même répertoire que la page HTML).

Exemple d'un lien, placé dans une page HTML, appelant un fichier JavaScript externe :

```
<script type="text/javascript" src="scriptsJS.js"></script>
```

Pour que vous puissiez facilement créer des programmes JavaScript avec Dreamweaver, nous vous proposons maintenant d'illustrer ci-dessous ces deux méthodes avec deux exemples très simples.

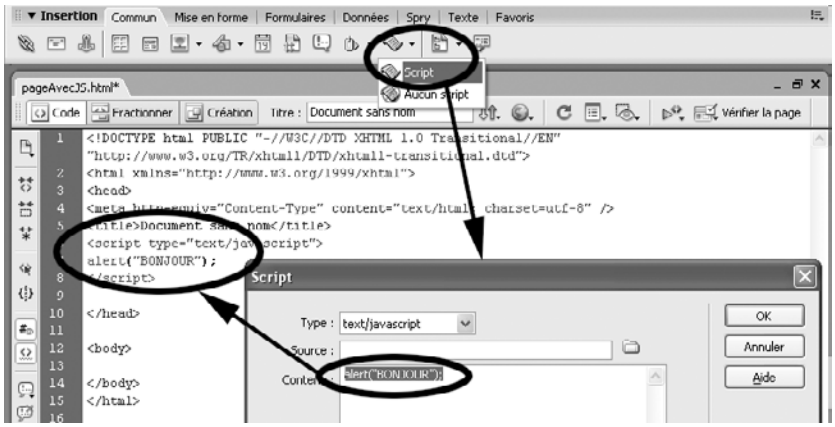
Insertion d'un script JavaScript dans une page HTML

Commencez par créer une page HTML comme nous l'avons indiqué au début de ce chapitre. Placez ensuite votre pointeur au point d'insertion où vous désirez ajouter votre script.

Dans la barre d'outils Insertion, cliquez sur l'onglet du panneau Commun, puis cliquez sur l'icône Script (voir figure 7-12) et sélectionnez l'option Script. Dans la boîte de dialogue, sélectionnez pour le type, l'option `text/javascript` puis saisissez le script désiré dans le champ Contenu. Pour ce premier exemple, nous saisirons une simple fonction JavaScript d'alerte (`alert('Bonjour')`) qui affichera le mot `Bonjour` dans une boîte de dialogue. Une fois le script saisi, validez la fenêtre en cliquant sur le bouton OK. Le script doit ensuite apparaître dans le code de la page HTML à l'endroit du point d'insertion.

Figure 7-12

Insertion d'un script JavaScript dans une page HTML



Test d'une page JavaScript

Pour tester rapidement votre page, vous pouvez aussi cliquer sur l'icône en forme de globe dans la barre d'outils de la fenêtre Document (vous pouvez aussi utiliser la touche F12). Une fenêtre du navigateur par défaut doit alors s'ouvrir pour vous permettre de tester votre premier programme JavaScript. Cependant, pour des essais répétitifs, nous vous conseillons (comme pour le test de la page PHP) d'utiliser les raccourcis Windows pour être plus productif pour la mise au point de votre programme.

Comme par la suite nous utiliserons des programmes JavaScript en interaction avec des fichiers PHP, il est important que la page HTML contenant le programme JavaScript à tester s'ouvre aussi depuis le serveur Web Local `http://localhost`. Si ce n'est pas le cas, revoyez la configuration du serveur d'évaluation du site Dreamweaver et assurez vous que le serveur Wamp5 est en marche.

Évidemment, une fois la page HTML affichée dans le Web Local, vous devrez ensuite utiliser les fonctionnalités de Firebug pour mettre au point vos programmes JavaScript contenus dans la page HTML. Nous détaillerons ces fonctionnalités lors des premiers ateliers de la partie 3 de cet ouvrage.

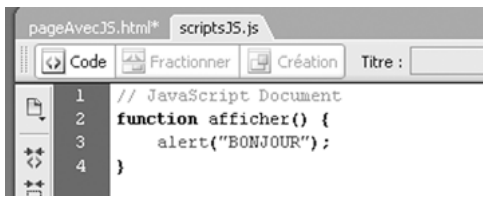
Lier un fichier JavaScript externe dans une page HTML

Commençons par créer le fichier JavaScript externe. Pour cela cliquez dans le menu sur Fichier puis sélectionnez Nouveau (ou utilisez le raccourci clavier Ctrl + N). Dans la fenêtre Nouveau document, sélectionnez Page vierge dans la première colonne et Javascript dans la seconde puis validez en cliquant sur le bouton Créer.

Une fois le fichier créé, vous pouvez saisir votre programme JavaScript directement dans le fichier sans avoir à ajouter de balise spécifique (voir figure 7-13). Pour ce premier exemple, nous nous contenterons d'une simple fonction `Afficher()` exécutant une fenêtre d'alerte dans la page, mais sachez que Dreamweaver peut aussi vous assister dans la création d'un programme plus conséquent à l'aide de son indicateur de code JavaScript (comme pour le HTML et le PHP). Enregistrez ensuite ce fichier dans le même répertoire que celui de la page HTML sous le nom de votre choix mais avec l'extension `.js`.

Figure 7-13

Rédaction d'une fonction JavaScript dans un fichier externe



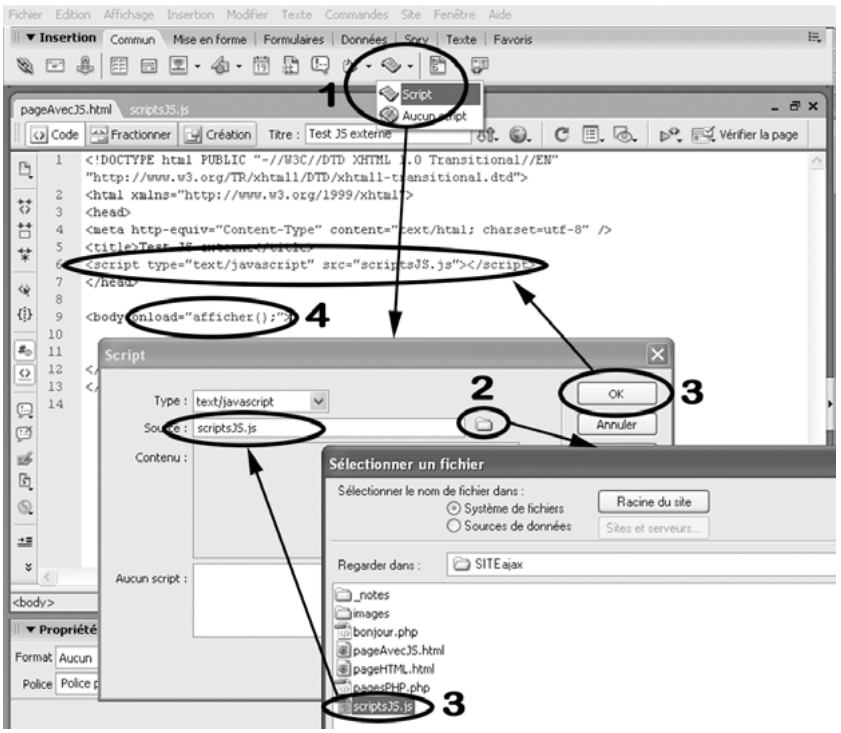
Créez ensuite une page HTML comme nous l'avons indiqué au début de ce chapitre. Avant de lier le fichier JavaScript dans la page HTML, veillez à enregistrer la page HTML concernée afin que Dreamweaver puisse ensuite créer automatiquement le chemin relatif qui mène au fichier source JavaScript.

Positionnez votre pointeur dans la balise `<head>` de la page, puis depuis la barre d'outils Insertion, cliquez sur l'onglet du panneau Commun, cliquez sur l'icône `Script` (voir

repère 1 figure 7-14) puis sélectionnez l'option Script. Dans la boîte de dialogue, sélectionnez pour le type, l'option `text/javascript` puis cliquez sur le petit dossier jaune situé à droite du champ Source (voir repère 2 voir figure 7-14). Localisez le fichier JS qui doit être utilisé et validez votre choix en cliquant sur le bouton OK. Validez ensuite la fenêtre Script en cliquant une seconde fois sur le bouton OK (voir repère 3 figure 7-14). Le lien doit alors apparaître dans le code de la page. Vous pouvez désormais exploiter toutes les fonctions du fichier externe comme si les déclarations de ces fonctions étaient placées directement dans la page HTML (voir repère 4 figure 7-14).

Figure 7-14

Liaison d'un fichier JavaScript externe dans une page HTML



Les fragments de code JavaScript

Dans le panneau Fichier onglet Fragment de code, Dreamweaver met à votre disposition toute une série de code préenregistrés. Ils sont regroupés selon leur usage et classés dans des dossiers spécifiques. Pour insérer un de ces fragments dans votre fichier JavaScript, il suffit de placer le curseur au point d'insertion dans la page et de faire un double clic sur le fragment désiré. Il sera alors copié automatiquement dans le document en cours à l'endroit du point d'insertion.

Par la suite vous pourrez aussi créer vos propres fragments de code (pour le JavaScript, mais aussi pour le PHP !). Vous pourrez ainsi capitaliser vos réalisations, voire les partager avec d'autres développeurs. Pour créer un fragment personnalisé, il suffit d'utiliser les boutons situés en bas du panneau des fragments et de l'enregistrer dans le dossier correspondant (pour les scripts PHP, pensez à créer au préalable un dossier PHP au même niveau que le dossier JavaScript actuel pour accueillir votre futures réalisations).

Les références JavaScript de poche

Dans le panneau Résultat onglet Référence, Dreamweaver met à votre disposition un dictionnaire de poche regroupant toutes les syntaxes des fonctions `JavaScript`. Pour y accéder, ouvrez le panneau Résultat et cliquez sur l'onglet Référence. Dans le haut du panneau, sélectionnez `O'REILLY - Référence Javascript` dans le premier menu déroulant Livre puis la famille de fonction que vous désirez consulter dans le deuxième menu déroulant et enfin la fonction dans le troisième menu déroulant. N'hésitez pas à consulter fréquemment ces informations si vous avez un doute sur la syntaxe d'une fonction.

Partie III

Ateliers de création d'applications Ajax-PHP

L'objectif des ateliers de cette partie n'est pas de réaliser des applications toujours opérationnelles et sophistiquées mais au contraire de bâtir pas à pas des applications Ajax très simples, à partir de quelques lignes de code, en alternant judicieusement des tests pratiques, analyses, constats et améliorations. Ainsi, au fil des versions successives de ces applications pédagogiques, vous découvrirez toutes les structures possibles et comprendrez le rôle des différentes parties d'une application Ajax-PHP.

8

Applications Ajax-PHP synchrones

Pour commencer simplement, je vous propose une série d'ateliers qui vous permettra de créer progressivement une première application synchrone.

Pour illustrer son fonctionnement nous réaliserons une petite application qui simulera le fonctionnement d'une machine à sous en ligne. Côté client, l'application sera constituée d'une page HTML dans laquelle nous construirons progressivement un moteur Ajax dont l'objectif sera de simuler le fonctionnement d'une machine à sous. Pour cela l'utilisateur devra appuyer sur un bouton pour déclencher une requête depuis son navigateur et relancer ainsi la machine à sous. Côté serveur, un fichier PHP réceptionnera et traitera la requête client puis renverra le montant du gain en retour qui s'affichera ensuite dans la page Web.

Atelier 8-1 : requête synchrone sur un fichier texte sans feuille de styles

Composition du système

Nous allons commencer par mettre en place une structure minimaliste pour tester le fonctionnement d'une première requête synchrone sur un simple fichier texte (voir figure 8-1). Cette première structure est composée :

- d'une page HTML (`index.html`) dans laquelle nous allons intégrer un bouton de formulaire pour déclencher le jeu, une zone d'affichage du résultat et le JavaScript du moteur Ajax ;
- d'un simple fichier texte (`gainMax.txt`) dans lequel nous allons saisir la valeur du gain maximum, soit 100.

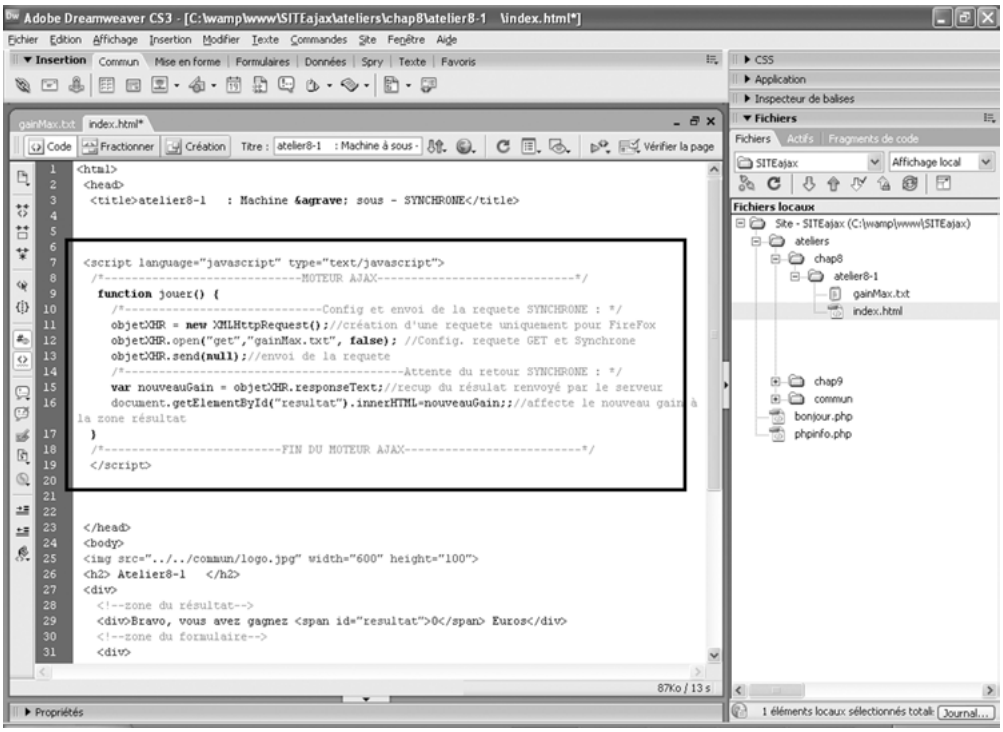


Figure 8-1

Saisie du moteur Ajax dans la balise <head> de la page

> **Prérequis concernant l'objet XMLHttpRequest (XHR)**
>
> Comme vous avez déjà découvert le fonctionnement de l'objet `XMLHttpRequest` dans le chapitre 4 de ce même ouvrage, nous nous appuierons sur ce prérequis dans la rédaction des ateliers de ce chapitre. Si toutefois certaines méthodes ou propriétés de cet objet vous semblent inconnues, n'hésitez pas à vous reporter au chapitre 4 pour revoir son concept.
>
> À noter aussi que nous aurons souvent l'occasion de faire référence à l'objet `XMLHttpRequest` dans cet ouvrage et pour alléger l'écriture, nous utiliserons fréquemment son appellation abrégée XHR.

Fonctionnement du système

Le bouton JOUER de la page permettra à l'utilisateur d'afficher la valeur maximum des gains. Ce bouton sera relié à un gestionnaire d'événement `onclick` qui appellera une fonction `jouer()` (voir code 8-1, à noter que pour cette première application le gestionnaire sera intégré dans la balise HTML `<input>` mais nous verrons par la suite qu'il est préférable de déclarer les gestionnaires d'événements directement dans le code JavaScript afin de bien séparer le programme de la structure de la page).

Cette fonction déclenchera le moteur Ajax (voir code 8-2) qui créera un objet XMLHttpRequest (par la suite, nous utiliserons son appellation abrégée XHR) puis le configurera (à l'aide de la méthode `open()` de l'objet : choix de la méthode `GET` et ciblage du fichier `gainMax.txt` en mode Synchrone) avant d'envoyer la requête (à l'aide de la méthode `send()` de l'objet).

L'information contenue dans le fichier texte `gainMax.txt` (soit la valeur 100) sera ensuite retournée au navigateur dans le corps de la réponse. Cette valeur sera enregistrée dans la variable `nouveauGain` par le biais de la propriété `responseText` de l'objet puis affectée à la zone de résultat à l'aide de la propriété `innerHTML` de l'élément résultat.

Conception du système

Téléchargement des codes sources des ateliers

Le moment est venu de passer à l'action. Les explications des différents ateliers vous permettront de créer vos différents scripts à partir de zéro. Cependant, vous avez la possibilité de télécharger tous les fichiers des exemples de cet ouvrage à votre disposition dans l'extension du livre sur le site *www.editions-eyrolles.com* (utilisez le nom de l'auteur comme mot clé pour accéder à l'extension de cet ouvrage).

Vous pourrez ainsi vous reporter à ces fichiers pour les comparer avec les vôtres en cas de problème, ou encore tester directement le fonctionnement de tous les ateliers directement sur ces ressources si vous ne désirez pas saisir vous-même les codes.

Ouvrez Dreamweaver (ou l'éditeur de votre choix) et sélectionnez le site Ajax que nous avons créé lors de l'installation de l'environnement de développement. Créez un nouveau fichier HTML et enregistrez-le tout de suite dans le répertoire `/ateliers/chap8/atelier8-1/` en le nommant `index.html`.

Organisation de vos ateliers

Nous vous suggérons de créer chaque atelier dans un répertoire différent portant le nom de l'atelier afin de bien isoler les fichiers utilisés dans le cadre de nos essais. Les deux fichiers (`index.html` et `gainMax.txt`) de ce premier atelier seront donc enregistrés dans un répertoire nommé « atelier8-1 ». Ainsi chaque atelier sera indépendant de l'autre, le seul élément qui ne sera pas dans le répertoire est l'image `logo.jpg` placée en haut de chaque page `index.html`, cette dernière étant commune à tous les ateliers, nous l'avons placé dans un répertoire nommé `/commun/`.

Dans ce premier exemple, nous avons choisi de ne pas lier la page HTML à une feuille de styles par souci de simplicité pour la mise en œuvre de votre première application. Néanmoins, nous allons structurer les différentes zones de la page HTML avec des balises `<div>` en prévision de la future feuille de styles que nous allons appliquer ensuite à cette page. Les deux éléments de la page (la zone `<span>` du résultat et le formulaire contenant le bouton JOUER) sont tous les deux insérés dans des balises `<div>` différentes et l'ensemble est regroupé dans une troisième balise `<div>` qui servira de conteneur pour la page (voir code 8-1).

Code 8-1 :

```
<div>
  <!--zone du résultat-->
  <div>
  Bravo, vous avez gagné <span id="resultat">0</span> euros
  </div>
  <!--zone du formulaire-->
  <div>
  <form>
   <input name="button" type="button" onclick="jouer();" value="JOUER" />
  </form>
  </div>
 </div>
```

Ressources sur les technologies associées

Nous avons regroupé dans la partie 4 de cet ouvrage plusieurs chapitres sur chacune des technologies utilisées dans les applications des ateliers. Nous vous invitons à vous y reporter pour obtenir des compléments d'informations si les explications qui accompagnent ces ateliers ne vous suffisent pas.

Placez-vous ensuite dans la balise `<head>` de la page et saisissez le code 8-2 ci-dessous.

Code 8-2 :

```
<script language="javascript" type="text/javascript">
/*-----------------------------MOTEUR AJAX-----------------------*/
  function jouer() {
 /*-------------------Config et envoi de la requête SYNCHRONE : */
  //création d'une requête uniquement pour Firefox
  objetXHR = new XMLHttpRequest();
  //Config. requête GET et Synchrone
  objetXHR.open("get","gainMax.txt", false);
  //envoi de la requête
  objetXHR.send(null);
/*--------------------------------Attente du retour SYNCHRONE : */
  //récupération du résultat renvoyé par le serveur
  var nouveauGain = objetXHR.responseText;
  //Affecte le nouveau gain à la zone résultat
  document.getElementById("resultat").innerHTML=nouveauGain;
 }
/*--------------------------FIN DU MOTEUR AJAX-------------------*/
 </script>
```

Enregistrez le fichier `index.html` après avoir terminé la saisie puis ouvrez un fichier texte (Depuis le menu de Dreamweaver : Fichier>Nouveau cliquez sur le bouton Autres à gauche de la fenêtre puis sélectionnez le type Texte dans la liste). Saisissez la valeur 100 dans le contenu du fichier et enregistrez-le sous le nom `gainMax.txt` (voir figure 8-2).

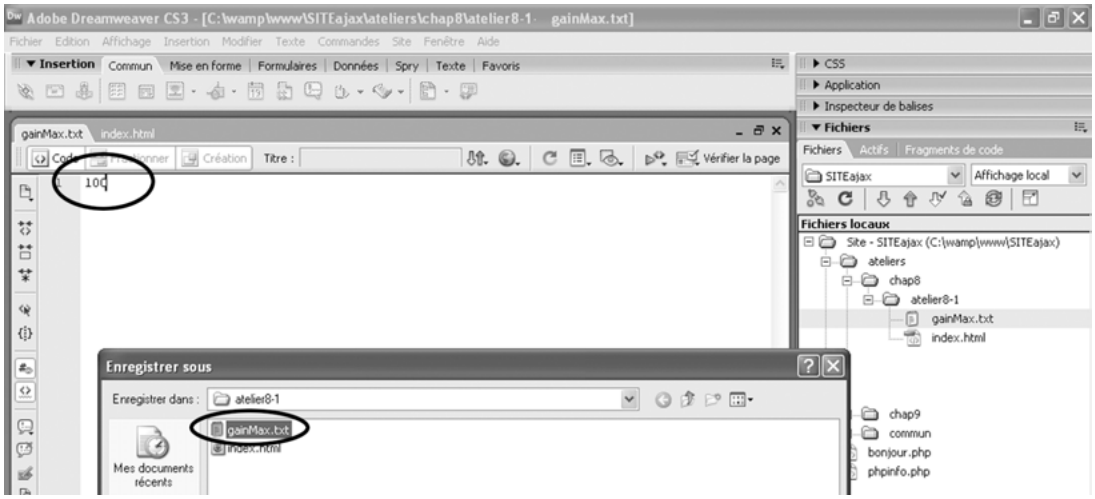


Figure 8-2

Création du fichier texte gainMax.txt

Pour ce premier script, il est intéressant d'expliquer d'une façon détaillée le code de cette page HTML afin de bien comprendre le mécanisme de cette application Ajax.

Dans la partie visible de page HTML (balise `<body>`, voir le code 8-1), la zone dans laquelle s'affichera le résultat est délimitée par une balise `<span>` à laquelle nous avons ajouté un identifiant `resultat` qui permettra ensuite d'en modifier le contenu à l'aide de la propriété `innerHTML` de l'élément ainsi constitué. Lors de son premier affichage, le contenu de cette balise est initialisé avec la valeur 0. Cette valeur sera ensuite remplacée par la valeur contenue dans le fichier texte (100).

```
<span id="resultat">0</span>
```

Un peu plus bas, un formulaire a été ajouté afin d'insérer un bouton de commande JOUER dans la page HTML. L'élément `<input>` est lié à un gestionnaire d'événement qui permettra d'appeler la fonction contenant le moteur Ajax (`onclick="jouer()"`). L'utilisateur pourra ainsi afficher le contenu retourné par le fichier texte par un simple clic sur ce bouton. À noter que la valeur du fichier texte étant toujours la même, il est nécessaire d'appuyer sur le bouton d'actualisation du navigateur (ou d'utiliser le raccourci clavier F5) pour revenir à l'état initial de la page avant d'appuyer de nouveau sur le bouton JOUER.

```
<form>
 <input type="button" onclick="jouer();" value="JOUER" />
</form>
```

Passons maintenant à la fonction `jouer()` contenant le moteur Ajax. La première instruction de cette fonction permet d'instancier la classe XMLHttpRequest et de créer un objet nommé `objetXHR`.

```
objetXHR = new XMLHttpRequest();
```

Une fois l'objet créé, il faut ensuite le configurer avec sa méthode `open()`. Trois paramètres seront nécessaires à sa configuration. Le premier permet d'indiquer que nous désirons utiliser la méthode `GET` pour émettre la requête HTTP. Le second précise le fichier ciblé par la requête, soit le fichier texte `gainMax.txt` pour ce premier exemple. Enfin le troisième paramètre est initialisé avec la valeur `false` afin d'indiquer que la requête devra être en mode synchrone.

```
objetXHR.open("get","gainMax.txt", false);
```

Maintenant que l'objet est créé et configuré, il ne reste plus qu'à l'envoyer. Pour cela, nous utiliserons la méthode `send()` de l'objet. À noter que l'argument de cette méthode sera utilisé lorsque nous aurons une méthode `POST` avec des paramètres à communiquer au serveur. Comme ce n'est pas le cas de notre exemple, ce paramètre sera configuré avec la valeur `null`.

```
objetXHR.send(null);
```

La requête étant synchrone, la communication restera ouverte dans l'attente de la réponse comme dans le cas d'une requête HTTP traditionnelle. Nous pouvons donc placer les instructions de traitement de la réponse immédiatement après l'envoi de la requête. La première instruction de ce traitement permet d'affecter la réponse texte à une variable nommée `nouveauGain`. Nous utilisons pour cela la propriété `responseText` de l'objet XHR.

```
var nouveauGain = objetXHR.responseText;
```

Nous arrivons maintenant au terme de la fonction du moteur Ajax. En effet, nous disposons de la valeur de la réponse côté client, il ne nous reste plus qu'à l'affecter à la zone

résultat afin qu'elle remplace la valeur 0 configurée par défaut. Pour cela, nous utiliserons la méthode `getElementById()` qui permet de référencer l'élément de la balise `<span>` par son identifiant `resultat`. Puis nous exploiterons la propriété `innerHTML` qui permettra de remplacer le contenu de l'élément par la valeur 100 sauvegardée précédemment dans la variable `nouveauGain`.

```
document.getElementById("resultat").innerHTML=nouveauGain;
```

Test du système

Pour tester le système, vous devez commencer par ouvrir la page `index.html` dans le Web Local avec le navigateur Firefox. Pour cela, plusieurs solutions s'offrent à vous. La première est la plus rapide mais nécessite d'avoir configuré le serveur d'évaluation dans la définition initiale du site Ajax (revoir si besoin le chapitre 7). Assurez-vous avant tout que Wamp5 est bien démarré (un icône en forme de demi cercle doit apparaître dans la zone d'état en bas à droite de l'écran de votre ordinateur).

Ouvrez la page à tester (`index.html` dans notre cas) dans Dreamweaver puis cliquez sur l'icône Aperçu/débogage (bouton ayant la forme d'une planète bleue) situé dans le menu de l'éditeur de fichier puis sélectionnez Firefox dans la liste des navigateurs (par la suite, nous vous conseillons d'utiliser le raccourci clavier F12). Le navigateur Firefox doit alors s'ouvrir et afficher la page concernée.

L'autre solution est plus longue mais pourra être utilisée dans tous les cas, même si vous n'utilisez pas Dreamweaver ou si le serveur d'évaluation n'a pas encore été configuré. Déroulez le menu du manager Wamp et sélectionnez l'option `localhost`. La page d'accueil du Web Local de Wamp doit alors s'ouvrir dans le navigateur Firefox (préalablement configuré comme le navigateur par défaut dans la procédure d'installation de Wamp5). Dans la zone Vos projets cliquez sur le petit répertoire nommé `SITEajax`. La racine de notre site Ajax n'ayant pas de fichier d'index, la liste des fichiers et répertoires s'affiche dans la nouvelle page. Cliquez successivement sur les répertoires `atelier`, `chap8` puis `atelier8-1`. La page `index.html` doit alors s'ouvrir comme dans le cas de la première solution.

Fonctionne uniquement sur Firefox

L'exemple de cet atelier fonctionne uniquement sur le navigateur Firefox (ou les autres navigateurs compatibles avec l'objet XMLHttpRequest). Si toutefois vous désirez le faire fonctionner dès maintenant avec Internet Explorer (version supérieure à 5.0), il suffit de remplacer la syntaxe d'instanciation de l'objet XHR, soit actuellement `new XMLHttpRequest()`, par celle-ci : `new ActiveXObject("MSXML2.XMLHttp")`. Soyez rassuré, nous présenterons plus loin le script à utiliser pour que vos applications puissent fonctionner avec tous les types de navigateur.

Nous pouvons remarquer dans la page `index.html` qui est maintenant affichée dans le navigateur que la valeur du gain est égale à 0. Si vous cliquez sur le bouton JOUER, vous déclenchez alors la requête synchrone et la valeur du gain doit être immédiatement remplacée par celle du gain maximum stockée dans le fichier `gainMax.txt` (soit 100).

L'intérêt de ces ateliers est surtout d'observer le fonctionnement du système afin de mieux comprendre les rouages du mécanisme d'une application Ajax et pour vous permettre par la suite de diagnostiquer un éventuel problème et dépanner une application plus complexe. Pour ces raisons nous allons activer l'extension Firebug à chaque test en cliquant sur l'icône placé en bas et à droite du navigateur.

Comme nous l'avons déjà vu dans le chapitre consacré à Firefox et à ses extensions, Firebug permet d'effectuer de multiples opérations lors du test d'une application Ajax. Par exemple, si vous cliquez sur l'onglet HTML (voir repère 3 de la figure 8-3), vous pouvez afficher le contenu des éléments en mémoire en déroulant successivement les différents éléments de la page. Attention, il s'agit des contenus en mémoire (représentation du DOM sous forme de balises) et non d'un simple affichage du code initial de la page. Ainsi, dans notre cas, après l'envoi de la requête, la valeur dans la zone de résultat est égale à 100 (voir repère 2 de la figure 8-3) et non à 0 (valeur qui serait visible dans cette même fenêtre avant l'action sur le bouton JOUER ou dans le cas de l'affichage du code source traditionnel d'une page). De plus si vous survolez avec votre souris l'un de ces éléments dans la fenêtre de Firebug, la zone correspondante dans l'écran du navigateur est alors mise en évidence (voir repère 1 de la figure 8-3).

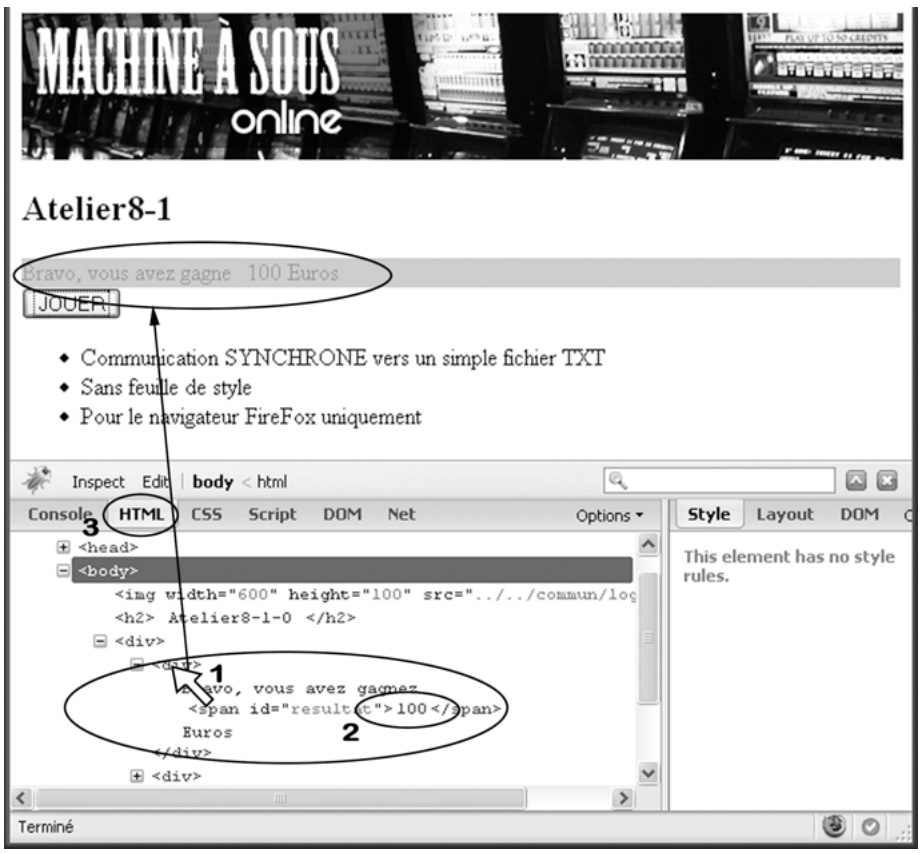


Figure 8-3

Affichage du contenu des éléments de la page HTML en mémoire à l'aide Firebug

Une autre fonctionnalité très intéressante de Firebug est de pouvoir observer les informations échangées entre le navigateur et le serveur. Vous pouvez ainsi faire apparaître tous les objets externes qui constituent votre page HTML (structure brute de la page HTML, images, feuille CSS, fichier JS externe…) lors de son appel initial et connaître le temps de chargement de chacun des objets individuellement. Mais cela est encore plus intéressant avec une application Ajax lors de l'envoi d'une requête car, si vous pouvez aussi

connaître le temps de traitement de la requête, vous pouvez surtout afficher le contenu de la requête HTTP (et de tous ses en-têtes) ainsi que le contenu de la réponse HTTP correspondante. Pour cela, il faut cliquer sur l'onglet Net de Firebug (voir repère 1 de la figure 8-4) puis sur le bouton Clear (voir repère 2 de la figure 8-4) pour nettoyer les chargements précédents. Pour simuler le chargement initial de la page, nous allons cliquer sur le bouton de réactualisation de la page (voir repère 3 de la figure 8-4, ou plus simplement à l'aide du raccourci F5). Vous devriez voir apparaître les deux objets constituant la page Web de notre exemple avec leurs temps de chargement respectifs, soient la structure brute de la page HTML et le chargement de l'image placée en tête de notre page (voir repère 4 de la figure 8-4).

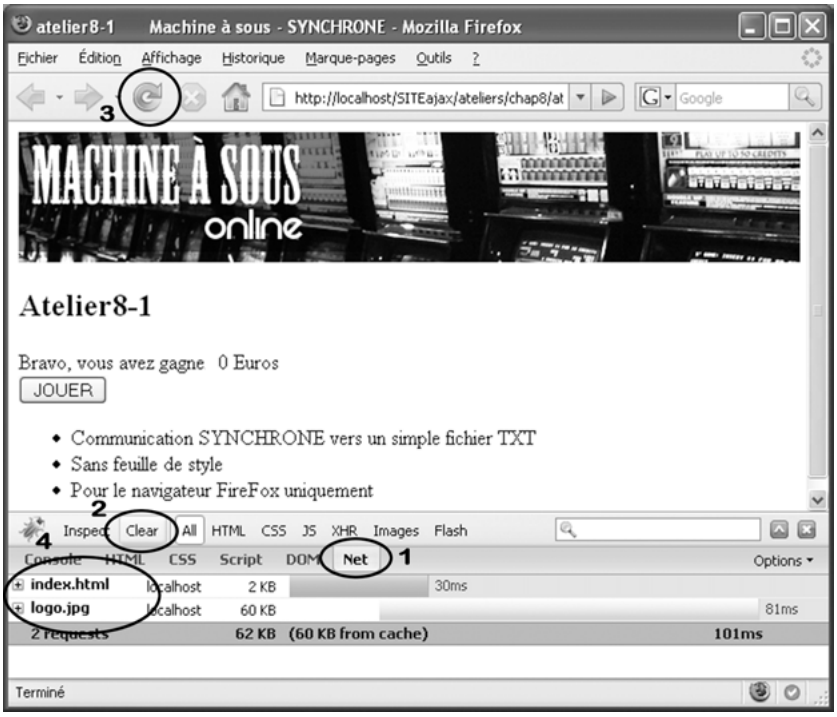


Figure 8-4

Affichage des temps de chargement des objets constituant une page Web à l'aide de Firebug

Pour l'instant, nous n'avons pas encore déclenché la requête Ajax. Si vous cliquez maintenant sur le bouton JOUER (voir repère 1 de la figure 8-5), le fichier texte doit alors être chargé dans le navigateur et apparaître à la suite de la liste des deux précédents objets de la page Web (voir repère 3 de la figure 8-5) et le résultat affiché dans la page doit prendre la valeur 100 (voir repère 2 de la figure 8-5).

Comme nous vous l'avons annoncé précédemment, Firebug permet aussi d'afficher le contenu de tous les en-têtes et le corps d'une réponse HTTP. Pour cela, cliquez sur le nom du fichier texte chargé lors de la requête HTTP (`gainMax.txt`, voir repère 1 de la figure 8-6). Une nouvelle zone doit alors apparaître en dessous du nom du fichier (utilisez votre souris pour redimensionner la fenêtre si besoin). Par défaut, c'est l'onglet des en-têtes qui doit être actif (`Headers`, voir repère 2 de la figure 8-6) mais si vous cliquez sur le second

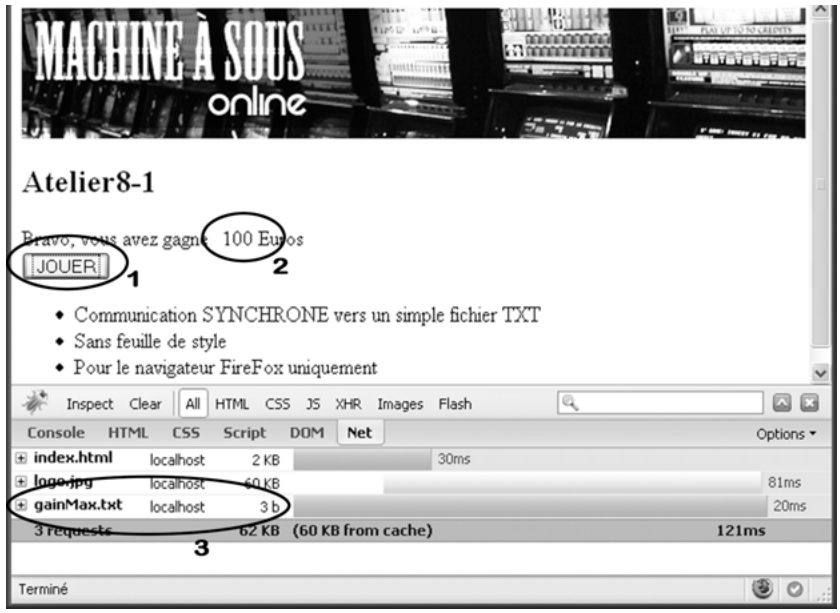


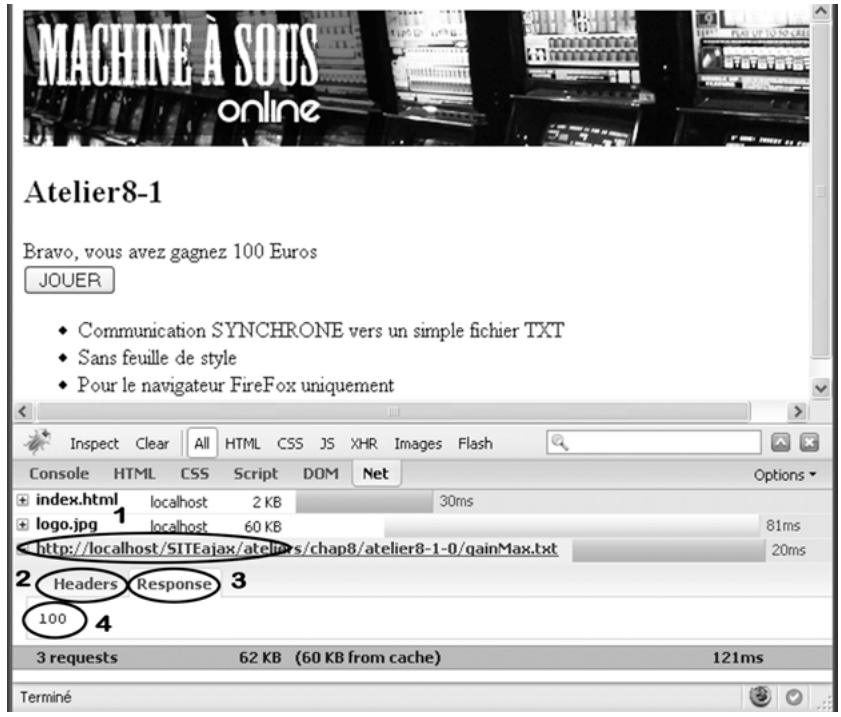
Figure 8-5

Affichage du temps de chargement du fichier texte lors de l'envoi de la requête Ajax à l'aide de Firebug

onglet (`Response`, voir repère 3 de la figure 8-6) vous pouvez alors voir le contenu du corps de la réponse (soit 100 dans notre exemple, voir repère 4 de la figure 8-6).

Figure 8-6

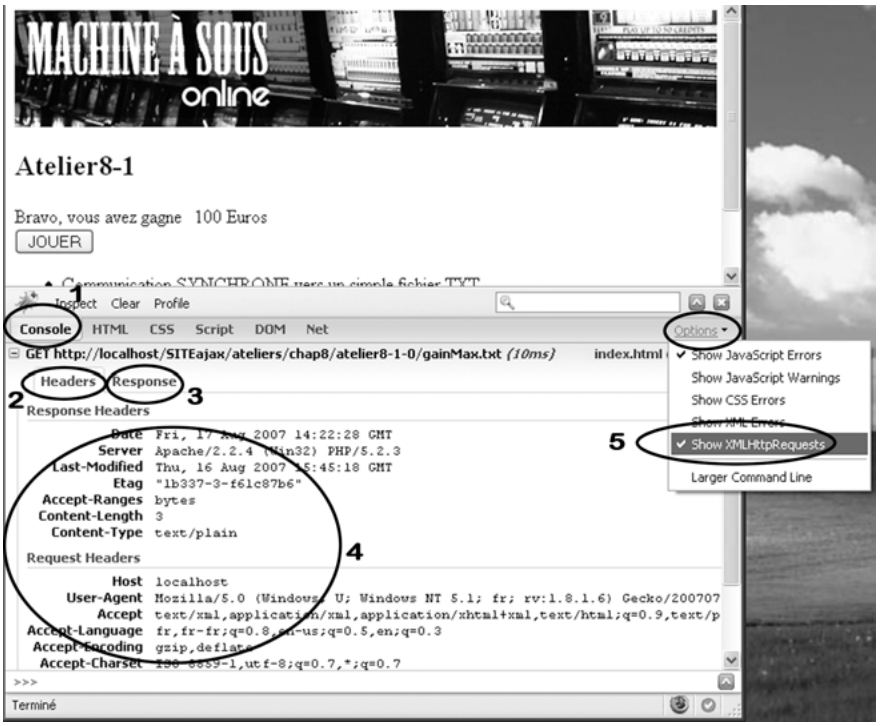
Affichage du contenu du corps de la réponse HTTP à l'aide de l'onglet Net de Firebug



Une autre alternative pour observer les en-têtes et le corps de la réponse consiste à utiliser la console de Firebug. La console permet d'avoir un historique de toutes les erreurs JavaScript, CSS ou XML mais permet aussi de suivre les transferts HTTP générés par un objet XHR. Pour utiliser la console, cliquez sur l'onglet Console (voir repère 1 de la figure 8-7) puis sur la requête concernée. Vous pouvez ensuite passer de l'affichage des en-têtes au corps de la réponse en utilisant des onglets similaires à ceux utilisés précédemment avec la fonctionnalité Net (voir repères 2 et 3 de la figure 8-7). Par exemple le repère 4 de la figure 8-7 illustre l'affichage des en-têtes de la réponse HTTP du fichier `gainMax.txt`. Si vous ne voyez pas apparaître ces informations dans la console, assurez vous que cette fonctionnalité est cochée dans le menu des options situé en haut à droite de la fenêtre (voir repère 5 de la figure 8-7).

Figure 8-7

Affichage des en-têtes de la réponse HTTP à l'aide la console de Firebug



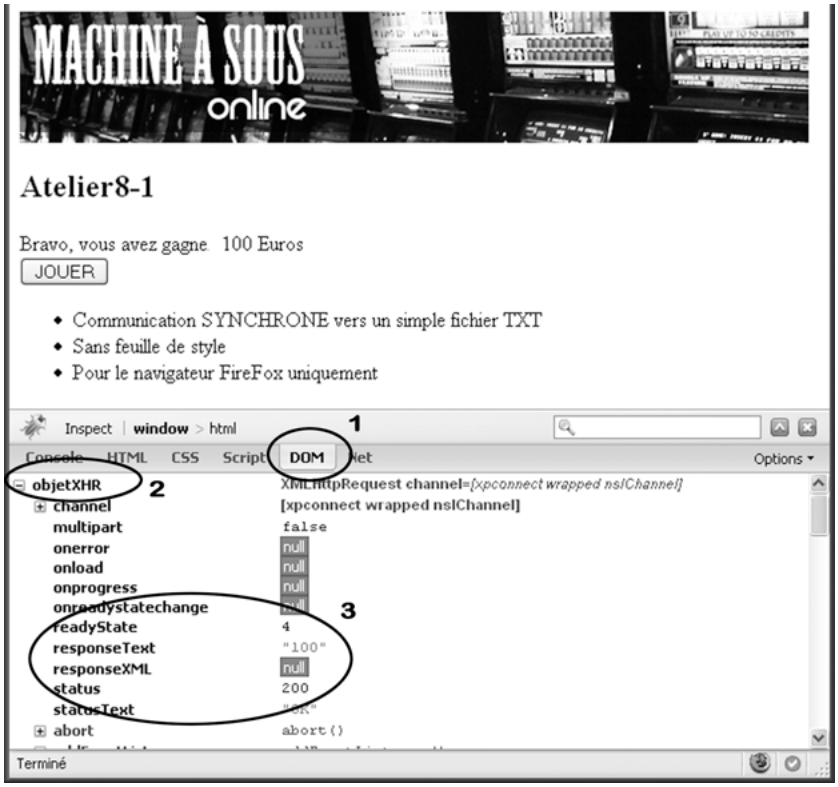
La fonctionnalité permettant d'explorer le DOM de la page est aussi très intéressante à parcourir pour comprendre le fonctionnement d'une requête Ajax. En effet, l'objet XHR étant désormais créé, il est visible dans le DOM de la page HTML et cela va nous permettre d'observer toutes les méthodes et propriétés de cet objet. Pour accéder à ces informations, il suffit de cliquer sur l'onglet DOM de Firebug (voir repère 1 de la figure 8-8) puis de dérouler l'objet XHR en cliquant sur le petit plus qui précède son nom (`objetXHR`, voir repère 2 de la figure 8-8). Vous pouvez alors consulter les valeurs des différentes propriétés de l'objet. L'évolution de certaines de ces valeurs, telles que `readyState`, `status` ou encore `responseTxt`, seront précieuses pour suivre les étapes et les résultats retournés par la réponse HTTP (voir repère 3 de la figure 8-8).

Ce premier système fonctionne avec le mode synchrone de l'objet XHR (ce mode est évidemment très peu utilisé dans les applications Ajax professionnelles). Pour simplifier le code, son usage est limité pour l'instant au navigateur Firefox de même qu'il est dénué

de mise en forme puisque la structure des éléments de la page HTML s'appuie sur une hiérarchie de balises `<div>` sans style spécifique.

Figure 8-8

Affichage des propriétés de l'objet XHR à l'aide de Firebug



Nous remarquons aussi que le joueur gagne toujours le gain maximum (soit 100 euros). En effet, le fichier texte est statique et contient toujours la même valeur. Cependant, nous pourrions imaginer que la valeur dans ce fichier puisse être modifiée manuellement (ou par une interface d'administration en ligne) par l'administrateur du site et, dans ce cas, la valeur du gain serait différente d'une mise à jour à l'autre.

L'intérêt de ce premier système est donc son extrême simplicité qui vous a permis de créer rapidement votre première application Ajax (même si dans ce cas, le A et le X du nom Ajax ne sont pas justifiés…) en intégrant un moteur Ajax basique dans une page Web.

Atelier 8-2 : requête synchrone sur un fichier texte avec une feuille de styles

Composition du système

Dans ce nouveau système, nous avons lié une feuille de styles externe afin de mettre en forme les différents conteneurs `<div>` de la page HTML. Nous profiterons de cette modification pour démontrer que le fait d'ajouter des identifiants à chaque balise `<div>` permet aussi de les manipuler dynamiquement avec JavaScript (rendre visible ou invisible, changer leur mise en forme…).

Cette structure est composée :

- d'une page HTML (`index.html`) identique à celle de l'atelier précédent mais dans laquelle nous avons attribué un identifiant à chaque balise `<div>` et ajouté une instruction pour lier la page HTML à la feuille de styles ;
- d'un simple fichier texte (`gainMax.txt`) identique au précédent ;
- d'une feuille de styles (`style.css`) dans laquelle nous avons défini les différents styles correspondant aux identifiants des balises `<div>`.

Fonctionnement du système

Le fonctionnement du système est semblable à celui de l'atelier précédent hormis le fait que la mise en forme des éléments `<div>` sera effectuée par une feuille de styles externe et que le message présentant la somme gagnée n'apparaîtra à l'écran que lorsque le résultat de la réponse HTTP sera disponible.

Conception du système

Nous allons commencer par modifier la page HTML en attribuant des identifiants spécifiques à chaque balise `<div>`. Pour cela, ouvrez la page HTML précédente (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/atelier8-2/` (profitez-en pour faire ensuite la même opération avec le fichier `gainMax.txt` qui est lié à cette page). Ajoutez ensuite les différents attributs `id` et leur nom dans les balises `<div>` concernées avec l'éditeur de Dreamweaver en mode code (voir code 8-1 et le repère 1 de la figure 8-9).

Code 8-3 :

```
<div id="page">
  <!--zone du résultat-->
  <div id="info">
   Bravo, vous avez gagné
   <span id="resultat">0</span>
   euros
  </div>
  <!--zone du formulaire-->
  <div id="formulaire">
  <form method="GET">
   <input name="button" type="button" onClick="jouer();" value="JOUER" />
  </form>
  </div>
 </div>
```

Placez-vous ensuite dans la balise `<head>` de la page HTML et ajoutez la ligne de code ci-dessous (voir code 8-4 et le repère 2 de la figure 8-9) qui permettra de lier les identifiants précédemment ajoutés aux balises `<div>` avec la feuille de styles (`style.css`) que nous allons créer ensuite.

Code 8-4 :

```
<link rel="stylesheet" type="text/css" href="style.css" />
```

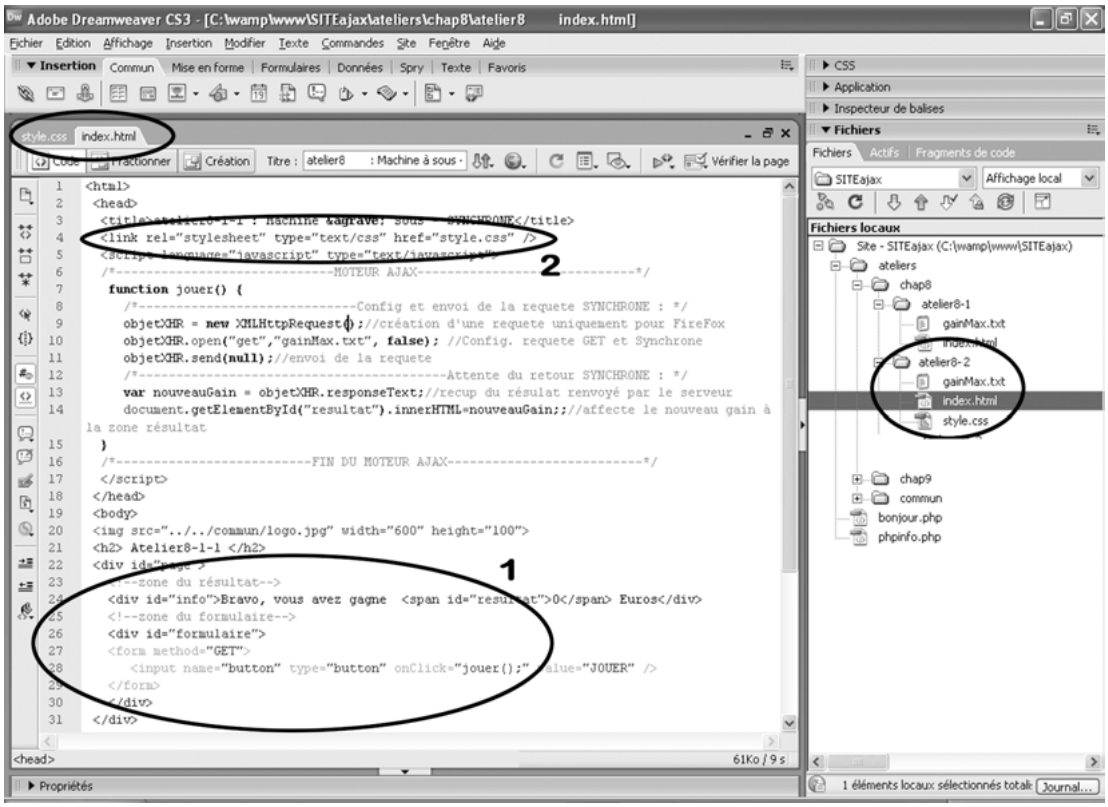


Figure 8-9

Code de la page index.html après l'ajout de la feuille de styles style.css

Créez une nouvelle feuille de styles avec Dreamweaver (depuis le menu Fichier>Nouveau, cliquez sur le bouton Pages vierges puis sélectionnez CSS dans la liste proposée). Saisissez ensuite les différentes règles de style ci-dessous (voir code 8-5). Enregistrez la feuille de styles sous le nom `style.css` dans le même répertoire que la page HTML.

Code 8-5 :

```
body, h1, h2, p { font-size: 1em; margin: 0; padding: 0; }
body {
 font-family: Verdana, Geneva, Arial, sans-serif;
 text-align: center;
}
#page {
  position: relative;
  margin: 0 auto;
  width: 600px;
  height: 200px;
  border-top: medium solid #ff0000;
  border-bottom: medium solid #ff0000;
}
#info {
 position: absolute;
 left: 100px;
 top: 30px;
}
```

```
#resultat {
 font-weight: bold;
}
#formulaire {
 position: absolute;
 left: 290px;
 top: 100px;
}
```

Les différentes règles de style permettront d'améliorer la présentation de la page. Ainsi, la règle de style `page` permettra de créer un conteneur principal et d'afficher un trait rouge en haut et en bas de cette zone centrale pour bien la délimiter. Les règles de style `info` et `formulaire` permettront de placer le bouton JOUER et la zone de résultat d'une manière relative au conteneur `page`. Enfin, la règle de style `resultat` permettra d'afficher le résultat placé dans la balise `<span>` en gras pour bien le mettre en évidence.

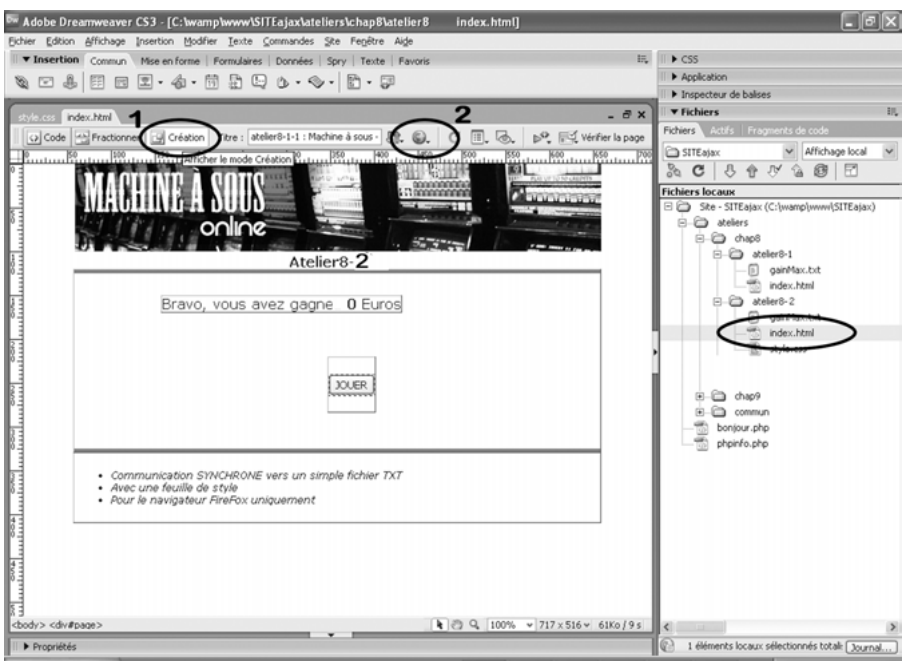
Ressources sur les CSS

Pour plus d'informations sur les styles, reportez-vous au chapitre 17 de cet ouvrage.

Après avoir saisi et enregistré la feuille de styles, revenez à la page HTML et passez la fenêtre du document en mode Création (voir repère 1 de la figure 8-10). Si tout est correct, les styles devraient alors être appliqués à la page directement dans la fenêtre document de Dreamweaver (voir figure 8-10). Vous pouvez aussi demander un aperçu de la page dans le navigateur (à l'aide du bouton `Aperçu`, voir repère 2 de la figure 8-10, ou plus simplement en utilisant le raccourci clavier F12) pour vous assurer que tous les styles fonctionnent aussi dans le navigateur.

Figure 8-10

Affichage de la page index.html en mode Création afin de vérifier que les styles sont bien appliqués à la page HTML



Nous allons maintenant modifier la règle de style de l'identifiant de la zone de résultat pour la rendre invisible lors du chargement initial de la page. Pour cela, ouvrez la feuille

de styles et ajoutez la directive `visibility: hidden` (à noter que nous aurions aussi pu utiliser la directive `display: none`) à la règle de style dont l'identifiant est `info` (voir code 8-6). Enregistrez ensuite votre feuille de styles et retournez dans la page HTML en mode Création. La zone de résultat et son message doivent maintenant avoir disparu.

Code 8-6 :

```
#info {
   position: absolute;
   left: 100px;
   top: 30px;
   visibility: hidden
}
```

Il nous reste maintenant à ajouter l'instruction qui rendra visible la balise dont l'identifiant se nomme `info` dès que la valeur du résultat sera connue. Pour cela nous allons manipuler le DOM en utilisant la méthode `getElementById()` pour référencer l'élément de la balise `<div>` à l'aide de son identifiant `info`. Nous agirons ensuite sur cet élément via la propriété `style.visibility` pour affecter à la propriété `visibility` sa nouvelle valeur `visible` qui, comme son nom l'indique, rendra visible l'élément concerné (voir code 8-7).

Code 8-7 :

```
/*-----------------------------MOTEUR AJAX-----------------------*/
  function jouer() {
/*--------------------Config et envoi de la requête SYNCHRONE : */
     objetXHR = new XMLHttpRequest();
     objetXHR.open("get","gainMax.txt", false);
     objetXHR.send(null);
/*--------------------------------Attente du retour SYNCHRONE : */
     var nouveauGain = objetXHR.responseText;
     document.getElementById("resultat").innerHTML=nouveauGain;
     document.getElementById("info").style.visibility="visible";
 }
 /*--------------------------FIN DU MOTEUR AJAX----------------*/
```

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 de Dreamweaver (ou avec la méthode de votre choix, revoir la procédure de test de l'atelier précédent). Cette fois, les éléments doivent être mis en page selon les paramètres de la feuille de styles à laquelle ils sont liés et aucun message de résultat par défaut ne doit s'afficher au chargement de la page. Cliquez ensuite sur le bouton JOUER. Le résultat maximum accompagné de son message introductif doivent maintenant s'afficher à l'écran (voir repère 2 figure 8-11).

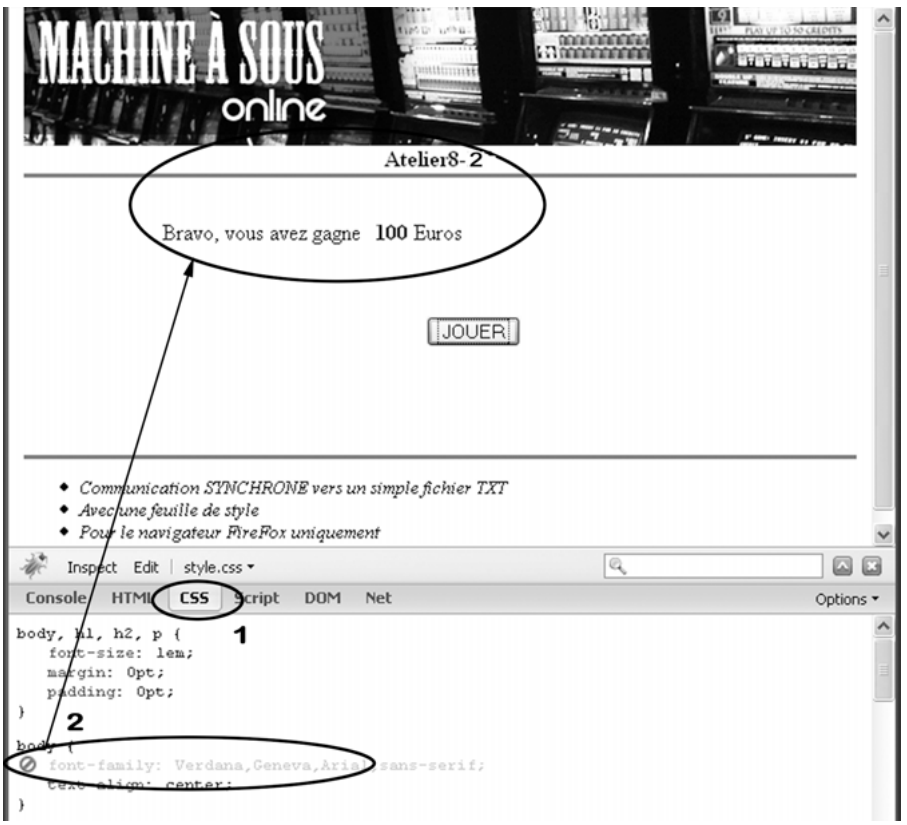
Figure 8-11

Test de la page HTML de l'atelier 8-2. Le message du résultat (repère 2) doit s'afficher uniquement après avoir appuyé sur le bouton JOUER (repère 1).

Nous allons aussi profiter de ce test pour découvrir les fonctionnalités que propose Firebug pour mettre au point une feuille de styles. Commençons par la fonctionnalité qui permet d'activer ou de désactiver certaines propriétés d'un style ou d'en modifier leur nom ou valeur. Pour cela, activez Firebug (cliquez sur le bouton vert en bas à droite du navigateur) et cliquez sur l'onglet CSS de la fenêtre de Firebug (voir repère 1 de la figure 8-12). Si vous survolez les différentes propriétés avec la souris, vous pouvez observer qu'une icône d'interdiction grisée apparaît à gauche de chaque propriété. Si vous cliquez sur l'une des propriétés, l'icône devient rouge et la propriété concernée est inhibée (voir le repère 2 de la figure 8-12 où nous avons inhibé la propriété `font-family` de la balise `<body>`). De même, dans cette fenêtre, vous pouvez changer en direct les noms et valeurs de chaque propriété. Il suffit pour cela de cliquer dessus et de saisir la nouvelle valeur, l'incidence de votre modification sera instantanément reportée sur l'écran du navigateur (pour tester cette fonctionnalité, essayez de changer la valeur de la propriété `font-size` du body par exemple).

Figure 8-12

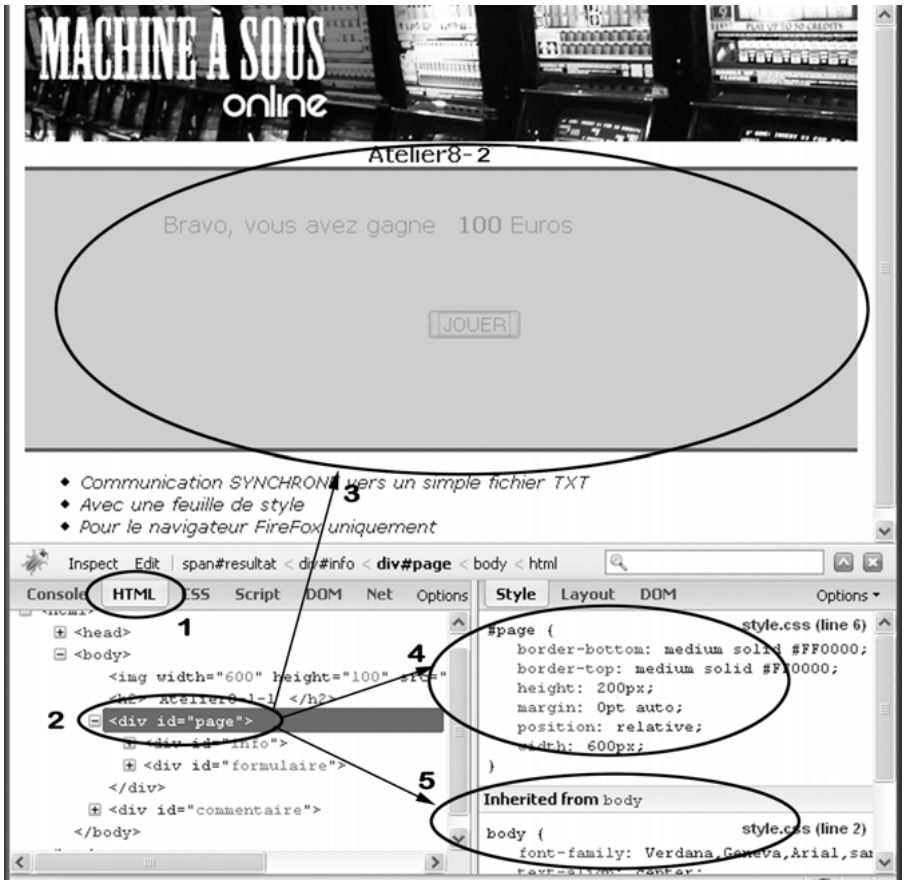
Fonctionnalité de Firebug pour mettre au point une feuille de styles



Firebug permet aussi d'accéder aux mêmes fonctionnalités de gestion des styles depuis l'onglet `HTML` (les zones 4 et 5 de la figure 8-13 disposent des mêmes fonctionnalités que la fenêtre CSS que nous venons de décrire). L'intérêt de passer par l'onglet HTML (voir repère 1 de la figure 8-13) est que le simple survol d'un élément concerné par un style (voir repère 2 de la figure 8-13) dans la fenêtre `HTML` affiche les propriétés du style correspondant (voir repère 4 de la figure 8-13) et les éventuels styles hérités (voir repère 5 de la figure 8-13) dans la fenêtre latérale et met en exergue la zone concernée dans l'écran du navigateur (voir repère 3 de la figure 8-13).

Figure 8-13

L'onglet HTML permet aussi d'accéder aux fonctionnalités de mise au point des styles.



Atelier 8-3 : requête HTTP traditionnelle avec un traitement PHP et une feuille de styles

Composition du système

Nous désirons maintenant remédier au fait que la valeur retournée par la réponse est toujours la même. Pour cela, nous devons adresser notre requête à un fichier PHP qui pourra renvoyer une valeur différente à l'issue de son traitement. S'agissant ici d'un jeu de hasard nous utiliserons une fonction de génération d'une valeur aléatoire pour définir la valeur du résultat du traitement. Cependant, plutôt que de passer directement à une application Ajax dont l'objet XHR serait configuré pour cibler un fichier PHP, nous avons trouvé opportun de profiter de l'occasion pour vous présenter dans cet atelier ce que serait un système équivalent réalisé avec une requête HTTP traditionnelle.

Cette structure est composée :

- d'une page PHP (`index.php`) identique à la page HTML de l'atelier précédent mais dans laquelle nous avons remplacé le moteur Ajax par un script PHP de traitement intégré directement dans la zone de résultat ;
- d'une feuille de styles (`style.css`) identique à la précédente.

Fonctionnement du système

Dans ce système, le fonctionnement sera un peu différent car nous ne chargerons pas initialement une page HTML contenant un moteur Ajax, mais un fichier contenant un script PHP destiné à afficher le résultat du traitement directement dans la page lors de la soumission du formulaire. Le formulaire contenant le bouton JOUER devra donc être complètement configuré cette fois, afin d'indiquer que le fichier cible sera la page dans laquelle il se trouve. Ainsi, lorsque nous allons cliquer sur le bouton JOUER, une requête `GET` sera envoyée à la page dans laquelle nous nous trouvons. Le script PHP intégré dans la zone de résultat devra ensuite détecter cette soumission et générer un traitement local pour définir le nouveau résultat et l'afficher dans la page.

Conception du système

Ouvrez la page HTML de l'atelier précédent (`index.html`) et sauvegardez-la sous le nouveau nom `index.php` (attention, il s'agit maintenant de l'extension `.php` et non plus `.html`) dans un nouveau répertoire nommé `/atelier8-3/`. Passez la fenêtre du document en mode Code et supprimez l'insertion JavaScript complète du moteur placée dans le haut de la page. Placez-vous ensuite dans la balise `<form>` puis supprimez le gestionnaire `onclick="jouer();"`, ajoutez l'attribut `method="get"` et l'attribut `action="index.php"` afin d'indiquer que c'est la page courante qui devra réceptionner la requête. Modifiez ensuite le type de la balise `<input>` avec `type="submit"` pour transformer le bouton JOUER en bouton de soumission du formulaire (voir code 8-8 et le repère 1 de la figure 8-14).

Code 8-8 :

```
<form method="get" action="index.php">
   <input name="button" type="submit" value="JOUER" />
</form>
```

Placez-vous ensuite dans la balise résultat et sélectionnez la valeur par défaut (0), puis remplacez cette valeur par le code PHP du code 8-9 ci-dessous (voir le repère 2 de la figure 8-14).

Code 8-9 :

```
<?php
  if(isset($_GET['button']))
  {
     $resultat = rand(0,100);
     echo $resultat ;
  }else{
     echo "0";
  }
?>
```

La première ligne de ce script PHP permet de détecter si le formulaire a été soumis.

```
if(isset($_GET['button']))
```

Pour cela, nous utilisons une structure de test `if()` et la fonction `isset()` appliquée à la variable `$_GET['button']` qui permet de vérifier si l'élément de formulaire nommé `button` (soit le bouton JOUER) existe. Ainsi, le script qui suit ne sera exécuté que lorsque l'utilisateur appuiera sur le bouton JOUER alors que lors du chargement initial de la page, c'est le script contenu dans la structure `else` (soit l'instruction `echo "0";`) qui s'exécutera, affichant ainsi la valeur initiale 0 à la place du montant du gain.

Dans le bloc conditionné par l'action sur le bouton JOUER, nous utiliserons la fonction `rand()` qui permet de calculer une valeur aléatoire (les deux paramètres de la fonction indiquent la plage possible du résultat, soit entre 0 et 100).

```
$resultat = rand(0,100);
```

La valeur ainsi obtenue sera affectée à une variable nommée `$resultat` qui sera ensuite affichée dans la zone de résultat grâce à la seconde ligne :

```
echo $resultat;
```

La structure `else` placée après ce premier bloc permet d'afficher la valeur 0 par défaut lors du chargement initial de la page PHP :

```
}else{
  echo "0";
}
```

Ressources sur le PHP

Pour plus d'informations sur la syntaxe du PHP, reportez-vous au chapitre 21 de cet ouvrage.

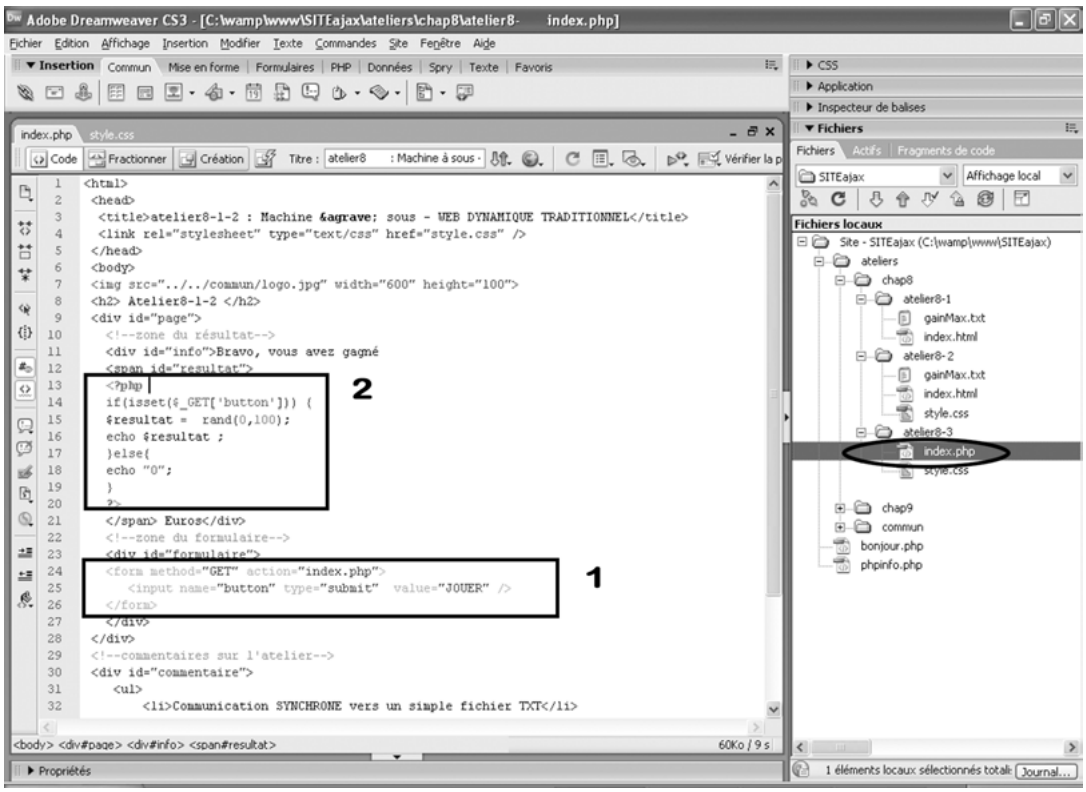


Figure 8-14

Modification de la page index.php

Les modifications de la page `index.php` sont terminées. Avant les tests, il faut cependant modifier la feuille de styles `style.css` afin d'indiquer que la zone résultat doit maintenant

être affichée dès le chargement initial de la page. Pour cela, nous allons modifier la valeur de la propriété `visibility` du style `info` en lui affectant la valeur `visible` directement, au lieu de la valeur `hidden` qui avait été configuré dans l'atelier précédent (voir code 8-10).

Code 8-10 :

```
#info {
  position: absolute;
  left: 100px;
  top: 30px;
  visibility: visible;
}
```

Test du système

Ouvrez la page `index.php` dans le navigateur Firefox en appuyant sur la touche F12 de Dreamweaver. Activez Firebug (cliquez sur le bouton vert en bas du navigateur) et sélectionnez la fonction Net (voir repère 1 de la figure 8-15). Dans la fenêtre Net de Firebug, nous pouvons observer que les trois objets constituant la page Web sont chargés correctement (voir repère 2 de la figure 8-15) et apprécier le temps de chargement de chacun de ces éléments. Nous constatons aussi que la valeur du résultat est bien initialisée à 0 (voir repère 3 de la figure 8-15) et qu'aucun paramètre n'est ajouté actuellement à l'URL de la page (voir repère 4 de la figure 8-15).

Figure 8-15

Test de la page index.php : chargement initial de la page

Cliquez maintenant sur le bouton JOUER et observez bien la fenêtre Net de Firebug. Vous devriez voir apparaître ponctuellement un appel à la page `index.php` à la suite des objets déjà chargés qui démontre qu'une requête vers cette même page a bien été générée (voir repère 1 de la figure 8-16). Cette apparition est furtive car dans notre cas le serveur renvoie la page Web complète qui appellera sa feuille de styles dès qu'elle sera chargée dans le navigateur, les deux nouveaux fichiers (la page `index.php` et la feuille de styles `style.css`) remplacent donc presque instantanément les fichiers précédemment chargés. Observez au passage que l'image ne sera pas rechargée car elle est désormais stockée dans le cache du navigateur. De même, l'URL de la page est maintenant suivie du paramètre `button` envoyé avec la méthode `GET` lors de la soumission du formulaire (voir

repère 2 de la figure 8-16). Enfin une valeur de gain aléatoire a pris place dans la zone de résultat et si vous renouvelez votre action sur le bouton JOUER, une valeur différente du gain doit s'afficher à chaque essai.

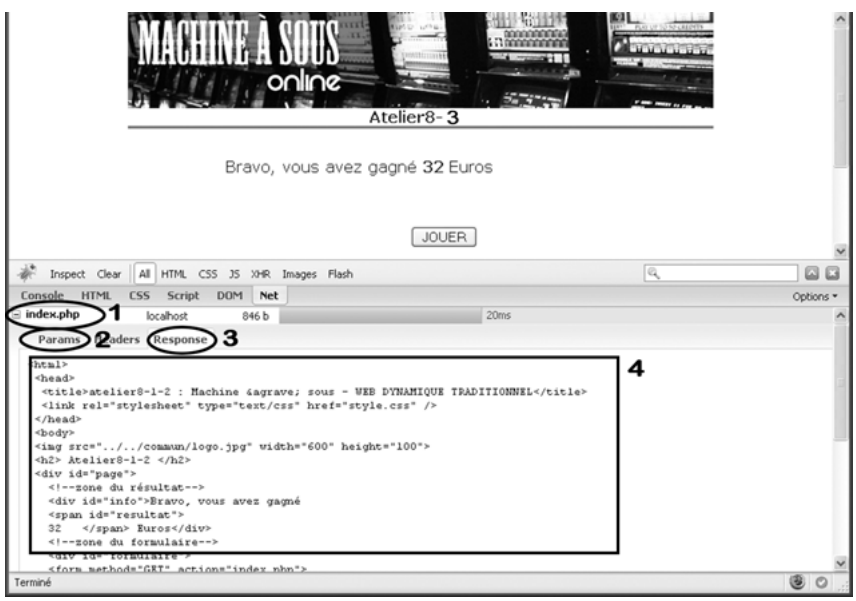
Figure 8-16

Test de la page index.php : envoi de la requête GET suite à une action sur le bouton JOUER

Si nous cliquons maintenant sur le petit + qui précède le nom du fichier `index.php` (voir repère 1 de la figure 8-17), nous pouvons voir cette fois qu'un onglet supplémentaire a pris place dans la partie qui s'affiche en dessous. Il s'agit de l'onglet Params (voir repère 2 de la figure 8-17) qui nous permet de voir les paramètres envoyés dans la requête. Dans notre exemple nous aurons le paramètre `button` dont sa valeur sera égale à `JOUER`. Cliquons maintenant sur le troisième onglet Response (voir repère 3 de la figure 8-17). L'information affichée par cet onglet est particulièrement intéressante car on constate ici que, contrairement aux autres ateliers Ajax, le corps de la réponse contient une page Web complète (voir repère 4 de la figure 8-17) et non pas uniquement la (ou les) donnée(s) nécessaire(s) à la modification.

Figure 8-17

Test de la page index.php : affichage du corps de la réponse contenu dans une page Web complète



Pensez à activer Firebug pour vos tests

Désormais nous ne reviendrons pas sur les fonctionnalités de Firebug déjà présentées dans les ateliers précédents pour observer le fonctionnement du système, mais nous vous invitons évidemment à utiliser ces fonctionnalités très intéressantes pour mettre au point vos programmes et à activer Firebug à chacun de vos tests.

Atelier 8-4 : requête synchrone sur un fichier PHP avec une feuille de styles

Composition du système

Nous allons maintenant nous intéresser à la réalisation d'un système équivalent à celui de l'atelier 8-3 mais réalisé cette fois avec un moteur Ajax synchrone.

Cette structure est composée :

- d'une page HTML (`index.html`), identique à celle de l'atelier 8-2, mais dans laquelle nous avons modifié le paramètre de la méthode `open()` de l'objet XHR qui cible, cette fois, un fichier PHP et non plus un simple fichier texte ;
- d'un nouveau fichier serveur PHP (`serveur.php`), dans lequel nous avons intégré un script de traitement de la requête ;
- d'une feuille de styles (`style.css`) identique à la précédente.

Fonctionnement du système

Le fonctionnement du système est semblable à celui de l'atelier 8-2 réalisé précédemment hormis le fait que le résultat affiché sera différent d'une requête à l'autre. En effet, cette fois la requête cible un fichier PHP (`gainAleatoire.php`) générant une valeur aléatoire comprise entre 0 et 100, et non plus un simple fichier texte (`gainMax.txt`) qui renvoyait la valeur 100 à chaque fois.

Conception du système

Ouvrez la page HTML de l'atelier 8-2 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/atelier8-4/`. Faites ensuite la même opération avec la feuille de styles qui est liée à cette page (`style.css`). Créez une nouvelle page PHP (depuis le menu Fichier>Nouveau cliquez sur le bouton Pages vierges puis sélectionnez PHP dans la liste proposée) et enregistrez-la sous le nom `gainAleatoire.php` dans le même répertoire. Passez l'éditeur du document en mode Code et supprimez tout le contenu HTML de la page puis saisissez à la place le script du code 8-11.

Code 8-11 :

```
<?php
//indique que le type de la réponse renvoyée sera du texte
header("Content-Type: text/plain");
//simulation du temps d'attente du serveur (2 secondes)
sleep(2);
//calcul du nouveau gain entre 0 et 100 euros
$resultat = rand(0,100);
//envoi de la réponse à la page HTML
echo $resultat;
?>
```

La première instruction permet de créer un en-tête qui précisera le type de contenu qui sera renvoyé au navigateur, à savoir du texte.

```
header("Content-Type: text/plain");
```

La seconde ligne a été ajoutée afin de simuler le temps de réponse du serveur en utilisant la fonction `sleep()` qui permet de mettre en « sommeil » le programme pendant le temps (en secondes) indiqué en paramètre de la fonction (soit 2 secondes dans notre cas). En effet, comme nous réalisons nos tests en local, les temps de transfert sont presque nuls, ce qui ne serait pas le cas si nous utilisions un script placé sur un serveur distant. Comme nos tests portent sur des communications synchrones et asynchrones, il sera particulièrement intéressant de voir comment se comporte le navigateur pendant le temps de traitement, mais encore faut-il qu'il existe.

```
sleep(2);
```

La fonction utilisée pour calculer le montant des gains d'une manière aléatoire a déjà été utilisée dans l'atelier précédent. Il s'agit de la fonction `rand()` configurée avec deux paramètres `0` et `100` de sorte à préciser la plage dans laquelle se situe la valeur aléatoire à renvoyer. Ce montant aléatoire sera affecté à une variable `$resultat` qui sera ensuite affichée à l'écran à l'aide de la fonction `echo`, dernière instruction du programme (voir figure 8-18).

```
$resultat = rand(0,100);
echo $resultat;
```

Figure 8-18

Création du fichier gainAleatoire.php

Une fois le code saisi (n'oubliez pas de fermer la balise PHP : `?>`), enregistrez votre fichier et revenez à la page HTML (`index.html`). Le contenu de ce fichier sera conservé en partie, la seule modification à effectuer concerne la méthode `open()` de l'objet XHR dans laquelle il faudra remplacer le second paramètre `gainMax.txt` par le nom du fichier contenant le script de génération du montant aléatoire que nous venons de réaliser : `gainAleatoire.php`.

```
objetXHR.open("get","gainAleatoire.php", false);
```

Une fois cette modification effectuée, enregistrez votre fichier. Le système est maintenant prêt à être testé.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 de Dreamweaver. Activez Firebug et sélectionnez la fonction Net. Dans la fenêtre Net de Firebug, nous pouvons observer que le chargement des trois objets (`index.html`, `style.css` et `logo.jpg`) constituant la page Web s'est déroulé correctement. Cliquez sur le bouton Clear de Firebug afin d'effacer ces informations.

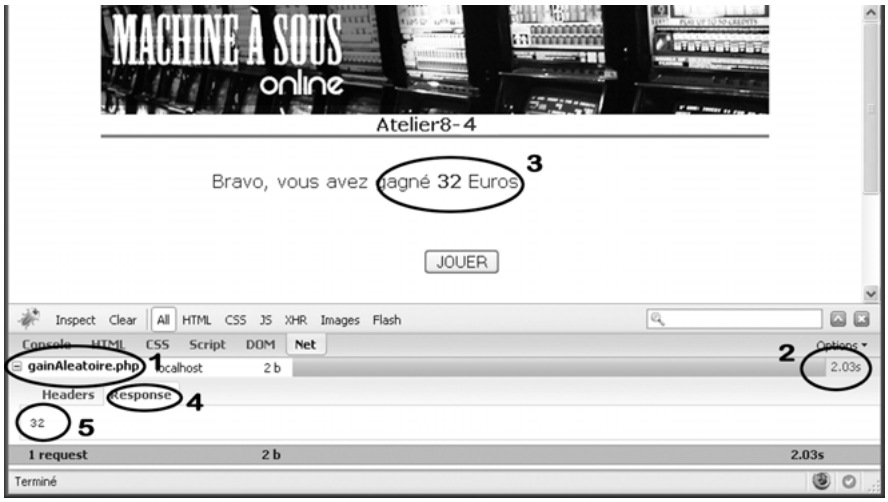
Nous allons maintenant cliquer sur le bouton JOUER pour émettre une requête synchrone vers le fichier serveur. La réponse du fichier serveur apparaît alors dans la liste de Firebug (voir repère 1 de la figure 8-19). Nous pouvons constater que la fonction `sleep()` de simulation du temps de transfert a bien fonctionné car le temps de chargement indiqué à droite de cette première ligne est légèrement supérieur à 2 secondes (voir repère 2 de la figure 8-19). De même, la valeur du gain affichée dans la zone de résultat répond à nos attentes puisqu'elle est comprise entre 0 et 100 (voir repère 3 de la figure 8-19).

Il est intéressant de remarquer que, pendant le temps d'attente de la réponse du serveur, l'utilisateur ne peut effectuer aucune action car le navigateur est bloqué (appuyez de nouveau sur le bouton JOUER et essayez de sélectionner le texte du message central pour vous en convaincre). Cela illustre très bien l'un des inconvénients des communications synchrones que nous allons tenter de solutionner avec la communication asynchrone de l'atelier suivant.

Nous pouvons aussi observer les en-têtes et le corps correspondants de la réponse. Pour cela cliquez sur le petit + qui précède le nom du fichier serveur, les en-têtes de la réponse apparaissent alors en dessous de cette ligne. Pour afficher le contenu du corps de la réponse, cliquez maintenant sur l'onglet Response (voir repère 4 de la figure 8-19). Vous pouvez alors vérifier que la valeur de la réponse correspond bien à celle indiquée dans la zone de résultat de la page Web (voir repère 5 de la figure 8-19).

Figure 8-19

Test de la page index.html de l'atelier 8-4



9

Applications Ajax-PHP sans paramètre

Nous allons maintenant nous intéresser aux applications Ajax-PHP asynchrones mais sans passage de paramètre (du client vers le serveur) dans un premier temps.

L'application utilisée pour ces ateliers sera une évolution de la machine à sous en ligne que nous avons déjà utilisée dans le chapitre précédent. Ces différents ateliers nous permettront de découvrir les problèmes des communications asynchrones Ajax et comment les résoudre.

Atelier 9-1 : requête asynchrone sur un fichier PHP avec une feuille de styles

Composition du système

Au cours de l'atelier précédent, nous avons remarqué la gêne occasionnée par le blocage du navigateur pendant le traitement d'une requête synchrone. Nous allons maintenant résoudre ce problème en vous présentant une application Ajax configurée en mode asynchrone.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure sera semblable à l'atelier précédent mais dont le script du moteur Ajax sera adapté au fonctionnement asynchrone d'une requête ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier précédent ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent.

Fonctionnement du système

Une fois la page `index.html` chargée dans le navigateur, le bouton JOUER est affiché et le message du résultat est rendu invisible par la configuration initiale de la feuille de styles. Lorsque l'utilisateur clique sur le bouton JOUER, la fonction `jouer()` est appelée. Un objet XHR est alors créé puis configuré en mode Asynchrone pour cibler le fichier serveur `gainAleatoire.php`. La requête est ensuite envoyée au serveur qui renverra une valeur aléatoire comprise entre 0 et 100 en retour. Pendant le temps de traitement, l'utilisateur peut continuer à utiliser le navigateur, contrairement à la requête synchrone que nous avons étudié dans l'atelier 8-4. À la réception de la réponse, le navigateur appelle la fonction de rappel `actualiserPage()`. Celle-ci récupérera la valeur de la réponse et l'affectera à la zone résultat comme dans le cas de l'atelier 8-4. Enfin, la dernière instruction de la fonction de rappel permettra de rendre la zone de résultat de nouveau visible et d'afficher ainsi le montant du gain à l'écran.

Conception du système

Ouvrez la page HTML de l'atelier 8-4 précédent (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap09/atelier9-1/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier. À noter que vous pouvez facilement faire des copier-coller de fichiers directement dans la fenêtre Fichiers de Dreamweaver. Il suffit pour cela de sélectionner le fichier désiré dans la liste de l'arborescence du site et de faire un clic droit puis de sélectionner l'option Édition puis Copier (ou plus rapidement, d'utiliser le raccourci clavier Ctrl + C). Placez-vous ensuite dans le répertoire de destination, cliquez droit et sélectionnez cette fois Édition puis Coller (ou plus rapidement, utilisez le raccourci clavier Ctrl + V).

Revenez à la page HTML et passez en mode Code. Localisez la ligne de la méthode `open()` de l'objet XHR et modifiez le troisième paramètre de la fonction pour le remplacer par la valeur `true`. Avec cette nouvelle configuration, la requête sera maintenant effectuée en mode asynchrone.

```
objetXHR.open("get","gainAleatoire.php", true);
```

Contrairement à une requête synchrone pour laquelle les instructions de traitement de la réponse peuvent être mis à la suite de l'envoi de la requête, en mode asynchrone, il faut définir une fonction JavaScript qui prendra en charge ce traitement d'une manière différée lorsque la réponse sera renvoyée par le serveur. Le nom de la fonction de rappel doit être mémorisé dans la propriété `onreadystatechange` de l'objet XHR avant l'envoi de la requête. Nous allons donc ajouter la ligne d'instruction ci-dessous pour configurer cette propriété et indiquer que la fonction de rappel sera la fonction `actualiserPage()` (attention à ne pas mettre de parenthèses après le nom de la fonction dans la ligne de code ci-dessous car nous désirons enregistrer la fonction et non le résultat qui pourrait être retourné par cette fonction).

```
objetXHR.onreadystatechange = actualiserPage;
```

Les instructions de traitement de la réponse placées après l'envoi de la requête doivent maintenant être transférées de la fonction `jouer()` dans la nouvelle fonction de rappel nommée `actualiserPage()` que nous allons définir ensuite. La nouvelle fonction `jouer()`, après modification, correspond désormais au code 9-1 ci-après.

Code 9-1 :

```
function jouer() {
  // Création d'une requête uniquement pour Firefox
  objetXHR = new XMLHttpRequest();
  // Configuration de la requête GET et Asynchrone
  objetXHR.open("get","gainAleatoire.php", true);
  // Désignation de la fonction de rappel
  objetXHR.onreadystatechange = actualiserPage;
  // Envoi de la requête
  objetXHR.send(null);
}
```

Nous allons maintenant définir le traitement effectué par la fonction de rappel `actualiserPage()`. Cette fonction étant exécutée à chaque changement de l'objet XHR, nous allons pouvoir l'utiliser pour effectuer le traitement des résultats du serveur. Nous allons donc insérer dans cette fonction les instructions de traitement de la réponse que nous avons précédemment supprimées de la fonction `jouer()`. Le code de la fonction de rappel `actualiserPage()` doit donc correspondre au code 9-2 ci-dessous :

Code 9-2 :

```
function actualiserPage() {
  // Récupération du résulat renvoyé par le serveur
  var nouveauGain = objetXHR.responseText;
  // Affecte le nouveau gain à la zone résultat
  document.getElementById("resultat").innerHTML=nouveauGain;
  // Affiche le nouveau message à l'écran
  document.getElementById("info").style.visibility="visible";
}
```

Les modifications de la page `index.html` sont maintenant terminées, il ne vous reste plus qu'à enregistrer la page et la tester dans le navigateur.

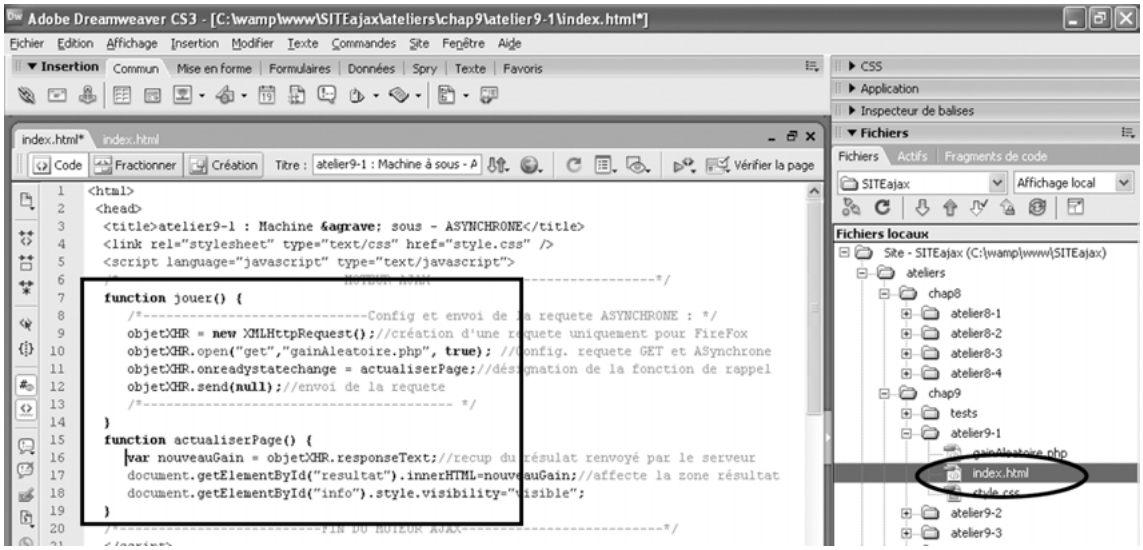


Figure 9-1

Configuration d'un moteur Ajax asynchrone

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 de Dreamweaver. Activez Firebug et sélectionnez l'onglet Net. Dans la fenêtre Net de Firebug, nous pouvons observer que le chargement des trois objets (`index.html`, `style.css` et `logo.jpg`) constituant la page Web s'est déroulé correctement. Cliquez sur le bouton Clear de Firebug afin d'effacer ces informations.

Nous allons maintenant cliquer sur le bouton JOUER pour émettre notre première requête asynchrone vers le fichier serveur. Dès que la requête est déclenchée, le nom du fichier serveur apparaît dans la liste de l'onglet Net de Firebug (voir repère 1 de la figure 9-2) et une petite animation est lancée pour indiquer que le traitement est en cours (voir repère 2 de la figure 9-2). Pendant l'attente de la réponse, nous pouvons constater cette fois que le navigateur n'est pas bloqué. Pour vous en convaincre, vérifiez qu'il est toujours possible de sélectionner le texte des commentaires situés en bas de l'écran du navigateur avec votre souris (voir repère 3 de la figure 9-2). En revanche, nous pouvons remarquer que le message du résultat s'affiche dès l'envoi de la requête alors que la valeur du gain n'est pas encore connue (voir repère 4 de la figure 9-2). Il faut ensuite attendre la fin du traitement pour que le gain s'affiche dans ce même message.

Figure 9-2

Test d'une communication asynchrone

Ce problème est lié au fait que la fonction de rappel `actualiserPage()` est appelée à chaque changement de l'objet XHR. Or celui-ci change quatre fois dans un cycle de communication HTTP (revoir si besoin le chapitre 4 sur l'objet XHR) dont la première fois juste après la configuration de l'objet avec la méthode `open()` ce qui explique que le message s'affiche dès que nous appuyons sur le bouton JOUER. Heureusement, l'objet XHR met à notre disposition la propriété `readyState` qui permet de connaître l'étape du cycle de communication HTTP.

Pour bien comprendre son fonctionnement, nous allons utiliser la console de Firebug pour suivre l'état de cette propriété au fil du processus de communication. Rappelons que la console permet d'éviter d'utiliser les traditionnels `alert()` pour déboguer un programme. Contrairement aux fenêtres `alert()` qui s'affichent à l'écran et bloquent le fonctionnement, l'utilisation de la console permet de capturer des informations et de les

visualiser dans une fenêtre de Firebug sans perturber le déroulement de l'application. Pour cela, nous allons ajouter une méthode `info()` de l'objet `console` afin d'afficher les différents états de `readyState` et la valeur de la réponse (`responseTxt`) à ce même moment dans la console de Firebug. Ouvrez le fichier `index.html` en mode Code et ajoutez dans la fonction de rappel la ligne de code en gras dans le code 9-3.

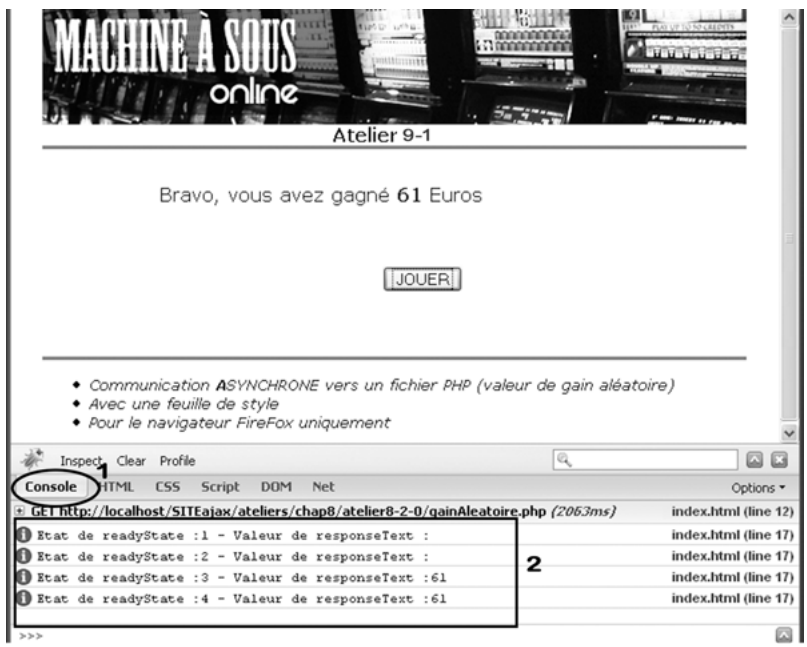
Code 9-3 :

```
function actualiserPage() {
//--------AFFICHAGE INFO DANS LA CONSOLE------------
console.info('Etat de readyState :'+objetXHR.readyState+' - Valeur de responseText
➥:'+objetXHR.responseText);
  //-------------------------------------------------
  var nouveauGain = objetXHR.responseText;
  document.getElementById("resultat").innerHTML=nouveauGain;
  document.getElementById("info").style.visibility="visible";
}
```

Testons maintenant notre système. Pour cela, appuyez sur la touche F12 pour lancer la page `index.html` dans le navigateur. Activez Firebug et cliquez sur l'onglet Console (voir repère 1 de la figure 9-3). Appuyez ensuite sur le bouton JOUER pour lancer la requête et observez la fenêtre de la console. À chaque changement d'état de la propriété `readyState`, la fonction de rappel sera appelée et affichera une ligne d'information en rapport. À la fin du cycle, nous remarquons que la propriété `readyState` a changé quatre fois d'état et que la valeur du résultat n'est disponible qu'à partir de l'état 3 de `readyState`. En pratique, nous utiliserons toujours l'état 4 de `readyState` pour déclencher le traitement de la réponse car l'état 3 correspond à une réception en cours (revoir tableau 3-6 pour mémoire) et ne nous assure pas que toutes les informations de la réponse soient bien réceptionnées selon la taille et le format de la réponse.

Figure 9-3

Utilisation de la console pour afficher les états de la propriété ready-State lors d'une communication asynchrone avec le serveur



Nos tests sont terminés, nous verrons comment exploiter cette propriété `readyState` pour éviter ce problème dans le prochain atelier.

Atelier 9-2 : requête asynchrone avec contrôle de la propriété readyState

Composition du système

Dans l'atelier précédent, nous avons remarqué qu'avec une communication asynchrone Ajax, la fonction de rappel était appelée à chaque changement d'état de l'objet XHR. Nous avons aussi analysé l'évolution de la propriété `readyState` de l'objet qui permet de connaître l'état de la communication. Nous allons maintenant solutionner ce problème en modifiant le code de la fonction de rappel en y intégrant un test qui permettra de s'assurer que la valeur de `readyState` est bien 4 avant d'exécuter les instructions de traitement du résultat.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure sera identique à celle de l'atelier précédent mais dont le code de la fonction de rappel sera modifié ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier précédent ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent.

Fonctionnement du système

Le fonctionnement sera semblable à celui de l'atelier précédent hormis le fait que le message indiquant le nouveau gain ne s'affichera que lorsque le résultat sera disponible dans le navigateur.

Conception du système

Ouvrez la page HTML de l'atelier 9-1 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/atelier9-2/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Revenez à la page HTML et passez en mode Code. Localisez la ligne de la fonction de rappel `actualiserPage()` et insérez une structure de test `if()` pour conditionner les instructions contenues dans la fonction selon l'état de la propriété `readyState` de l'objet XHR. La nouvelle fonction `actualiserPage()`, après modification, correspond désormais au code 9-4 ci-dessous. À noter que nous avons conservé l'instruction d'appel de la console dans le corps de la fonction (voir zone "AFFICHAGE INFO DANS LA CONSOLE" dans le code 9-4). Vous pourrez évidemment supprimer ou commenter cette partie de code après avoir constaté que le bloc ainsi conditionné ne sera désormais appelé que pour l'état 4 de la propriété `readyState`.

Code 9-4 :

```
function actualiserPage() {
 if (objetXHR.readyState == 4) {
  //--------AFFICHAGE INFO DANS LA CONSOLE------------
  console.info('Etat de readyState :'+objetXHR.readyState+' - Valeur de responseText
  ➥:'+objetXHR.responseText);
  //--------------------------------------------------
  var nouveauGain = objetXHR.responseText;
  document.getElementById("resultat").innerHTML=nouveauGain;
  document.getElementById("info").style.visibility="visible";
 }
}
```

Les modifications de la page `index.html` sont maintenant terminées, il ne vous reste plus qu'à enregistrer la page et à la tester dans le navigateur.

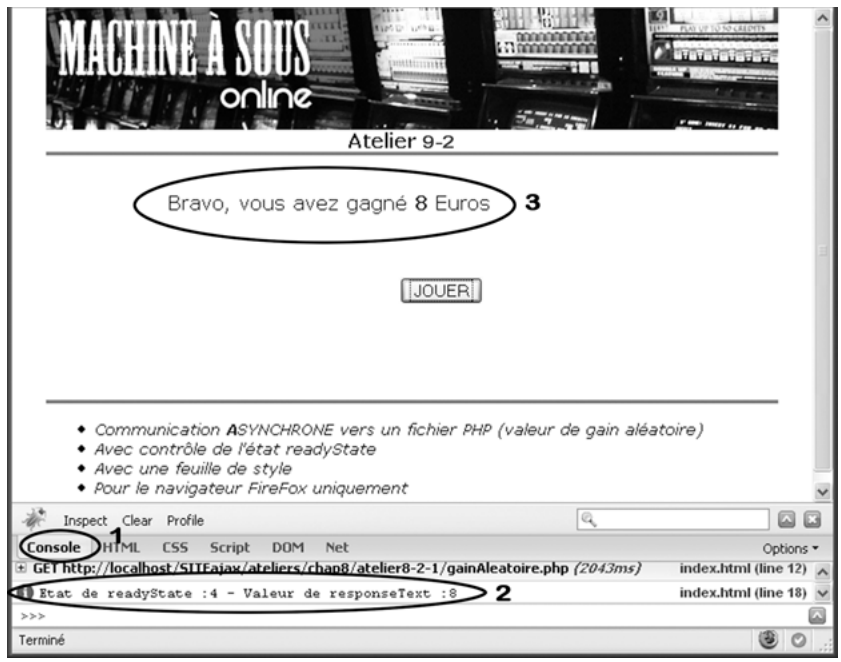
Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Activez Firebug et sélectionnez l'onglet Console (voir repère 1 de la figure 9-4).

Cliquez sur le bouton JOUER pour émettre la requête asynchrone vers le fichier serveur. Une fois que la réponse du serveur est réceptionnée par le navigateur, une information est affichée dans la console précisant que l'état de `readyState` est égal à 4 et que la valeur `responseTxt` est maintenant connue (voir repère 2 de la figure 9-4). D'autre part, nous pouvons constater que, désormais, le message du résultat et la nouvelle valeur de gain ne s'affiche dans la zone de résultat qu'au terme de la communication asynchrone (voir repère 3 de la figure 9-4).

Figure 9-4

Test d'une communication asynchrone avec contrôle de readyState.

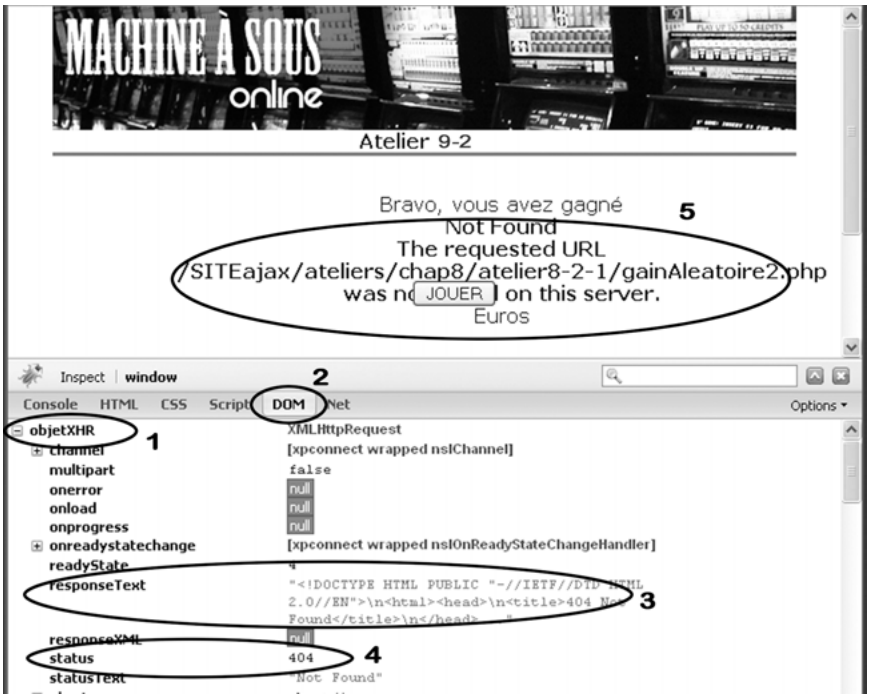


Le système semble fonctionner correctement mais voyons maintenant, ce qui se passera si un problème réseau survient lors de la communication avec le serveur. Pour cela, nous allons modifier le nom du fichier PHP ciblé par la requête (par exemple, remplacez le nom du fichier PHP actuel par `gainAleatoire2.php` dans la méthode `open()` du moteur Ajax). Une fois la modification effectuée, essayez de nouveau votre système en appuyant sur F12 puis en cliquant sur le bouton JOUER. Vous constatez que le système ne fonctionne évidemment plus et que le message affiché à la place de la valeur est plutôt alarmant pour l'utilisateur (voir repère 5 de la figure 9-5). Cependant, dans d'autres circonstances, qui peuvent varier selon la configuration de votre serveur, la situation pourrait être encore pire si vous n'aviez aucun message car l'utilisateur attendrait désespérément la réponse du serveur.

Pour analyser le problème, cliquez sur l'onglet DOM (voir repère 2 de la figure 9-5) de Firebug et développez l'objet XHR de l'arbre DOM (voir repère 1 de la figure 9-5). Cette vue nous permet de connaître toutes les valeurs des différentes propriétés de l'objet XHR. En l'occurrence, nous pouvons observer que la valeur de la propriété `status` de l'objet XHR correspondant à la valeur du statut HTTP retournée lors de la réponse (revoir si besoin le tableau 3-1 pour connaître les différentes valeurs possibles de ce paramètre) n'est pas égale à 200 comme d'habitude lorsque la communication se déroule bien, mais à 404 (cette valeur correspond à une erreur serveur lorsque le fichier ciblé est introuvable (voir repère 4 de la figure 9-5)). De même, nous pouvons constater que la réponse retournée par le serveur n'est pas une valeur comprise entre 0 et 100 mais le code HTML complet d'une page web signalant une erreur 404 (voir repère 3 de la figure 9-5).

Figure 9-5

Test d'une communication asynchrone avec une erreur du serveur non contrôlée



Nos tests sont à présent terminés. Nous avons constaté qu'en cas d'erreur du serveur, l'application se bloque et affiche un message alarmant (ou n'affiche rien du tout selon le contexte) dans l'interface utilisateur et renvoie un statut HTTP différent de 200 dans la propriété du même nom de l'objet XHR (`status`). Pour éviter ce problème, nous verrons comment exploiter la propriété `status` dans le prochain atelier.

Atelier 9-3 : requête asynchrone avec contrôle de la propriété status

Composition du système

Dans l'atelier précédent, nous avons remarqué que les erreurs serveur n'étaient pas contrôlées par le moteur Ajax. Nous allons voir maintenant comment remédier à ce problème.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure sera identique mais dont le code de la fonction de rappel sera modifié ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier précédent ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent.

Fonctionnement du système

Pour le fonctionnement du système de cet atelier, nous conserverons notre situation d'erreur serveur afin de voir comment la contrôler (le fichier ciblé sera donc encore modifié afin qu'il soit introuvable). Une fois la modification effectuée, si nous testons l'application en appuyant sur le bouton JOUER, il n'y aura plus de message d'erreur mais le message habituel d'affichage du montant du gain sera remplacé par un avertissement signalant une erreur serveur en précisant le statut correspondant au problème.

Conception du système

Ouvrez la page HTML de l'atelier 9-2 précédent (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/atelier9-3/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Revenez à la page HTML et passez en mode Code. Localisez la fonction de rappel `actualiserPage()` et insérez une seconde structure de test `if()` en dessous de la précédente pour conditionner les instructions contenues dans la fonction selon l'état de la propriété `status` de l'objet XHR.

```
if (objetXHR.status == 200) {
```

Ce premier test permettra dorénavant d'éviter que le traitement normal de la réponse soit exécuté en cas d'erreur serveur. Cependant, si nous désirons afficher un message dans l'interface pour avertir l'utilisateur du problème, il faut alors ajouter une structure `else` à la fin du bloc conditionné de sorte à programmer les instructions à exécuter en cas d'erreur. Dans notre cas, nous allons remplacer le message contenu dans la balise `<div>` dont l'identifiant est `info` par un message informant de l'erreur serveur. Pour réaliser cela nous utiliserons l'attribut `innerHTML` associé à la méthode `getElementById()` qui permettra de récupérer l'élément `info` concerné. Le message d'erreur sera, quant à lui, composé d'un texte (`Erreur serveur :`) auquel on ajoutera le code du statut et sa signification à l'aide des propriétés de l'objet XHR disponibles (soient `status` et `statusText` qui seront égaux respectivement à `404` et à `Not Found` dans notre exemple). À la suite de ces instructions, nous ajouterons aussi la méthode `abort()` de l'objet XHR afin de le supprimer en cas d'erreur serveur.

La nouvelle fonction `actualiserPage()` après modification correspond désormais au code 9-5 ci-dessous.

Code 9-5 :

```
function actualiserPage() {
 if (objetXHR.readyState == 4) {
  if (objetXHR.status == 200) {
    var nouveauGain = objetXHR.responseText;
    document.getElementById("resultat").innerHTML=nouveauGain;
```

```
    document.getElementById("info").style.visibility="visible";
  }else{
    // Message d'erreur
    document.getElementById("info").innerHTML="Erreur serveur :
    ➥"+objetXHR.status+" - "+ objetXHR.statusText;
    document.getElementById("info").style.visibility="visible";
    // Annule la requête en cours
    objetXHR.abort();
    objetXHR=null;
  }
}
```

Les modifications de la page `index.html` sont maintenant terminées, il ne vous reste plus qu'à enregistrer la page et à la tester dans le navigateur.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Cliquez sur le bouton JOUER pour déclencher l'envoi de la requête asynchrone vers le fichier erroné. Dès que le serveur renvoie son statut erreur (404 dans notre cas), un message informant de l'erreur serveur et de sa nature est affiché à la place du message de résultat habituel (voir repère 1 de la figure 9-6).

Figure 9-6

Test d'une erreur serveur avec contrôle de status

Retournez dans Dreamweaver pour corriger le problème en remplaçant le nom du fichier serveur erroné par le nom initial (`gainAleatoire.php`) puis testez de nouveau la page dans le navigateur pour vous assurer que tout est rentré dans l'ordre.

Le système semble de nouveau fonctionner correctement mais il subsiste encore un problème quant à l'utilisation de la machine à sous. En effet, entre le moment où l'utilisateur clique sur le bouton JOUER et l'affichage du nouveau gain, l'utilisateur n'est pas du tout informé qu'un traitement est en cours. Depuis nos multiples essais, nous sommes désormais habitués au fonctionnement de l'application, mais imaginez qu'un nouvel internaute utilise la machine pour la première fois, cela peut être très déstabilisant pour lui !

Par ailleurs, l'utilisateur, inquiet de ne rien voir apparaître pendant ce temps d'attente, peut cliquer de multiples fois sur le bouton JOUER (car avec une communication asynchrone, les fonctionnalités du navigateur ne sont pas bloquées pendant le traitement de la requête). Bien évidemment, ce comportement n'accélérera pas le traitement de sa demande. Il va d'une part surcharger le serveur et le réseau inutilement mais d'autre part, il risque surtout de saturer le navigateur, voire de le bloquer dans le cas de requêtes volumineuses à traiter. Par conséquent, en ce qui concerne des requêtes faisant appel à un même script serveur, il est préférable de ne soumettre qu'une requête à la fois (même si techniquement le navigateur peut gérer plusieurs requêtes en parallèle). Aussi, pour éviter ces problèmes, nous verrons dans l'atelier suivant comment bloquer l'utilisation du bouton JOUER et mettre en place un indicateur de traitement pour informer l'utilisateur.

Atelier 9-4 : requête asynchrone avec indicateur de traitement et contrôle du bouton

Composition du système

Dans l'atelier précédent, nous avons remarqué qu'aucune information ne permettait à l'utilisateur de la machine à sous de savoir si le traitement était en cours ou non et que le bouton JOUER était toujours actif pendant ce temps. Nous allons remédier à ces problèmes dans cet atelier en contrôlant le bouton JOUER et en intégrant un indicateur de chargement graphique.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure sera identique à celle de l'atelier précédent mais dont le code du moteur Ajax sera modifié ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier précédent ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent ;
- d'une nouvelle animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement de ce système est semblable à celui de l'atelier précédent (une fois l'erreur sur le nom du fichier serveur corrigée évidemment) hormis le fait que le bouton JOUER sera bloqué pendant la période de la communication HTTP et qu'une animation apparaîtra pendant cette même période pour signaler que le traitement est en cours.

Conception du système

Ouvrez la page HTML de l'atelier 9-3 précédent (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/atelier9-4/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Revenez à la page HTML et passez en mode Code. Nous allons commencer par ajouter l'animation qui signalera le traitement en cours dans la page. Pour cela, copiez dans le même répertoire un fichier GIF de votre choix et nommez-le `chargeur.gif`. Vous pouvez aussi télécharger celui qui a été utilisé pour notre exemple sur l'extension de ce livre à l'adresse

www.editions-eyrolles.com ou encore obtenir d'autres animations de chargement sur Internet (voir l'encadré ci-après). Ajoutez ensuite dans la page HTML le code permettant d'afficher cette animation et assurez-vous par la même occasion que l'identifiant du bouton est bien renseigné (voir code 9-6).

Code 9-6 :

```
<div id="page">
 <!--Zone du résultat-->
 <div id="info">Bravo, vous avez gagné <span id="resultat">0</span> Euros</div>
 <!--Zone du chargeur-->
 <img id="charge" src="chargeur.gif" />
 <!--Zone du formulaire-->
 <div id="formulaire">
 <form method="GET">
   <input name="button" id="button" type="button" onClick="jouer();" value="JOUER" />
 </form>
 </div>
</div>
```

Obtenir d'autres animations pour signaler le traitement

Si vous désirez obtenir d'autres animations GIF pour les insérer dans vos futures applications Ajax, il suffit de vous rendre à l'adresse suivante pour télécharger celle qui correspondra le mieux à vos besoins : http://ajaxload.info

Ouvrez la feuille de styles (style.css) pour définir le style qui permettra de positionner l'animation dans la page et de le rendre invisible par défaut. Ajoutez pour cela une nouvelle règle de style charge à la suite des autres règles déjà présentes dans ce fichier (voir code 9-7).

Code 9-7 :

```
#charge {
  position: absolute;
  left: 310px;
  top: 50px;
  visibility:hidden
}
```

Revenez dans la page HTML, localisez ensuite la fonction de rappel jouer() et insérez les deux lignes de code qui permettront de neutraliser le bouton et d'afficher l'animation juste avant l'appel de la méthode send() de l'objet XHR (voir code 9-8). Dans ces deux instructions, nous utiliserons la méthode getElementById() qui permet de référencer directement l'élément passé en paramètre. Nous associerons ensuite aux éléments ainsi référencés les propriétés que nous désirons modifier (disabled pour l'élément button et style.visibility pour l'élément charge). Nous pourrons ainsi facilement affecter à ces propriétés les valeurs adaptées pour neutraliser le bouton disabled=true et afficher l'animation visibility = "visible" (voir code 9-8).

Code 9-8 :

```
function jouer() {
 /*----Configuration et envoi de la requête ASYNCHRONE----*/
 objetXHR = new XMLHttpRequest();
 objetXHR.open("get","gainAleatoire.php", true);
```

```
 objetXHR.onreadystatechange = actualiserPage;
 // Gestion du bouton et du chargeur
 document.getElementById("button").disabled= true;
 document.getElementById("charge").style.visibility="visible";
 // Envoi de la requête
 objetXHR.send(null); // Envoi de la requête
 /*-------------------------------------- */
}
```

Nous allons maintenant ajouter dans la fonction de rappel `actualiserPage()` deux instructions inverses qui permettront de rétablir l'état actif du bouton JOUER et de rendre invisible l'animation dès la fin de la communication. Ces instructions sont identiques à celles de la fonction `jouer()` hormis qu'ici nous affecterons la valeur `false` à la propriété `disabled` du bouton pour le rendre de nouveau actif et la valeur `hidden` à la propriété `visibility` du style de l'animation de chargement. À noter que nous devons aussi ajouter ces mêmes instructions dans le bloc `else` qui gère le message d'erreur serveur afin de rétablir l'état initial du bouton et supprimer le chargeur en cas d'erreur serveur (voir code 9-9).

Code 9-9 :

```
function actualiserPage() {
 if (objetXHR.readyState == 4) {
  if (objetXHR.status == 200) {
   var nouveauGain = objetXHR.responseText;
   document.getElementById("resultat").innerHTML=nouveauGain;
   document.getElementById("info").style.visibility="visible";
   // Gestion du bouton et du chargement
   document.getElementById("button").disabled= false;
   document.getElementById("charge").style.visibility="hidden";
 }else{
   // Message d'erreur serveur
   document.getElementById("info").innerHTML="Erreur serveur :
   ➥"+objetXHR.status+" - "+ objetXHR.statusText;
   document.getElementById("info").style.visibility="visible";
   // Gestion du bouton et du chargeur
   document.getElementById("button").disabled= false;
   document.getElementById("charge").style.visibility="hidden";
   // Annule la requête en cours
   objetXHR.abort();
   objetXHR=null;
  }
 }

}
```

Les modifications de la page `index.html` sont maintenant terminées, il ne vous reste plus qu'à enregistrer la page et à la tester dans le navigateur.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Cliquez sur le bouton JOUER pour déclencher l'envoi de la requête asynchrone et vérifiez que le bouton devient bien inactif (voir repère 1 de la figure 9-7) pendant le temps d'attente de la réponse et qu'une animation apparaît (voir repère 2 de la figure 9-7) pour vous indiquer que le traitement est en cours.

Le système fonctionne désormais correctement dans le navigateur Firefox mais nous ne l'avons pas encore testé dans Internet Explorer. Pour cela vous avez deux alternatives, utiliser l'extension IE Tab de Firefox (qui permet de simuler le fonctionnement de IE dans Firefox) ou utiliser directement un navigateur Internet Explorer externe. Si vous désirez utiliser la première alternative, il suffit de cliquer sur le petit bouton avec l'icône Firefox en bas à droite de votre navigateur (pour disposer de ce bouton, il faut bien évidemment avoir installé préalablement l'extension IE Tab, revoir si besoin le chapitre 5). Le bouton est alors remplacé par celui de IE afin de vous rappeler que votre navigateur simule désormais son fonctionnement. Pour tester votre programme directement dans un véritable navigateur Internet Explorer, nous vous invitons à utiliser le bouton Aperçu/Débogage de Dreamweaver puis à sélectionner l'option Aperçu dans IExplore placé en seconde place après Firefox dans la liste de choix (le bouton Aperçu/Débogage a la forme d'une planète bleue et se trouve dans la barre de menus de l'éditeur de document de Dreamweaver). Évidemment, en dernier recours, vous avez aussi la possibilité de copier l'adresse de la page à tester depuis la barre d'adresse de Firefox et à la coller dans celle du navigateur IE.

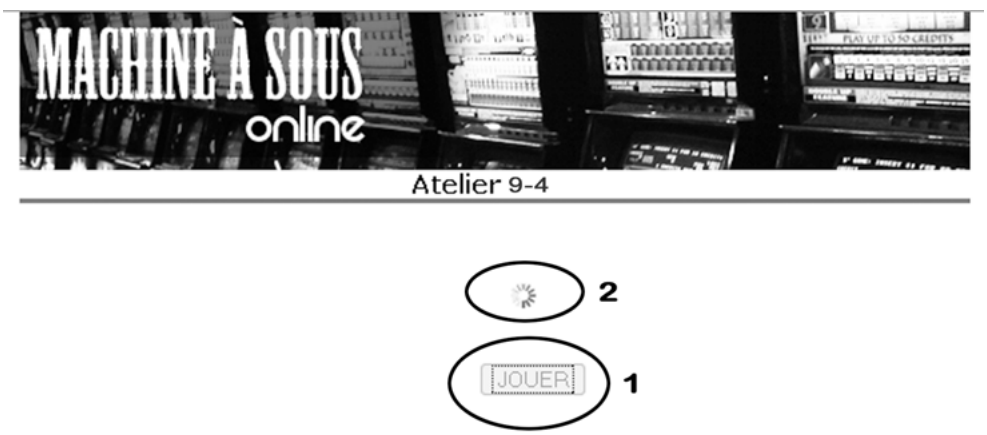


Figure 9-7

Test du système avec blocage du bouton et affichage de l'animation de traitement

Pour nos premiers tests, nous utiliserons l'aperçu dans le navigateur IE mais quelle que soit la solution choisie, vous remarquerez que lorsque vous appuierez sur le bouton JOUER, une erreur JavaScript sera générée vous indiquant que l'objet XHR est indéfini (voir figure 9-8). Pour afficher les détails de l'erreur, cliquez sur le petit triangle jaune qui doit apparaître en bas du navigateur IE (voir repère 1 de la figure 9-8) puis sur le bouton Détails de la boîte de dialogue (voir repère 2 de la figure 9-8).

Ce comportement n'est pas surprenant puisque nous avons vu dans le chapitre 4 consacré à l'objet XHR que Internet Explorer nécessite une syntaxe différente de celle de Firefox pour créer l'objet XHR (revoir le tableau 3-3 si besoin). Le prochain atelier sera donc consacré à la mise en œuvre d'une solution pour pouvoir créer un objet XHR qui fonctionne quel que soit le navigateur utilisé.

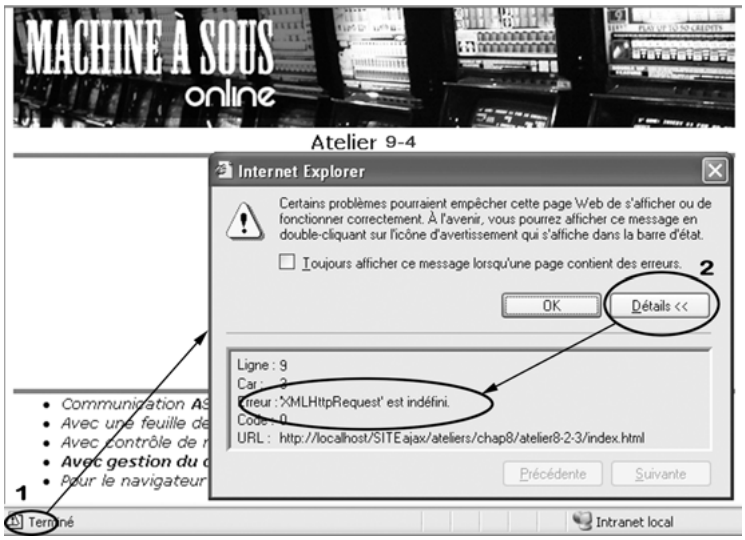


Figure 9-8

Test du système dans Internet Explorer

Atelier 9-5 : requête asynchrone avec une fonction universelle de création d'objet XHR

Composition du système

Dans l'atelier précédent nous avons remarqué que le système ne fonctionnait pas avec Internet Explorer. Nous proposons de remédier à ce problème dans le présent atelier en ajoutant une fonction universelle pour la création de l'objet XHR.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure sera identique à celle de l'atelier précédent mais dont le code du moteur Ajax sera modifié ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier précédent ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement de ce système est identique à celui de l'atelier précédent hormis le fait qu'il pourra être exécuté sur tous types de navigateurs (ou presque …).

Conception du système

Ouvrez la page HTML de l'atelier 9-4 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/atelier9-5/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Revenir à la page HTML et passez en mode Code. Nous allons commencer par ajouter une nouvelle fonction `creationXHR()` qui permettra de créer un objet XHR sur tous les

navigateurs courants (version 5.0 d'Internet Explorer comprise). Cette fonction s'appuie sur une structure d'interception d'exceptions (`try-catch`, pour plus d'informations sur le fonctionnement de cette structure, reportez-vous aux chapitres des Ressources et plus particulièrement à celui sur la technologie JavaScript à la fin de cet ouvrage). Cette structure permettra de tester successivement la création d'un objet XHR selon les différentes syntaxes compatibles avec les navigateurs usuels. Dès qu'une des syntaxes s'exécute correctement, l'objet ainsi créé est sauvegardé dans la variable `resultat` qui est elle-même retournée à la fonction appelante à l'aide du mot-clé `return`. Si aucune syntaxe ne fonctionne, la fonction retourne alors l'état `null`.

Saisissez le code de cette fonction (voir code 9-10) à la suite des deux autres fonctions JavaScript du moteur Ajax.

Code 9-10 :

```
function creationXHR() {
var resultat=null;
try { // Test pour les navigateurs : Mozilla, Opera...
    resultat= new XMLHttpRequest();
    }
catch (Error) {
     try { // Test pour les navigateurs Internet Explorer > 5.0
     resultat= new ActiveXObject("Msxml2.XMLHTTP");
     }
     catch (Error) {
        try { // Test pour le navigateur Internet Explorer 5.0
        resultat= new ActiveXObject("Microsoft.XMLHTTP");
        }
        catch (Error) {
          resultat= null;
       }
     }
   }
return resultat;
}
```

Maintenant que la fonction a été créée, il faut remplacer l'instruction de création de l'objet XHR utilisé jusqu'à présent dans la fonction `jouer()` par un appel à la nouvelle fonction de création d'objet XHR universel `creationXHR()` fonctionnant sur presque tous les navigateurs actuels. Pour cela, modifiez la première ligne du code de la fonction `jouer()` en vous référant au code 9-11.

Code 9-11 :

```
function jouer() {
  /*----Configuration et envoi de la requête ASYNCHRONE : */
  objetXHR = creationXHR();
  objetXHR.open("get","gainAleatoire.php", true);
  objetXHR.onreadystatechange = actualiserPage;
  // Gestion du bouton et du chargement
  document.getElementById("button").disabled= true;
  document.getElementById("charge").style.visibility="visible";
 // Envoi de la requête
 objetXHR.send(null);
  /*--------------------------------------- */
 }
```

Les modifications de la page `index.html` sont maintenant terminées, il ne vous reste plus qu'à enregistrer la page et à la tester dans différents navigateurs (Internet Explorer et Firefox, par exemple).

Test du système

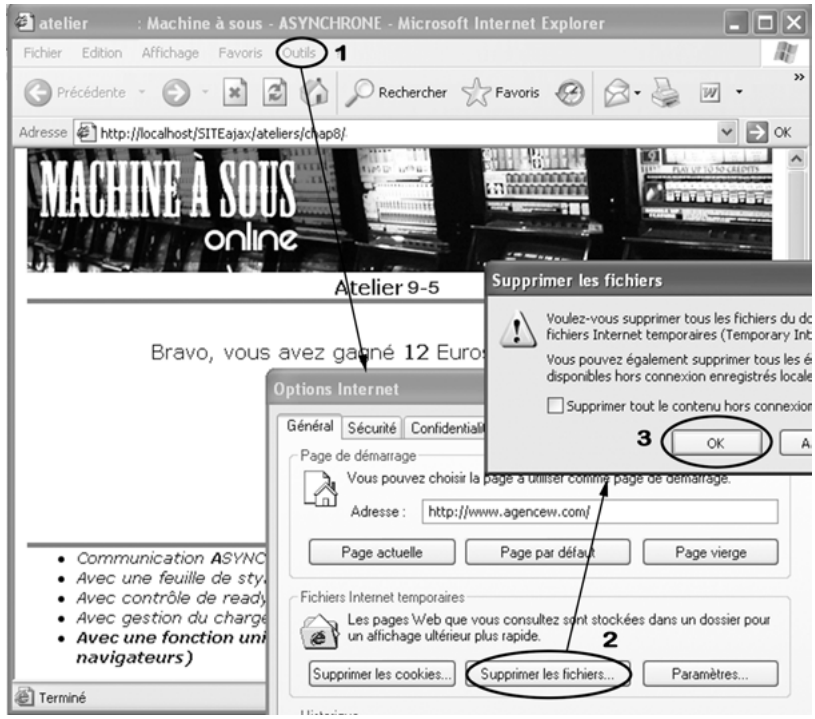
Ouvrez la page `index.html` dans le navigateur Internet Explorer en cliquant sur le bouton Aperçu/Débogage de Dreamweaver puis en sélectionnant l'option Aperçu dans IExplore dans la liste. Cliquez ensuite sur le bouton JOUER pour déclencher l'envoi de la requête asynchrone. Cette fois le système doit fonctionner dans le navigateur IE comme dans le navigateur Firefox (voir atelier précédent). Le bouton JOUER doit devenir inactif pendant la communication et l'animation doit apparaître pendant cette même période. Au terme de la communication, le message et son résultat aléatoire doivent apparaître à l'écran.

Renouvelez maintenant votre action sur le bouton JOUER, vous constatez que le bouton reste actif et que la valeur du résultat ne change plus. Que s'est-il passé ?

En réalité, le navigateur Internet Explorer a stocké la réponse correspondante à la requête `GET` du navigateur lors du premier test. Cette procédure est employée par les navigateurs pour accélérer le téléchargement et l'affichage des pages Web. Si tous les navigateurs stockent les images pour permettre cette optimisation de votre navigation, Internet Explorer, quant à lui stocke aussi les réponses des requêtes `GET` s'il estime qu'elle a déjà été effectuée auparavant.

Figure 9-9

Suppression du cache dans le navigateur Internet Explorer



Ainsi désormais, lorsque vous appuyez de nouveau sur le bouton, le navigateur identifie cette nouvelle demande comme étant identique à la précédente et renvoie le même résultat que celui de la première réponse. Pour vous en convaincre, vous allez supprimer le cache (vous pouvez aussi utiliser le raccourci clavier Ctrl+F5 pour actualiser votre page en centralisant des valeurs du cache) en sélectionnant l'entrée Outils du menu du navigateur IE (voir repère 1 de la figure 9-9), puis Options Internet. Une boîte de dialogue doit alors s'ouvrir, cliquez sur le bouton Supprimer les fichiers (voir repère 2 de la figure 9-9) puis sur le bouton `OK` de la seconde boîte de dialogue (voir repère 3 de la figure 9-9). Si vous

testez de nouveau le système en cliquant sur le bouton JOUER, le même scénario doit alors se reproduire.

Nous verrons dans l'atelier suivant comment éviter ce problème de mise en cache de la réponse d'une requête `GET` avec Internet Explorer.

Atelier 9-6 : requête asynchrone avec anti-cache

Composition du système

Dans l'atelier précédent nous avons constaté que le premier résultat de la requête `GET` est resté dans le cache du navigateur Internet Explorer. Dans ce nouvel atelier, nous allons donc étudier les solutions pour remédier à ce problème.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure sera identique à celle de l'atelier précédent mais dont le code du moteur Ajax sera modifié ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier précédent mais dont le code sera modifié ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement de ce système est identique à celui de l'atelier précédent hormis l'ajout de l'anti-cache que nous allons mettre en place. Ceci devrait maintenant permettre de réaliser de multiples jeux dans le navigateur Internet Explorer sans avoir à vider le cache manuellement comme nous l'avons fait précédemment.

Conception du système

Ouvrez la page HTML de l'atelier 9-5 précédent (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/atelier9-6/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Dans cet atelier, nous allons vous proposer deux solutions qui devraient, indépendamment l'une de l'autre, lever le problème de cache que nous avons identifié avec Internet Explorer. Vous pourriez utiliser l'une ou l'autre selon votre choix mais l'idéal est évidemment d'exploiter ces deux solutions ensemble. La première solution agit sur le moteur Ajax alors que la seconde entraîne la modification du programme serveur qui renvoie la réponse. Notez que la première solution peut être très intéressante lorsque l'on n'a pas accès au script serveur.

Commençons par la première solution. Le principe consiste à tromper le navigateur en lui faisant croire que les requêtes sont différentes les unes des autres. Ainsi, le navigateur pensant qu'il s'agit d'une nouvelle requête, ne récupérera pas la valeur précédemment mémorisée dans le cache mais celle qui sera renvoyée par le serveur. Pour mettre en œuvre cette technique, nous ajouterons une variable et sa valeur à la suite de l'URL du script serveur qui est configurée dans le second paramètre de la méthode `open()`. La variable ne sera pas réellement utilisée mais l'important est que la valeur de cette variable soit

toujours différente d'une requête à l'autre. Nous pourrions par exemple utiliser une valeur générée aléatoirement ou la valeur du temps. Dans notre exemple, nous allons utiliser le second choix pour mettre en œuvre l'anti-cache. Pour récupérer le temps et l'enregistrer dans une variable, nous allons utiliser la méthode `getTime()` de l'objet `Date`. Celle-ci sera ensuite enregistrée dans une variable nommée `temps`.

Affichez la page HTML en mode Code et ajoutez l'instruction suivante au début de la fonction `jouer()` afin de mémoriser dans la variable `temps` la valeur du temps à l'instant même où la fonction sera appelée.

```
var temps = new Date().getTime();
```

Une fois la valeur du paramètre connue, il ne reste plus qu'à l'ajouter à la suite de l'URL du fichier serveur dans le deuxième argument de la méthode `open()` de l'objet XHR (situé dans la fonction `jouer()` du moteur Ajax). Dans notre exemple, nous avons nommé le paramètre d'URL `anticache` mais vous pourriez très bien utiliser le nom de votre choix car il ne sera jamais utilisé.

```
objetXHR.open("get","gainAleatoire.php?anticache="+temps, true);
```

Enregistrez ensuite la page `index.html`. Une fois modifiée, la fonction `jouer()` sera semblable au code 9-12.

Code 9-12 :

```
function jouer() {
  /*----Configuration et envoi de la requête ASYNCHRONE----*/
  objetXHR = creationXHR();
  var temps = new Date().getTime();
  objetXHR.open("get","gainAleatoire.php?anticache="+temps, true);
  objetXHR.onreadystatechange = actualiserPage;
  // Gestion du bouton et du chargeur
  document.getElementById("button").disabled= true;
  document.getElementById("charge").style.visibility="visible";
  objetXHR.send(null);
  /*---------------------------------------- */
 }
```

La seconde solution consiste à ajouter un en-tête `Cache-Control` dans la réponse renvoyée par le serveur. Cet en-tête permet de contrôler la mise en cache de la réponse HTTP et il dispose pour cela de plusieurs valeurs possibles. Dans notre cas nous utiliserons les valeurs `no-cache` et `private` qui indiquent aux navigateurs et aux proxy de ne pas placer la réponse HTTP dans leur cache. À noter que cet en-tête est adapté aux serveurs exploitant le protocole HTTP actuel (1.1). Si vous désirez que le blocage du cache fonctionne aussi sur les serveurs utilisant le protocole HTTP de la version antérieure (1.0), il faudra dans ce cas ajouter un second en-tête nommé `Pragma` et lui affecter la valeur `no-cache`.

Pour générer ces en-têtes depuis le fichier PHP `gainAleatoire.php`, nous utiliserons des fonctions `header()` déjà utilisées dans ce même fichier pour indiquer le type des données renvoyées dans la réponse. Une fois modifiée, le fichier `gainAleatoire.php` sera semblable au code 9-13.

Code 9-13 :

```
// Indique que le type de la réponse renvoyée au client sera du texte
header("Content-Type: text/plain");
// Anticache pour HTTP/1.1
header("Cache-Control: no-cache , private");
```

```
// Anticache pour HTTP/1.0
header("Pragma: no-cache");
// Simulation du temps d'attente du serveur (2 secondes)
sleep(2);
// Calcul du nouveau gain entre 0 et 100 euros
$resultat = rand(0,100);
// Envoi de la réponse à la page HTML
echo $resultat ;
```

Test du système

Ouvrez la page `index.html` dans le navigateur Internet Explorer en cliquant sur le bouton Aperçu/Débogage de Dreamweaver puis en sélectionnant l'option Aperçu dans IExplore dans la liste. Cliquez ensuite sur le bouton JOUER pour déclencher l'envoi d'une première requête asynchrone. Dès que le résultat est affiché, renouvelez votre action sur le bouton JOUER pour vous assurer que le système de cache fonctionne correctement. Le résultat affiché par la seconde requête devra alors être différent du premier test.

Atelier 9-7 : requête asynchrone avec les fonctions DOM

Composition du système

Nous allons maintenant nous intéresser à la conformité de notre code aux spécifications W3C et plus particulièrement, à celle de l'attribut `innerHTML` que nous avons déjà utilisé à maintes reprises dans les ateliers précédents.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure sera identique à celle de l'atelier précédent mais dont le code du moteur Ajax sera modifié ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier précédent ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement de ce système est strictement identique à celui de l'atelier précédent. Le principal bénéfice des modifications que nous allons effectuer sera de disposer d'un moteur Ajax conforme à la normalisation du W3C et d'améliorer ainsi la compatibilité de notre code avec tous les navigateurs qui respectent cette norme.

Conception du système

Ouvrez la page HTML de l'atelier 9-6 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/atelier9-7/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Avant de commencer la modification du code, rappelons que l'attribut `innerHTML` n'est pas normalisé par le W3C même si on le retrouve dans les attributs d'un élément de l'arbre DOM. Cet attribut est souvent utilisé dans les codes des moteurs Ajax car il est très pratique pour mettre à jour le contenu d'un élément. En effet, il permet de remplacer très facilement le contenu d'un élément par une simple affectation de la nouvelle valeur. Dans l'exemple ci-dessous, la valeur du nouveau gain (`nouveauGain`) est ainsi affectée à l'attribut `innerHTML` de l'élément `resultat`. Ainsi, le nouveau contenu de l'élément `resultat` sera remplacé par la valeur de `nouveauGain`.

```
var nouveauGain = objetXHR.responseText;
document.getElementById("resultat").innerHTML=nouveauGain;
```

Cependant, il n'est pas conseillé d'utiliser `innerHTML` car la construction des nouveaux contenus avec cet attribut ne respecte pas la structuration des spécifications du DOM. La solution recommandée consiste à exploiter les méthodes de manipulation des nœuds de l'arbre DOM (comme `appendChild()` ou `removeChild()`, par exemple). Évidemment, cette technique est beaucoup plus complexe, mais elle a le mérite d'être conforme au W3C et de faciliter la maintenance de votre code JavaScript.

Pour vous faciliter la chose, nous vous proposons de créer deux petites fonctions qui permettront, par la suite, de vous conformer au W3C sans avoir à redévelopper des scripts complexes à chaque fois.

La première permet de supprimer tous les nœuds enfants de l'élément passé en paramètre (voir code 9-14). Elle utilise la méthode DOM `removeChild()` intégrée dans une boucle parcourant tous les nœuds enfants de l'élément ().

Code 9-14 :

```
function supprimerContenu(element) {
if (element != null) {
  while(element.firstChild)
    element.removeChild(element.firstChild);
}
}
```

Resssources sur le DOM

Pour plus de détails sur les méthodes et propriétés utilisées dans cette fonction, reportez-vous au chapitre 20 consacré à la gestion DOM à la fin de cet ouvrage

La seconde fonction, exploite la première, et permet de remplacer tout le contenu d'un élément par un texte. Pour cela, il faudra préciser dans les paramètres de la fonction, l'identifiant de l'élément `id` et le texte à utiliser `texte` (voir code 9-15). Cette seconde fonction utilise les méthodes DOM `createTextNode()` et `appendChild()` pour construire un nouveau contenu conforme aux spécifications du W3C.

Code 9-15 :

```
function remplacerContenu(id, texte) {
 var element = document.getElementById(id);
 if (element != null) {
   supprimerContenu(element);
   var nouveauContenu = document.createTextNode(texte);
   element.appendChild(nouveauContenu);
 }
}
```

Saisissez ces deux nouvelles fonctions dans notre page `index.html` à la suite de la fonction `creationXHR()` du moteur Ajax. Une fois la saisie terminée, nous pourrons alors remplacer les instructions actuelles utilisant l'attribut `innerHTML` par la fonction `remplacerContenu()`.

Commençons par appliquer cette nouvelle technique à la partie de code qui actualise la balise `<div>` (d'identifiant `resultat`) par le nouveau gain renvoyé par le serveur (cette partie est située dans la fonction de rappel `actualiserPage()`).

Récupérez tout d'abord la valeur du nouveau gain dans la variable `nouveauGain`. Ensuite, il suffit d'utiliser la fonction déclarée précédemment `remplacerContenu()` en passant en paramètre l'identifiant de l'élément à modifier (`resultat`) et la valeur du nouveau texte (mémorisée dans `nouveauGain`) pour effectuer l'actualisation de la zone de résultat (voir code 9-16). Au niveau du fonctionnement du système, cela ne changera strictement rien mais par contre vous aurez une structure d'arbre DOM conforme au W3C.

Code 9-16 :

```
var nouveauGain = objetXHR.responseText;
// Actualisation du résultat
remplacerContenu("resultat", nouveauGain);
```

Nous pouvons maintenant appliquer la même démarche à l'autre partie de code utilisant l'attribut `innerHTML` afin d'afficher l'éventuel message d'erreur serveur. Une fois, les modifications effectuées, la nouvelle fonction `actualiserPage()` ressemblera au code 9-17.

Code 9-17 :

```
function actualiserPage() {
if (objetXHR.readyState == 4) {
  if (objetXHR.status == 200) {
    var nouveauGain = objetXHR.responseText;
    //---- Actualisation du résultat
    // Actualise le contenu de l'élément résultat avec nouveauGain
    actualiserContenu("resultat", nouveauGain);
    //-----------------------------------
    // Affiche la zone info
    document.getElementById("info").style.visibility="visible";
    // Gestion du bouton et du chargeur
    document.getElementById("button").disabled= false;
    document.getElementById("charge").style.visibility="hidden";
  }else{
    // Message d'erreur serveur
    var erreurServeur="Erreur serveur : "+objetXHR.status+" -
    ➥"+ objetXHR.statusText;
    remplacerContenu("info", erreurServeur);
    document.getElementById("info").style.visibility="visible";
  }
}
}
```

Les modifications de la page `index.html` sont maintenant terminées, il ne vous reste plus qu'à enregistrer la page et à la tester dans Firefox.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Cliquez sur le bouton JOUER pour déclencher l'envoi de la requête asynchrone. Le système doit fonctionner exactement de la même manière que dans l'atelier précédent qui utilisait l'attribut `innerHTML`.

Atelier 9-8 : requête asynchrone avec fichiers JS externes

Composition du système

Au cours des ateliers précédents, nous avons ajouté de nombreuses fonctions JavaScript dans le moteur Ajax. Le moment est venu de faire le ménage et d'organiser ces différentes fonctions dans des fichiers JS externes. Avec une structure de fichiers JS externes, la maintenance de vos codes sera plus facile et vous pourrez ainsi capitaliser vos scripts dans vos futurs développements.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure sera identique à celle de l'atelier précédent mais sans les fonctions JavaScript qui seront déplacées dans des fichiers JS externes ;
- d'un nouveau fichier JS (`fonctionsAjax.js`) qui contiendra les fonctions communes à tous les moteurs Ajax ;
- d'un nouveau fichier JS (`fonctionsMachine.js`) qui contiendra les fonctions spécifiques à l'application Ajax de la machine à sous ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier précédent ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement de ce système est strictement identique à celui de l'atelier précédent.

Conception du système

Ouvrez la page HTML de l'atelier 9-7 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/atelier9-8/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Parmi les fonctions créées, certaines sont spécifiques à l'application de la machine à sous alors que d'autres sont génériques et pourront être réutilisées sans modification dans une

autre application. Nous allons donc commencer par ventiler ces fonctions dans ces deux familles afin de pouvoir les répartir ensuite dans des fichiers JS différents.

Tableau 9-1 Ventilation des fonctions JavaScript

Fonctions génériques	Fonctions spécifiques
`creationXHR()`	`jouer()`
`supprimerContenu()`	`actualiserPage()`
`remplacerContenu()`	

Une fois la ventilation effectuée, il faut transférer ces fonctions du fichier `index.html` dans les fichiers JS correspondant. Nous allons donc créer deux fichiers JS externes : le premier, pour les fonctions génériques, s'appellera `fonctionsAjax.js` et le second, pour les fonctions spécifiques à l'application de la machine à sous, s'appellera `fonctionsMachine.js`.

Commençons par créer le premier fichier JS dans Dreamweaver. Dans le menu Fichier, cliquez sur Nouveau > Pages vierges puis sélectionner l'option JavaScript dans la liste. Une fois le fichier créé, enregistrez-le tout de suite sous le nom `fonctionsAjax.js`. Passez ensuite dans la page `index.html` et coupez successivement les fonctions listées dans la colonne Fonctions génériques du tableau 9-1 puis collez-les dans le nouveau fichier JS. Une fois les trois fonctions déplacées, enregistrez votre fichier et suivez la même procédure pour le fichier des fonctions spécifiques `fonctionsMachine.js`.

Lorsque toutes les fonctions JavaScript seront ainsi ventilées dans les deux fichiers JS externes, vous pourrez supprimer les balises `<script>` qui contenaient ces fonctions de la page `index.html`. Pour lier ces nouveaux fichiers externes, en revanche, nous devons maintenant créer deux nouveaux liens `<script>` pointant sur les fichiers précédemment créés. Pour cela, nous vous suggérons d'utiliser l'assistant d'une fonctionnalité de la barre d'outils Insertion de Dreamweaver. Cliquez sur l'onglet Commun de la barre d'outils Insertion (situé en haut de l'interface de Dreamweaver, voir figure 9-10) puis sur le bouton Script et sélectionnez la première option du menu (voir repère 1 de la figure 9-10). Dans la boîte de dialogue Script, cliquez sur le petit dossier jaune (voir repère 2 de la figure 9-10) et sélectionnez ensuite le fichier JS désiré dans l'autre boîte de dialogue (voir repère 3 de la figure 9-10). Cliquez enfin sur le bouton OK des deux boîtes de dialogue pour valider votre choix. La balise `<script>` qui permettra de lier votre fichier JS externe doit alors apparaître dans votre code (voir code 9-18). Renouvelez ensuite cette démarche pour le second fichier JavaScript.

Code 9-18 :

```
<script type="text/javascript" src="fonctionsAjax.js"></script>
<script type="text/javascript" src="fonctionsMachine.js"></script>
```

Les modifications de la page `index.html` sont maintenant terminées, il ne vous reste plus qu'à enregistrer la page et à la tester dans Firefox.

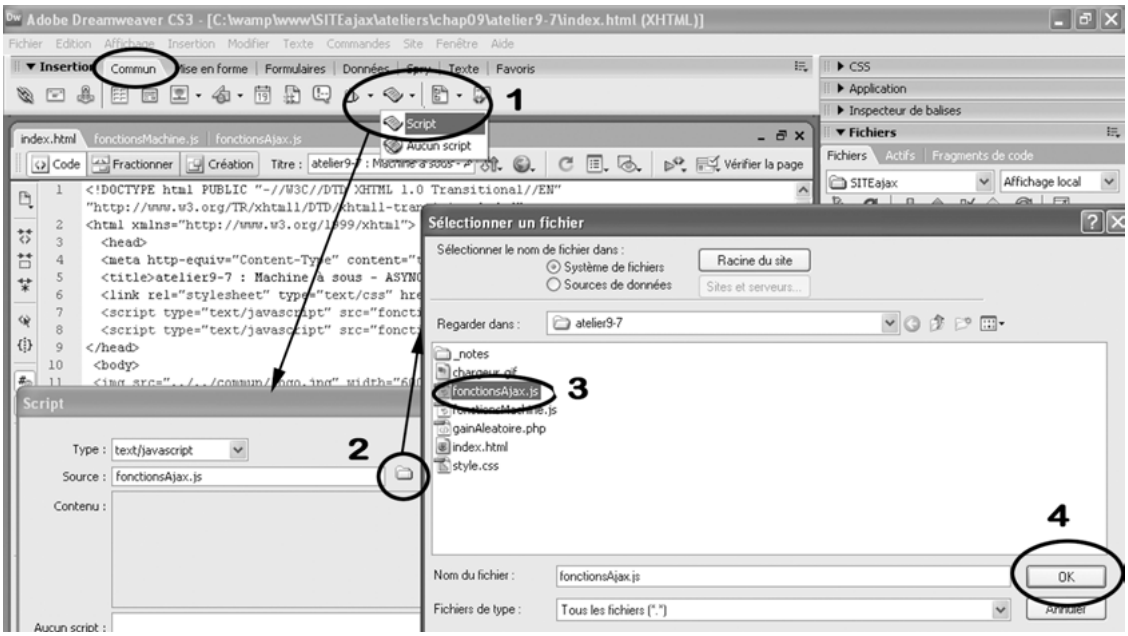


Figure 9-10

Création d'un lien sur un fichier JS externe avec Dreamweaver

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Cliquez sur le bouton JOUER pour déclencher l'envoi de la requête asynchrone. Le système doit fonctionner exactement de la même manière que dans l'atelier précédent dans lequel les fonctions étaient déclarées dans la page `index.html`.

10

Applications Ajax-PHP avec paramètres GET

Jusqu'à présent, nous nous sommes contentés d'envoyer des requêtes sans paramètre : nous n'en avions pas besoin car le serveur nous renvoyait une valeur aléatoire qui changeait à chaque appel. Or, dans la majorité des applications Ajax, il est intéressant de pouvoir communiquer un ordre au script serveur de sorte qu'il puisse traiter la requête en fonction de cette information et renvoyer une réponse en rapport. Pour cela, nous allons devoir ajouter des paramètres lors de l'envoi de la requête et les gérer côté serveur. Les ateliers de ce chapitre sont consacrés à la mise en œuvre de telles applications.

Atelier 10-1 : requête asynchrone avec un champ texte

Composition du système

Pour commencer, nous allons traiter un exemple simple en ajoutant un champ de saisie au formulaire de l'application de l'atelier précédent proposant au joueur d'indiquer son nom afin que la réponse du serveur soit personnalisée en conséquence (la valeur saisie dans ce champ sera ensuite récupérée et envoyée en paramètre au serveur). Comme nous aurons cette fois deux informations retournées par le serveur (le nom du joueur et son gain) nous en profiterons pour vous présenter une solution simple pour gérer une réponse HTTP du serveur constituée de plusieurs valeurs.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure sera identique à celle de l'atelier précédent mais avec un champ de saisie en plus dans le formulaire ;
- d'un fichier JS (`fonctionsAjax.js`) qui contiendra les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contiendra les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification sera identique à celle de l'atelier précédent ;

- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier précédent mais qui sera modifié pour gérer le paramètre de la requête ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Une fois la page `index.html` chargée dans le navigateur, l'utilisateur devra saisir son nom avant de cliquer sur le bouton JOUER pour déclencher le moteur Ajax. Celui-ci récupérera la valeur saisie dans le champ et l'enverra en paramètre au serveur. À la réception de la requête, le serveur récupérera le paramètre pour conditionner le traitement à effectuer. Dans notre exemple, le traitement sera très simple car le serveur se contentera de renvoyer le nom de l'utilisateur dans sa réponse en plus de la valeur du gain. À la réception de la réponse, la fonction de rappel du moteur Ajax analysera son contenu pour en tirer les deux informations attendues (le nom du joueur et son gain). Il mettra ensuite en forme ces informations dans un texte qui s'affichera à l'écran selon la structure ci-dessous (avec `XX1` pour le nom du joueur et `XX2` pour la valeur du gain).

```
Bravo M. XX1, vous avez gagné XX2 euros
```

Conception du système

Ouvrez la page HTML de l'atelier 9-8 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap10/atelier10-1/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Nous allons commencer par ajouter le champ de saisie dans le formulaire. Pour cela, ouvrez la page `index.html` en mode Création et agrandissez la zone `<div>` (dont l'identifiant est `formulaire`) à l'aide de ses poignées de sorte à pouvoir y insérer l'objet du champ de saisie et un texte d'information (voir repères 3 et 4 de la figure 10-1). Pour ajouter le champ dans le formulaire, vous pourrez avantageusement utiliser les éléments de formulaire de l'onglet Formulaire de la barre d'outils Insertion (voir repère 1 de la figure 10-1). Sélectionnez ensuite le champ de saisie et renseignez l'information Champ texte du panneau des Propriétés avec la valeur `nom` de sorte à identifier ce nouvel élément (cette procédure configure les attributs `name` et `id` du champ). Une fois modifiée, la partie de la zone `formulaire` doit correspondre au code 10-1.

Code 10-1 : structure de la page `index.html` :

```
<div id="formulaire">
   <form method="GET">
     Indiquez votre nom :
     <input type="text" id="nom" name="nom" />
     avant de
     <input name="button" id="button" type="button" onClick="jouer();" value="JOUER" /
  </form>
  </div>
```

Évidemment, au lieu de redimensionner la taille de la zone formulaire avec les poignées, vous pouvez aussi modifier la règle de styles `formulaire` directement dans la feuille de styles `style.css`. Dans ce cas, utilisez les informations du code 10-2.

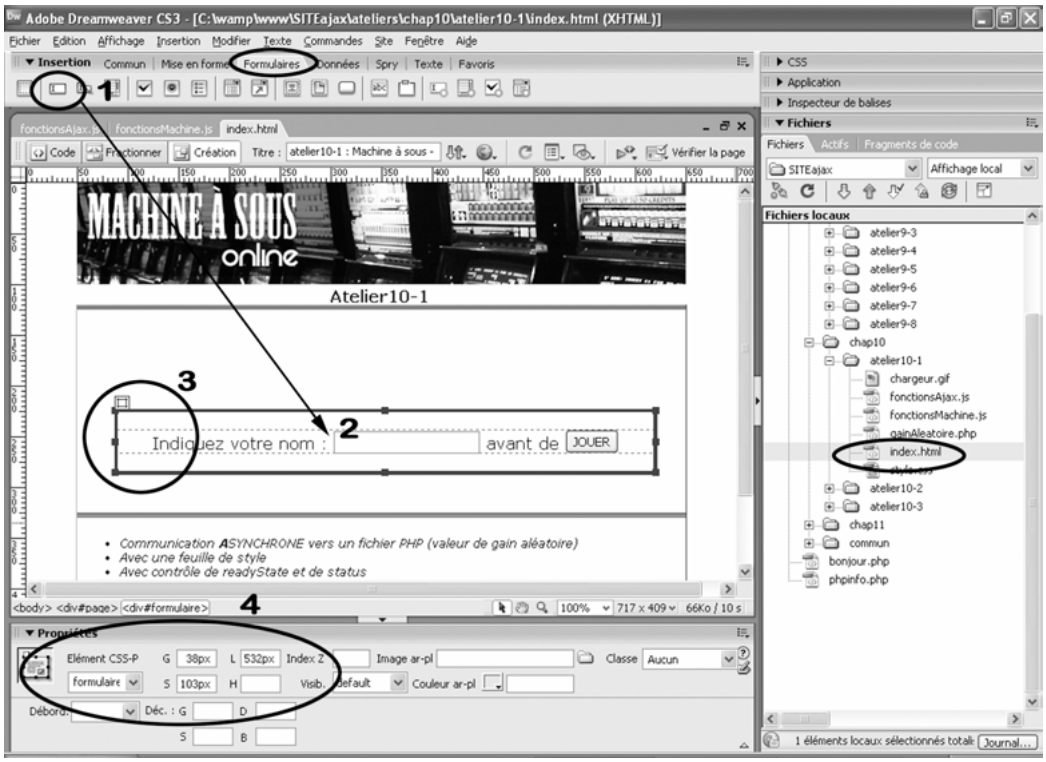


Figure 10-1

Modification de la zone du formulaire et ajout du champ de saisie

Code 10-2 : règle de style `formulaire` :

```
#formulaire {
   position: absolute;
   left: 38px;
   top: 103px;
   width: 532px;
}
```

Restez dans cette même page `index.html` mais passez en mode Code, puis localisez la ligne de la balise `<div>` dont l'identifiant est `info`. Modifiez ensuite son contenu en ajoutant une nouvelle balise `<span>` dont l'identifiant sera `gagnant` de sorte à pouvoir y insérer le nom du gagnant selon l'exemple de code suivant.

```
<div id="info">Bravo, M <span id="gagnant"></span> vous avez gagné
➥<span id="resultat"></span> euros</div>
```

Les modifications de la page `index.html` sont maintenant terminées, enregistrez votre fichier ainsi que le fichier `style.css` qui a été aussi modifié automatiquement par Dreamweaver lors du redimensionnent de la balise `<div>`.

Ouvrez maintenant le fichier `fonctionsMachine.js`. Nous devons insérer dans ce fichier, le script qui permettra de récupérer la valeur saisie par l'utilisateur dans le champ `nom` et l'ajouter en paramètre à l'URL lors de l'appel du serveur. Pour cela nous allons commencer

par récupérer la valeur saisie dans une variable locale de la fonction `jouer()` en utilisant l'attribut `value` de l'élément concerné.

```
var nom = document.getElementById("nom").value;
```

Il faut ensuite construire les paramètres à envoyer en associant la variable `anticache` avec la nouvelle variable `nom` précédemment récupérée. Pour cela, nous allons créer une nouvelle variable locale nommée `parametres` qui contiendra une chaîne de caractères réalisée par la concaténation des deux variables et de leur valeur au format d'URL.

```
var parametres = "nom="+ nom +"&anticache="+temps ;
```

Maintenant que nous disposons de tous les éléments regroupés dans une seule variable `parametres`, il suffit d'ajouter cette même variable à la suite de l'URL du serveur (précédée du caractère `?`) dans le second paramètre de la méthode `open()` :

```
objetXHR.open("get","gainAleatoire.php?"+parametres, true);
```

La fonction `jouer()` après sa modification doit être semblable au code 10-3 :

Code 10-3 : fonction `jouer()` :

```
function jouer() {
  objetXHR = creationXHR();
  var temps = new Date().getTime();
  var nom = document.getElementById("nom").value;
  var parametres = "nom="+ nom +
                   "&anticache="+temps ;
  objetXHR.open("get","gainAleatoire.php?"+parametres, true);
  objetXHR.onreadystatechange = actualiserPage;
  document.getElementById("button").disabled= true;
  document.getElementById("charge").style.visibility="visible";
  objetXHR.send(null);
}
```

Les modifications du code côté client ne sont pas terminées (le fichier `fonctionsMachine.js` doit encore être modifié), mais nous allons d'abord nous préoccuper du fichier serveur en PHP avant de terminer les modifications du moteur Ajax.

Ouvrez le fichier `gainAleatoire.php` et passez en mode Code. Le fichier serveur doit interpréter le nouveau paramètre envoyé dans la requête puis traiter et renvoyer une réponse en rapport au navigateur.

Nous allons donc commencer par récupérer la valeur passée en paramètre dans une variable locale `$nom` du programme. Pour cela nous utiliserons le code suivant.

```
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
```

Dans ce code, nous testons si la variable envoyée en paramètre existe avec la fonction `isset($_REQUEST['nom'])`. À noter que, dans notre exemple, comme la requête est envoyée en `GET`, nous aurions aussi pu utiliser la variable HTTP `$_GET['nom']` mais `$_REQUEST['nom']` a le mérite de fonctionner aussi bien avec la méthode `GET` qu'avec la méthode `POST`. Dans le cas où le paramètre n'existe pas, nous utilisons une structure `else` pour affecter la valeur `inconnu` à la variable `$nom` pour ne pas générer de message d'erreur en cas d'absence de paramètre.

Comme nous n'effectuons pas de traitement spécifique sur le nom de l'utilisateur, il ne nous reste plus maintenant qu'à préparer la réponse à envoyer au navigateur. Or, dans ce

nouveau système, nous désirons renvoyer deux informations au client : le gain mais aussi le nom du joueur. Pour cela nous allons utiliser un format spécifique dans lequel nous allons séparer les deux valeurs avec un caractère choisi. En l'occurrence nous allons utiliser le caractère « : » comme séparateur mais vous pourriez aussi utiliser le caractère de votre choix (comme « | » par exemple qui est aussi souvent employé pour cet usage). La réponse renvoyée par le serveur aura donc la structure suivante (avec `XX1` pour la valeur du nom de l'utilisateur et `XX2` pour la valeur du gain) :

```
XX1 : XX2
```

Pour construire la réponse dans ce format en PHP, nous allons utiliser des opérateurs de concaténation pour insérer le caractère « : » entre les deux valeurs. La chaîne ainsi composée sera ensuite enregistrée dans la variable `$resultat`.

```
$resultat=$nom.':'.$gain;
```

Par rapport à l'atelier précédent, vous remarquerez que le gain est maintenant enregistré dans une variable spécifique nommée `$gain`, il faudra donc changer le nom de cette variable lors de son calcul. Les modifications du code du fichier serveur `gainAleatoire.php` sont maintenant terminées, vous pouvez l'enregistrer avant de revenir à notre fichier `fonctionsMachine.js`.

```
//indique le type de la réponse renvoyée
header("Content-Type: text/plain");
//simulation du temps d'attente du serveur (2 secondes)
sleep(2);
//récupération du paramètre HTTP nom
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
//calcul du nouveau gain entre 0 et 100 euros
$gain = rand(0,100);
//mise en forme du résultat avec le nom
$resultat=$nom.':'.$gain;
//envoi de la réponse à la page HTML
echo $resultat ;
```

Maintenant que nous avons présenté le format dans lequel vont être renvoyés les résultats, nous pouvons revenir au fichier `fonctionsMachine.js` pour gérer la récupération de la réponse en tenant compte de ce format. Pour mémoire, lors de la réception de la réponse par le navigateur, c'est la fonction de rappel du moteur Ajax (`actualiserPage()`) qui va être exécutée pour appliquer les nouveaux résultats à la page Web. C'est donc dans cette même fonction que nous allons ajouter notre code de traitement des résultats.

Pour récupérer individuellement chacun des deux résultats, nous allons utiliser la méthode `split()` qui permet de retourner dans un tableau de variables les différents éléments d'une chaîne séparés par un caractère spécifique. Évidemment, vous avez deviné que le caractère que nous allons utiliser sera « : ». Ainsi, si on applique cette méthode au résultat disponible dans la propriété `responseText` de l'objet XHR, nous disposerons d'un tableau des résultats dans lequel le premier élément (indice 0 du tableau) sera le nom du joueur et le second (indice 1) sera le montant du gain.

```
var nouveauResultat = objetXHR.responseText.split(":");
```

Il est maintenant facile d'affecter ces deux informations aux zones qui les concernent avec la fonction DOM que nous vous avons présenté dans un atelier précédent. Ainsi, la

valeur d'indice 1 du tableau de résultat (soit nouveauResultat[1]) remplacera le contenu de la balise d'identifiant resultat et celui d'indice 0 (soit nouveauResultat[0]) remplacera le contenu de la balise d'identifiant gagnant. À noter la présence de la méthode decodeURI() qui permettra de décoder le format d'URL de la réponse renvoyée par le serveur. Par exemple, si la réponse contient un espace, il est alors remplacé par la suite de caractère %20 lorsqu'elle est formatée en URL par le serveur avant de la renvoyer au navigateur, cette méthode remplacera donc cette suite de caratères par l'espace initial avant de l'intégrer dans la page.

```
remplacerContenu("resultat", decodeURI(nouveauResultat[1]));
remplacerContenu("gagnant", decodeURI(nouveauResultat[0]));
```

Après ces modifications, le code de la fonction actualiserPage() doit être semblable à celui du code 10-4.

Code 10-4 : fonction actualiserPage() :

```
function actualiserPage() {
   if (objetXHR.readyState == 4) {
     if (objetXHR.status == 200) {
       //récupération des résultats dans le tableau nouveauResultat[]
       var nouveauResultat = objetXHR.responseText.split(":");
       //actualisation du nom
       remplacerContenu("resultat", decodeURI(nouveauResultat[1]));
       //actualisation du nom
       remplacerContenu("gagnant", decodeURI(nouveauResultat[0]));
       document.getElementById("info").style.visibility="visible";
       document.getElementById("button").disabled= false;
       document.getElementById("charge").style.visibility="hidden";
     }else{
…
     }
   }
}
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Sous Dreamweaver, ouvrez la page index.html dans le navigateur Firefox en appuyant sur la touche F12. Saisissez ensuite votre nom dans le champ du formulaire (voir repère 1 de la figure 10-2) et appuyez sur le bouton JOUER (voir repère 2 de la figure 10-2). La requête doit alors être envoyée, le bouton JOUER neutralisé et l'animation du chargeur apparaître pendant le temps du traitement. À la réception des résultats, le bouton doit se débloquer, l'animation du chargeur disparaître et les résultats être affichés à l'écran selon la formulation que nous avons définie dans le programme (voir repères 3 et 4 de la figure 10-2).

Nous désirons maintenant simuler un problème de compatibilité du navigateur lors de la création d'un objet XHR. Pour mettre en œuvre cette simulation, nous allons revenir dans Dreamweaver et modifier la fonction creationXHR() du fichier fonctionsAjax.js. Pour cela, commentez la ligne de création de l'objet XHR (en ajoutant « // » en début de ligne) correspondant au navigateur Firefox (et autres navigateurs compatibles).

```
//resultat= new XMLHttpRequest();
```

Figure 10-2

Test du système avec la saisie du nom du joueur

Enregistrez votre fichier et testez de nouveau la page `index.html` dans Firefox. Après son chargement, la page doit s'afficher normalement. Si vous saisissez maintenant votre nom et appuyez sur le bouton JOUER, un message d'erreur JavaScript doit alors apparaître dans la barre d'état située en bas de l'écran (voir repère 1 de la figure 10-3). Dans le cas de notre exemple qui ne nécessite que la saisie d'un simple champ, le temps perdu par l'utilisateur n'est pas très important ; mais imaginez qu'il ait à saisir un long formulaire avant de déclencher la requête de soumission et se rende compte qu'il vient de perdre plusieurs minutes inutilement puisque son navigateur ne prend pas en compte les applications Ajax. Cela risque certainement de le décevoir…

Figure 10-3

Test du système avec simulation d'une erreur de création de l'objet XHR

D'autre part, pour le joueur, le système restera ainsi bloqué et les symptômes du problème doivent se limiter à ce simple avertissement, ce qui est plutôt déroutant, il faut l'avouer.

En ce qui vous concerne, comme vous êtes le développeur de l'application et que vous avez préalablement installé l'extension Firebug sur votre navigateur, vous avez la possibilité d'en savoir plus en cliquant sur l'indication de l'erreur en bas du navigateur. Firebug doit alors s'ouvrir et sa console doit se positionner automatiquement sur la ligne provoquant le problème (voir repère 2 de la figure 10-3). Il est alors facile d'en déduire que l'origine du dysfonctionnement est lié au fait que l'objet XHR ne peut pas être créé avec cette version de navigateur.

Je pense que vous concevez qu'un problème de ce genre doit être contrôlé afin d'informer explicitement l'utilisateur que son navigateur est trop vieux, ou du moins, qu'il n'est pas compatible avec l'application Ajax avant même qu'il ne perde du temps à saisir des informations inutilement. Aussi, nous allons consacrer l'atelier suivant à lever ce problème.

Atelier 10-2 : requête asynchrone avec test du navigateur

Composition du système

Dans l'atelier précédent, nous avons remarqué qu'en cas d'incompatibilité du navigateur avec l'application Ajax, le message d'erreur n'apparaissait que lors de l'envoi de la requête et était pour le moins discret et incompréhensible pour la majorité des utilisateurs. Nous allons voir dans cet atelier comment contrôler ce problème afin d'afficher un message d'erreur explicite à l'écran et surtout détecter la compatibilité du navigateur dès le chargement de la page Web.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure de base avant modification sera identique à celle de l'atelier précédent ;
- d'un fichier JS (`fonctionsAjax.js`) qui contiendra les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contiendra les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification sera identique à celle de l'atelier précédent ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier précédent ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système sera identique à celui de l'atelier précédent hormis le fait qu'en cas d'incompatibilité (réelle ou simulée) du navigateur, un message d'erreur s'affichera dès le chargement initial de la page Web.

Conception du système

Ouvrez la page HTML de l'atelier 10-1 précédent (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap10/atelier10-2/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Pour contrôler la compatibilité du navigateur dès son chargement, nous allons mettre en œuvre une fonction spécifique nommée `testerNavigateur()` que nous appellerons dès la fin du chargement de la page (pour être sûr que tous les éléments de la page sont bien disponibles). Cette fonction étant spécifique à l'application, nous l'enregistrerons dans le fichier `fonctionsMachine.js`.

Pour vérifier que le navigateur est capable de créer un objet XHR, nous allons en créer un (ou du moins essayer si toutefois il n'est pas compatible) dès le début de la fonction. Nous rappelons que la fonction `creationXHR()` renvoie l'objet créé si l'un des tests `try` à fonctionné ou l'état `null` dans le cas contraire. Nous allons donc nous appuyer sur cette valeur de retour pour conditionner un test `if()` dont le bloc contiendra les instructions à exécuter si le navigateur n'est pas compatible (voir code 10-5).

Code 10-5 : Début de la fonction `testerNavigateur()` :

```
objetXHR = creationXHR();
if(objetXHR==null) {
//instructions à exécuter si le navigateur n'est pas compatible
}
```

Ainsi, en cas d'incompatibilité, nous pouvons désactiver le bouton JOUER par exemple avec l'instruction ci-dessous :

```
document.getElementById("button").disabled= true;
```

De même, nous pouvons afficher un message d'erreur dans la zone `info` avec les trois instructions ci-dessous :

```
var erreurNavigateur="Erreur Navigateur : Création d'objet XHR impossible";
remplacerContenu("info", erreurNavigateur);
document.getElementById("info").style.visibility="visible";
```

Comme nous avons déjà utilisé ce type d'instructions dans un atelier précédent pour afficher un message d'erreur serveur, il est inutile de revenir dessus. Une fois créée, la fonction `testerNavigateur()` doit être semblable au code 10-6.

Code 10-6 : fonction `testerNavigateur()` :

```
function testerNavigateur() {
  objetXHR = creationXHR();
  if(objetXHR==null) {
    document.getElementById("button").disabled= true;
    var erreurNavigateur="Erreur Navigateur : Création d'objet XHR impossible";
    remplacerContenu("info", erreurNavigateur);
    document.getElementById("info").style.visibility="visible";
  }
}
```

Pour appeler cette fonction au chargement de la page nous pourrions simplement ajouter un gestionnaire d'événements `onload()` dans la balise `<body>` de la page Web en ajoutant dans la page `index.html` :

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

Cependant, il est préférable d'utiliser un gestionnaire d'événement DOM (utilisant la propriété `onload` de l'objet `document`, voir le chapitre 20 sur le DOM Event pour plus de détail) placé directement dans le code JavaScript et non dans une balise HTML. Cette technique améliore la lisibilité du code et facilite sa maintenance en dissociant parfaitement la structure et le contenu du code du traitement JavaScript. Pour cela, il suffit d'ajouter l'instruction ci-dessous directement dans le code JavaScript de l'application :

```
window.document.onload=testerNavigateur ;
```

D'autre part, pour la même raison que celle invoquée précédemment (séparation de la structure et du traitement JS), nous allons profiter de cette nouvelle fonction `testerNavigateur()`, qui sera appelée à la fin de chaque chargement de la page, pour y intégrer le gestionnaire d'événement `onclick` du bouton JOUER qui était, jusqu'à présent, placé dans la balise HTML `<input>` de la page `index.html`.

Pour cela, il suffit de supprimer le gestionnaire actuel de la page `index.html` (`onclick="jouer();"`) et de le remplacer par l'instruction ci-dessous placée à la fin de la fonction `testerNavigateur()`.

```
function testerNavigateur() {
   …
   document.getElementById("button").onclick=jouer;
}
```

Déclaration d'un gestionnaire d'événement dans le code JS

Les gestionnaires d'événements appliqués à un élément d'une page HTML doivent être déclarés après le chargement de l'élément concerné. En pratique, nous vous conseillons de regrouper ces déclarations dans un gestionnaire `onload` dont le contenu sera exécuté après le chargement complet de la page HTML

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Sous Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Testez le système normalement (vérifiez que la ligne de création de l'objet pour Firefox a bien été décommentée) pour vous assurer que nos modifications n'ont pas perturbé le fonctionnement du système. Ouvrez ensuite le fichier `fonctionsAjax.js` dans Dreamweaver et commentez la ligne de création de l'objet XHR correspondant au navigateur Firefox comme nous l'avons fait dans la seconde partie des tests de l'atelier précédent afin de simuler une erreur.

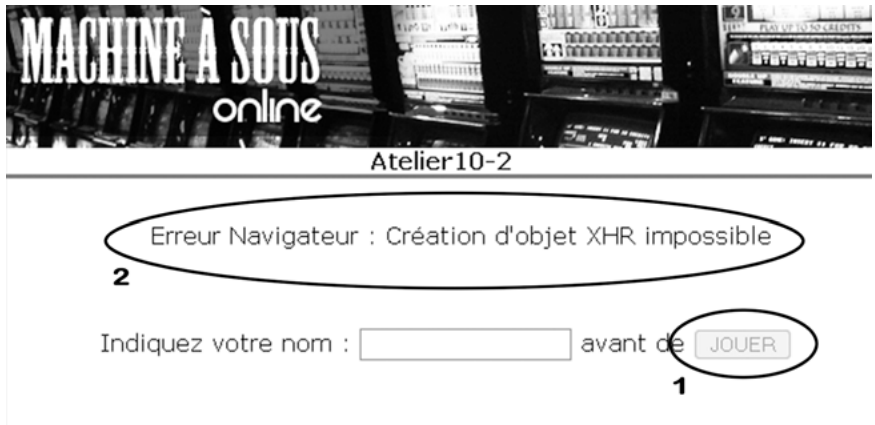
Enregistrez votre fichier et testez de nouveau votre système en appuyant sur F12 après avoir ouvert la page `index.html` dans Dreamweaver. Cette fois, contrairement aux derniers tests de l'atelier précédent, un message d'erreur compréhensible doit apparaître dans la page Web dès le chargement de celle-ci, vous signalant que le navigateur utilisé n'est pas compatible et que l'objet XHR n'a pas pu être créé (repère 2 voir figure 10-4) et le bouton JOUER doit être désactivé (repère 1 voir figure 10-4) afin d'éviter que l'utilisateur tente d'envoyer une requête.

Retournez dans Dreamweaver et rétablissez l'état initial de la fonction `creationXHR()` (en décommentant la ligne de création de l'objet pour Firefox). Sauvegardez votre fichier

puis renouvelez vos tests pour vérifier que tout est bien rétabli et que le système fonctionne de nouveau comme avant.

Figure 10-4

Test du système avec simulation d'une erreur de navigateur



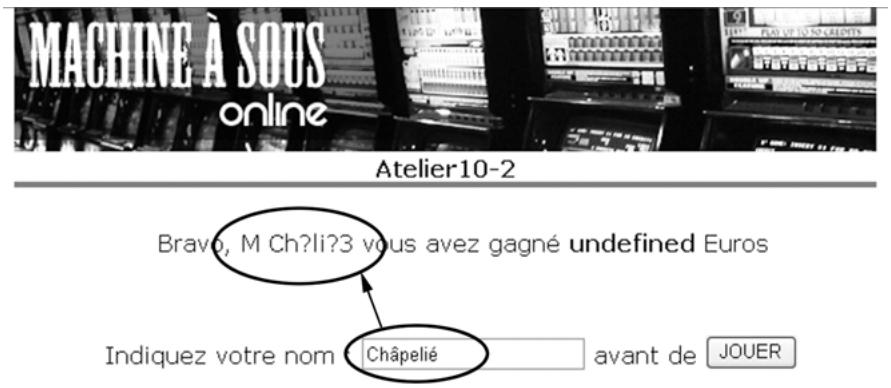
Nous vous proposons maintenant d'utiliser pour vos tests le navigateur Internet Explorer pour s'assurer que nous n'avons pas de problème de cache. Pour faire des tests sous IE, nous vous rappelons que vous pouvez soit utiliser l'extension IE Tab de Firefox, soit sélectionner le navigateur IE dans la liste du bouton Aperçu/Débogage de Dreamweaver. Quelle que soit la méthode utilisée, le résultat devrait être semblable à celui des tests avec Firefox réalisés précédemment, prouvant ainsi que notre anti cache fonctionne aussi avec l'envoi d'un paramètre dans la requête.

Profitons d'être avec le navigateur IE pour faire des tests plus poussés. Pour cela, nous vous suggérons de saisir dans le champ le nom suivant « Châpelié » avant de cliquer sur le bouton JOUER.

Le résultat obtenu est alors pour le moins surprenant (voir figure 10-5). Les caractères accentués du nom semblent avoir complètement paralysé notre système !

Figure 10-5

Test du système avec des caractères accentués dans le nom du joueur



En réalité, nous venons de mettre le doigt sur le problème de l'encodage des données entre le navigateur et le serveur. Nous allons voir dans l'atelier suivant comment lever ce problème.

Atelier 10-3 : requête asynchrone avec gestion de l'encodage

Composition du système

Dans l'atelier précédent, nous avons identifié un problème d'encodage avec le navigateur Internet Explorer lorsque le nom du joueur comportait des caractères accentués. Nous vous proposons maintenant de remédier à ce problème en ajoutant au système les instructions adéquates à une bonne gestion de l'encodage.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure de base avant modification sera identique à celle de l'atelier précédent ;
- d'un fichier JS (`fonctionsAjax.js`) qui contiendra les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contiendra les fonctions spécifiques à l'application Ajax de la machine à sous ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la structure de base avant modification sera identique à celle de l'atelier précédent ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système sera identique à celui de l'atelier précédent hormis le fait que lors de la saisie d'un nom comportant des caractères accentués, celui-ci sera correctement interprété côté serveur mais aussi côté client lors de la réception de la réponse.

Conception du système

Ouvrez la page HTML de l'atelier 10-2 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap10/atelier10-3/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Pour gérer correctement l'encodage des caractères dans une application Ajax, il faut s'assurer que celui-ci est le même côté serveur et côté client (navigateur). Nous pourrions travailler en ISO-8859-1 (Latin-1) ou en UTF-8, l'important est qu'il soit configuré de la même manière de part et d'autre. Rappelons tout de même que l'encodage ISO-8859-1 permet de coder tous (ou presque) les caractères utilisés en Europe occidentale. L'UTF-8, quant à lui, fait partie du standard Unicode qui permet de coder tous les caractères utilisés dans le monde entier. La partie commune de ces deux types de codage correspond aux 128 caractères ASCII comprenant tous les caractères non accentués d'un clavier d'ordinateur. Cela explique notamment qu'il est rare d'avoir des problèmes d'encodage avec les lettres sans accent, ce qui n'est pas le cas dès que l'on commence à utiliser des caractères accentués ou des symboles particuliers.

Chaque système d'encodage a des avantages et des inconvénients mais il serait trop long de les énumérer ici. Aussi, dans cet atelier, nous allons employer l'encodage UTF-8 et allons voir comment le paramétrer côté client, pour les informations issues d'un champ

de formulaire par exemple puis envoyées au serveur et, à l'inverse côté serveur pour les informations retournées dans les réponses HTTP du serveur.

Commençons par le client : l'encodage d'une application Ajax pouvant différer d'un navigateur à l'autre (comme nous venons de le constater avec IE), il est préférable de prendre les devants et d'encoder nous même en UTF-8 les paramètres émis lors d'une requête. Pour cela, nous utiliserons la fonction `encodeURIComponent()` qui encode une chaîne de caractères en UTF8.

Pour vous faciliter la tâche, sachant que les informations émises en paramètre dans une requête HTTP seront très souvent issues d'un champ de formulaire ou du moins d'une balise pouvant être identifiée par son identifiant, nous allons vous proposer de créer une petite fonction qui retournera le contenu d'un élément codé automatiquement en UTF-8 en indiquant simplement l'identifiant de l'élément concerné en paramètre. Les instructions de cette fonction sont très simples, la première permet de récupérer le contenu de l'élément à l'aide de sa propriété `value` et la seconde permet de lui appliquer le code UTF-8 avec `encodeURIComponent()` avant de retourner le résultat ainsi codé (voir code 10-7).

Code 10-7 : fonction `codeContenu()` :

```
function codeContenu(id) {
 var contenu=document.getElementById(id).value;
 return encodeURIComponent(contenu);
}
```

Cette fonction pouvant être exploitée par la suite dans d'autres applications Ajax sans modification, nous allons donc l'inclure dans le fichier `fonctionsAjax.js`. Ouvrez pour cela ce fichier, saisissez le code 10-8 à la suite des autres fonctions et enregistrez votre modification.

Par contre, l'utilisation de cette fonction devra se faire dans le fichier `fonctionsMachine.js`. Ouvrez le fichier et localisez la ligne de code qui permet de créer la variable `parametres` dans la fonction `jouer()`. Il suffit ensuite de remplacer la variable `nom` par un appel à la fonction précédemment créée pour encoder en UTF-8 le paramètre envoyé au serveur.

```
var parametres = "nom="+ codeContenu("nom") + "&anticache="+temps;
```

Evidemment, l'instruction précédent cette ligne de code, qui permettait de récupérer la valeur de l'élément `nom` peut maintenant être supprimée puisqu'elle est désormais intégrée à la fonction. La fonction `jouer()` une fois modifiée doit être semblable au code 10-8.

Code 10-8 : fonction `jouer()` :

```
function jouer() {
   objetXHR = creationXHR();
   var temps = new Date().getTime();
   var parametres = "nom="+ codeContenu("nom") +
                    "&anticache="+temps ;
   objetXHR.open("get","gainAleatoire.php?"+parametres, true);
   objetXHR.onreadystatechange = actualiserPage;
   document.getElementById("button").disabled= true;
   document.getElementById("charge").style.visibility="visible";
   objetXHR.send(null);//envoi de la requête
}
```

Étudions maintenant le côté serveur et ouvrons le fichier `gainAleatoire.php` dans Dreamweaver. Comme nous l'avons mentionné précédemment, l'encodage doit être le même côté client et côté serveur. L'encodage par défaut du serveur pouvant varier selon sa

configuration, il est préférable de s'assurer que la réponse sera bien renvoyée en UTF-8 en ajoutant un en-tête adapté. Dans notre fichier PHP, nous avions déjà paramétré l'en-tête `Content-type` avec `text/plain` pour indiquer que le type de réponse était du texte simple. Nous allons maintenant ajouter à cet en-tête une seconde propriété `charset=utf-8` afin de préciser que les données de la réponse seront encodées en UTF-8. Ainsi, côté client, à la réception de cet en-tête, JavaScript pourra interpréter correctement l'information contenue dans le corps de la réponse car il saura que l'encodage utilisé est UTF-8.

Après sa modification, le fichier serveur `gainAleatoire.php` doit être semblable à celui du code 10-9.

Code 10-9 : fichier `gainAleatoire.php` :

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
sleep(2);
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
$gain = rand(0,100);
$resultat=$nom.':'.$gain;
echo $resultat;
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur Internet Explorer.

Test du système

Ouvrez la page `index.html` dans le navigateur Internet Explorer en utilisant le bouton `Aperçu/Débogage` de Dreamweaver et en sélectionnant le navigateur IE dans la liste. Nous allons renouveler le test de l'atelier précédent qui avait abouti à une erreur en saisissant le nom « Châpelié » dans le champ du navigateur puis en cliquant sur le bouton JOUER. Cette fois, le système doit fonctionner sans erreurs et le nom correctement orthographié doit s'afficher dans le message de la réponse (voir figure 10-6).

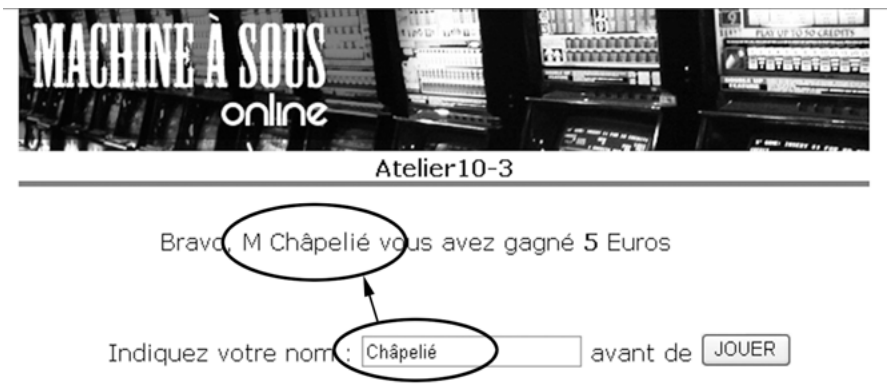


Figure 10-6

Test de l'envoi de caractères accentués après intégration du système de gestion de l'encodage en UTF-8

Voyons maintenant le cas d'un utilisateur distrait qui oublie de saisir son nom. Dans ce cas, le système ne génère pas d'erreur mais aucun nom n'apparaîtra dans le message du résultat. Si pour notre application cela n'est pas grave, il peut en être autrement dans des

systèmes où l'information est nécessaire au traitement des données côté serveur. Aussi, dans l'atelier suivant, nous vous proposerons d'étudier la mise en place d'un contrôle des données saisies par l'utilisateur.

Atelier 10-4 : requête asynchrone avec contrôle de la saisie

Composition du système

Dans l'atelier précédent, nous avons remarqué que si l'utilisateur oubliait de saisir son nom dans le champ du formulaire, la requête était tout de même envoyée au serveur. Nous désirons maintenant changer ce fonctionnement et forcer l'utilisateur à saisir son nom.

Cette structure est composée :

- d'une page HTML (`index.html`) identique à celle de l'atelier précédent ;
- d'un fichier JS (`fonctionsAjax.js`) qui contiendra les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contiendra les fonctions spécifiques à l'application Ajax de la machine à sous ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier précédent ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système sera identique à celui de l'atelier précédent, hormis le fait que si l'utilisateur oublie de saisir son nom dans le champ du formulaire, une boîte d'alerte s'affichera pour lui rappeler que le nom est obligatoire et le champ concerné deviendra rouge.

Conception du système

Ouvrez la page HTML de l'atelier 10-3 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap10/atelier10-4/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Nous pourrions traiter ce contrôle de saisie de différentes manières. La façon la plus simple serait d'insérer un simple gestionnaire d'événement (`onChange` ou `onBlur` par exemple) sur le champ concerné afin de déclencher une fonction de contrôle du champ au fil de la saisie et afficher un message d'alerte si nécessaire. Mais dans le cadre de cet atelier, nous avons décidé de faire le contrôle à la fin de la saisie, au moment où l'utilisateur appuiera sur le bouton JOUER, afin de bloquer l'envoi de la requête si le test est négatif.

La fonction `jouer()` étant appelée par une action sur le bouton JOUER, c'est au début de cette fonction que nous allons placer la procédure de contrôle. Si, lors du contrôle, le champ est détecté comme vide, nous afficherons une simple boîte de dialogue rappelant à l'utilisateur que la saisie du champ `nom` est obligatoire et nous changerons la couleur de ce même champ pour bien le mettre en évidence.

Ouvrez le fichier `fonctionsMachine.js` et placez-vous au début du bloc de la fonction `jouer()`. Avant de faire le test, nous allons récupérer la valeur du champ de saisie dans une variable que nous allons nommer `nom`.

```
var nom=document.getElementById("nom").value;
```

Maintenant que le contenu du champ est connu, nous pouvons mettre en place le test qui va conditionner le message d'alerte et la modification de la couleur du champ en rouge. À noter que si vous désirez modifier avec le DOM une propriété de style dont le nom équivalent en CSS est composé (comme par exemple : `background-color`), il faudra transformer le nom de cette propriété avant de l'appliquer à l'objet `style` en supprimant le tiret et en ajoutant une majuscule à la première lettre du second mot (soit pour l'exemple considéré précédemment : `backgroundColor`, voir si besoin le tableau d'équivalence de ces propriétés dans la partie consacrée à la manipulation des styles du chapitre 20 sur la gestion du DOM).

```
if(nom==""){
   document.getElementById("nom").style.backgroundColor="red";
   alert("Attention : vous devez saisir votre nom avant de jouer");
```

Une fois que les actions d'information de l'utilisateur sont exécutées, nous devons sortir de la fonction `jouer()` pour éviter d'envoyer la requête. Pour cela, nous ajouterons l'instruction `return` juste avant la fin du bloc conditionné par le test `if()`.

```
return null;
```

Si nous nous arrêtons à ces seules instructions, le champ de saisie restera toujours rouge même après que l'utilisateur ait corrigé son erreur et renseigné son nom. Il faut donc prévoir une instruction d'initialisation par défaut de la couleur du champ en blanc au tout début de la fonction `jouer()` (voir le code 10-10).

Code 10-10 : fonction `jouer()` :

```
jouer() {
  document.getElementById("nom").style.backgroundColor="white";
  var nom=document.getElementById("nom").value;
  if(nom==""){
     document.getElementById("nom").style.backgroundColor="red";
     alert("Attention : vous devez saisir votre nom avant de jouer");
     return null;
  }
… // reste de la fonction jouer
}
```

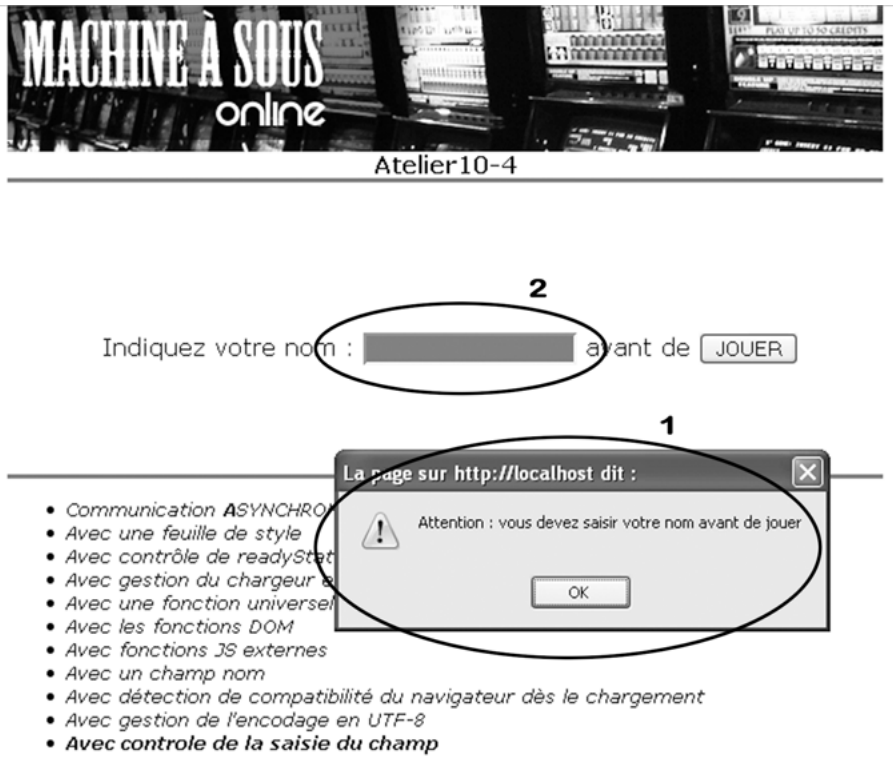
Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer le fichier `fonctionsMachine.js` et tester le nouveau système dans un navigateur.

Test du système

Sous Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Ne saisissez pas votre nom et appuyez directement sur le bouton JOUER. Dès que le bouton est actionné, le champ de saisie doit changer de couleur et devenir rouge (voir repère 2 de la figure 10-7) puis une boîte d'alerte doit apparaître pour vous indiquer que le champ `nom` est obligatoire (voir repère 1 de la figure 10-7).

Figure 10-7

Test du contrôle de saisie du champ nom



Validez la boîte d'alerte en appuyant sur le bouton OK puis saisissez maintenant votre nom dans le champ rouge. Cliquez de nouveau sur le bouton JOUER pour valider votre saisie. Le champ doit être réinitialisé en blanc et la requête doit être envoyée normalement.

Atelier 10-5 : double requête asynchrone avec actualisation automatique

Composition du système

L'objectif pédagogique de cet atelier est double :

- apprendre à créer un système qui s'actualisera automatiquement avec des données issues du serveur sans qu'un événement généré par l'utilisateur ne soit nécessaire ;
- savoir faire cohabiter deux requêtes en parallèle.

Concrètement, dans notre application de machine à sous, nous allons mettre en place un système qui affichera toutes les 6 secondes le cumul des gains du joueur tout en lui permettant de continuer à jouer avec l'application déjà opérationnelle de l'atelier précédent.

Cette structure est composée :

- d'une page HTML (`index.html`) identique à celle de l'atelier précédent ;
- d'un fichier JS (`fonctionsAjax.js`) qui contiendra les fonctions communes à tous les moteurs Ajax ;

- d'un fichier JS (`fonctionsMachine.js`) qui contiendra les fonctions spécifiques à l'application Ajax de la machine à sous ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier précédent ;
- d'un nouveau fichier serveur PHP (`gainCumul.php`) ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier précédent ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Parallèlement au jeu tel qu'il fonctionnait dans l'atelier précédent, l'interface comprendra une nouvelle zone d'information indiquant le cumul des gains de l'utilisateur dont le nom figure dans le champ de saisie du formulaire. L'actualisation du cumul des gains étant réalisée automatiquement, l'utilisateur n'aura pas à s'en préoccuper. Toutes les 6 secondes cette information s'actualisera par rapport au cumul des gains de l'utilisateur conservé côté serveur dans une variable de session.

Conception du système

Ouvrez la page HTML de l'atelier 10-4 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap10/atelier10-5/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Pour réaliser ce nouveau système, nous allons devoir mettre en place un second moteur Ajax côté client et un autre fichier serveur en PHP avec lequel le moteur communiquera.

Commençons par créer les fonctions de ce deuxième moteur Ajax. Il sera structuré de la même manière que le premier moteur, à savoir une fonction de déclenchement du moteur nommée `cumul()` qui créera l'objet XHR puis enverra la requête au serveur et une autre de rappel, nommée `actualiserCumul()`, qui sera automatiquement sollicitée lorsque la réponse du serveur sera réceptionnée.

Ces deux fonctions étant spécifiques au système de la machine à sous, nous allons donc les ajouter à la suite des fonctions du fichier `fonctionsMachine.js`. Les instructions de la fonction `cumul()` seront très semblables à celle de la fonction `jouer()` au détail près que l'objet XHR créé portera un nom différent (`objetXHR2`) pour éviter les conflits avec le premier objet.

```
objetXHR2 = creationXHR();
```

De même, pour connaître le cumul des gains d'un joueur, nous aurons besoin d'envoyer au serveur le nom du joueur en paramètre. Nous aurons donc, ici aussi, une similitude avec la fonction `jouer()` pour la préparation des paramètres et leur intégration dans la méthode `open()` de l'objet. À noter que, cette fois, le fichier serveur ne sera plus `gainAleatoire.php` mais `gainCumul.php` (nous développerons ce nouveau fichier PHP plus tard).

```
var temps = new Date().getTime();
var parametres = "nom="+ codeContenu("nom") +
                 "&anticache="+temps;
objetXHR2.open("get","gainCumul.php?"+parametres, true);
```

La fonction de rappel portant le nom `actualiserCumul()`, nous allons devoir déclarer cette information avant d'envoyer la requête. Pour cela, il suffit d'affecter le nom de cette fonction de rappel à la propriété `onreadystatechange` de l'objet.

```
objetXHR2.onreadystatechange = actualiserCumul;
```

Pour signaler à l'utilisateur que le moteur est en communication avec le serveur, nous utiliserons aussi la même animation que dans l'autre moteur, elle sera simplement personnalisée par un identifiant différent (`charge2`). Cette animation devant apparaître au moment de l'envoi de la requête, nous insérerons ici une instruction qui permettra de changer la propriété `visibility` de son style afin de rendre visible cette seconde animation.

```
document.getElementById("charge2").style.visibility="visible";
```

Après tous ces préparatifs, nous pouvons maintenant envoyer la requête au serveur en utilisant la méthode `send()` appliquée à `objetXHR2`.

```
objetXHR2.send(null);
```

Pour actualiser périodiquement la valeur du cumul, nous allons devoir maintenant ajouter une instruction que nous n'avions pas dans la précédente fonction `jouer()`. Il s'agit de l'instruction `setTimeout()` qui appellera la fonction `cumul()` (définie dans le premier paramètre), dans laquelle se trouve l'instruction elle-même, au bout d'un temps de 6 secondes (définie dans le second paramètre : 6000 millisecondes).

```
setTimeout("cumul()",6000);
```

Ainsi, il suffira d'appeler une première fois la fonction `cumul()` pour que le système soit autonome et que la fonction `cumul()` soit exécutée toutes les 6 secondes. Pour réaliser cet appel initial, nous allons placer `cumul()` à la fin de la fonction `testerNavigateur()` qui est, elle-même, appelée lors du chargement de la page par le gestionnaire d'événement `onload`.

```
function testerNavigateur() {
…
 cumul();
}
```

Au terme de votre saisie, la fonction `cumul()` doit être semblable au code 10-11 (nous avons mis en gras les instructions qui diffèrent de la fonction `jouer()` du premier objet).

Code 10-11: fonction `cumul()` :

```
function cumul() {
  objetXHR2 = creationXHR();
  var temps = new Date().getTime();
  var parametres = "nom="+ codeContenu("nom") +
                   "&anticache="+temps;
  objetXHR2.open("get","gainCumul.php?"+parametres, true);
  objetXHR2.onreadystatechange = actualiserCumul;
  document.getElementById("charge2").style.visibility="visible";
  objetXHR2.send(null);//envoi de la requête
  //---------------------------------------
  setTimeout("cumul()",6000);//timer de 6 secondes
}
```

Nous allons passer maintenant à la construction de la fonction de rappel de ce deuxième objet XHR2. Ici aussi, bon nombre d'instructions seront communes à la fonction de rappel du premier objet XHR (`actualiserPage()`). Commençons par mettre en place la

déclaration de la fonction `actualiserCumul()` et les deux tests `if()` qui permettront d'exécuter les instructions qui suivent lorsque le résultat sera disponible dans le navigateur (`readyState==4`) et que nous aurons vérifié que le transfert HTTP se sera déroulé correctement (`status==200`).

```
function actualiserCumul() {
  if (objetXHR2.readyState == 4) {
   if (objetXHR2.status == 200) {
```

Pour l'affichage du cumul des gains, nous nous contenterons d'afficher sa valeur sans rappeler le nom du joueur auquel il correspond. Nous n'aurons donc besoin que d'un seul résultat dans la réponse, que nous récupérerons dans une variable nommée `cumulGain`.

```
var cumulGain = objetXHR2.responseText;
```

Une fois le résultat connu, nous devons actualiser sa valeur dans la balise `<span>` qui lui est attribuée (identifiant `cumul`). Pour cela, nous utiliserons la fonction `remplacerContenu()` que nous avions développée dans un atelier précédent.

```
remplacerContenu("cumul", cumulGain);
```

De même, le transfert HTTP étant maintenant terminé, nous devons rendre invisible l'animation du deuxième chargement.

```
document.getElementById("charge2").style.visibility="hidden";
```

La fonction de rappel de l'application d'actualisation du cumul est maintenant terminée et devrait être semblable au code 10-12 (nous avons mis en gras les instructions qui diffèrent de la fonction `actualiserPage()` du premier objet).

Code 10-12 : fonction `actualiserCumul()` :

```
function actualiserCumul() {
  if (objetXHR2.readyState == 4) {
    if (objetXHR2.status == 200) {
     var cumulGain = objetXHR2.responseText;
     remplacerContenu("cumul", cumulGain);
     document.getElementById("charge2").style.visibility="hidden";
    }
  }
}
```

Les deux fonctions du nouveau moteur étant terminées, vous pouvez enregistrer votre fichier `fonctionsMachine.js` et ouvrir maintenant le fichier `index.html` dans lequel nous allons préparer la zone d'affichage du cumul et son animation de chargement.

En mode Code, ajoutez une nouvelle balise `<div>` à la suite de la balise du formulaire en utilisant le code ci-dessous.

```
<div id="zoneCumul">Votre cumul des gains est de <span id="cumul" >0
➥</span> Euros </div>
```

Vous remarquerez que l'identifiant de cette nouvelle balise `<div>` est `zoneCumul` (nous en aurons besoin prochainement pour paramétrer son style qui positionnera la balise dans la page Web). De même, cette balise `<div>` comprend elle-même une balise `<span>` d'identifiant `cumul` qui contiendra par la suite la valeur du cumul des gains renvoyée par le serveur.

Placez ensuite le code de la seconde animation de chargement après celle déjà en place dans la page.

```
<img id="charge" src="chargeur.gif" />
<img id="charge2" src="chargeur.gif" />
```

Enregistrez les modifications de la page `index.html`. Nous allons maintenant passer aux paramétrages des styles des éléments que nous venons d'ajouter. Pour cela, ouvrez la feuille de styles `style.css` et saisissez les nouveaux styles suivants à la suite des règles déjà en place (code 10-13).

Code 10-13 : règles de styles à ajouter dans `style.css` :

```
#zoneCumul {
  position: absolute;
  left: 40px;
  top: 180px;
  width: 530px;
  height: 25px;
}
#cumul {
  font-weight:bold;
}
#charge2 {
  position: absolute;
  left: 470px;
  top: 180px;
  visibility: hidden;
}
```

La première règle de style `#zoneCumul` permet de positionner la zone dans laquelle figurera le texte d'information et la valeur du cumul d'une manière absolue par rapport au conteneur `#page`. Il en sera de même pour `#charge2` qui correspond au positionnement de la seconde animation de chargement mais pour lequel nous ajouterons en plus une directive qui la rendra initialement invisible (`visibility: hidden`). Enfin, le style `#cumul` permet d'afficher en gras la valeur du cumul pour bien la mettre en évidence. Une fois la saisie terminée, enregistrez votre fichier et revenez éventuellement à la page `index.html` en mode Création pour constater l'incidence de ces styles sur la disposition des éléments dans la page (attention, l'animation de chargement étant configurée par défaut comme invisible, il faudra changer la valeur de sa propriété à `visible` si vous désirez la voir apparaître pour vos vérifications).

Il est maintenant grand temps de passer du côté serveur. Cependant, avant de créer le nouveau fichier qui va gérer les requêtes de ce second moteur, nous allons ouvrir le fichier `gainAleatoire.php` afin d'ajouter les instructions qui permettront de mémoriser le cumul des gains dans des variables de session.

En PHP, dès lors que vous utilisez des variables de session (que ce soit pour les créer, les modifier ou les utiliser) il faut ajouter la fonction `session_start()` au début du code. Cette fonction permet d'activer la gestion des sessions dans la page et de pouvoir ainsi les manipuler comme de simples informations stockées dans un tableau de variables (tableau `$_SESSION`).

```
<?php
session_start();
```

Le calcul du cumul sera très facile à mettre en œuvre, car nous utiliserons un simple opérateur d'affectation (+=) dans sa forme compacte qui permet d'ajouter la valeur située à droite (soit `$gain`) à la valeur actuelle de celle de gauche (soit le cumul stocké dans `$_SESSION[$nom]`) avant d'effectuer l'affectation habituelle (revoir si besoin le chapitre sur les ressources PHP à la fin de cet ouvrage). Nous ajouterons donc cette instruction juste après le calcul du gain comme l'illustre le code ci-dessous.

```
$gain = rand(0,100);
$_SESSION[$nom]+=$gain;
```

Ainsi, à chaque fois qu'un utilisateur jouera, son nouveau gain sera cumulé dans une variable de session dont la clé sera son nom. Nous pourrons, avec cette technique, avoir plusieurs variables de session si au cours de la même session plusieurs joueurs différents utilisent l'application (par exemple : `$_SESSION['Defrance']` et `$_SESSION['Dupond']`).

Enregistrez ce fichier et ouvrez maintenant une nouvelle page PHP que vous allez nommer `gainCumul.php`. Nous commencerons par ajouter la fonction `session_start()` en début du fichier de sorte à pouvoir accéder à la valeur des cumuls mise à jour par le précédent fichier.

```
session_start();
```

Pour la suite, la structure de la page sera semblable à celle du premier fichier serveur (revoir si besoin les explications de chacune de ces lignes dans les ateliers précédents).

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
sleep(2);
```

La requête du second moteur Ajax envoyant le nom du joueur avec le même paramètre (`nom`), nous aurons de la même manière besoin d'une structure de test semblable à celle du premier fichier pour récupérer cette information dans une variable locale `$nom`.

```
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
```

Comme nous disposons maintenant du nom du joueur et que l'accès aux variables de session a été activé, nous pouvons récupérer la valeur du cumul des gains par une simple affectation.

```
$resultat = $_SESSION[$nom];
```

Toutefois, pour éviter des messages d'erreur dans le cas où la variable de session de l'utilisateur n'existe pas, nous conditionnerons cette affectation par un test `if()` utilisant la fonction `isset()` (retourne `true` si la variable contenue dans son paramètre existe). Si le test se révélait négatif nous affecterions alors la valeur zéro au résultat grâce à la structure `else` qui complète celle du `if()`.

```
if(isset($_SESSION[$nom])) $resultat = $_SESSION[$nom];
else $resultat=0;
```

Le traitement étant terminé, le résultat est prêt à être renvoyé au navigateur. Pour cela, il suffit de l'afficher à l'aide d'une simple fonction `echo`.

```
echo $resultat;
```

Après la saisie de ces instructions, le contenu du fichier `gainCumul.php` doit être semblable au code 10-14.

Code 10-14 : fonction `gainCumul.php` :

```
session_start();
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
sleep(2);
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
if(isset($_SESSION[$nom])) $resultat = $_SESSION[$nom];
else $resultat=0;
echo $resultat;
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer vos fichiers et tester le nouveau système dans un navigateur.

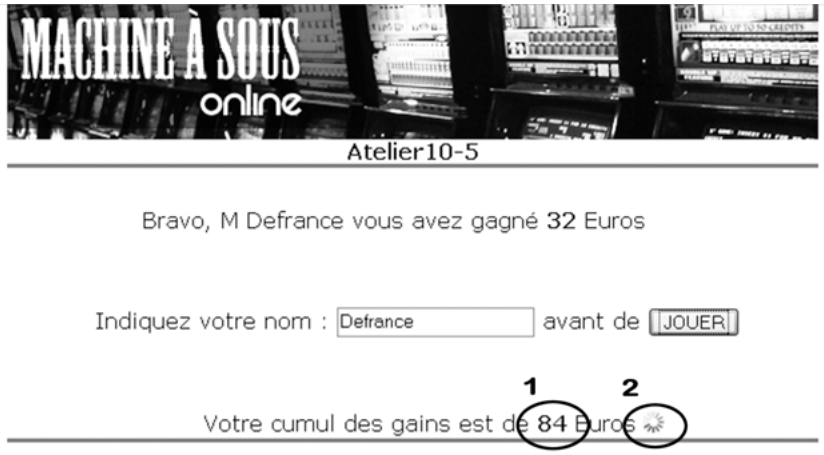
Test du système

Sous Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Dès le chargement de la page, les requêtes d'actualisation du cumul des gains commencent à solliciter le fichier `gainCumul.php` toutes les 6 secondes. Les différentes requêtes envoyées sont facilement identifiables grâce au chargeur du cumul qui s'affiche pendant le temps de traitement à droite de la zone d'affichage du cumul des gains (voir repère 2 de la figure 10-8). Cependant, comme le nom du joueur n'a pas encore été saisi, le résultat renvoyé est toujours égal à zéro.

Saisissez maintenant votre nom dans le champ correspondant et cliquez sur le bouton JOUER. Après le traitement de votre requête, le nouveau gain précédé de votre nom doit alors s'afficher dans la zone de résultat comme dans les ateliers précédents. Dès lors, la prochaine requête d'actualisation du cumul renverra cette valeur dans la zone d'affichage du cumul des gains en dessous du formulaire (voir repère 2 de la figure 10-8). Si vous renouvelez votre action sur le bouton JOUER, une nouvelle valeur de gain s'affichera dans la zone de résultat et la requête d'actualisation du cumul qui suivra, renverra cette fois la somme des deux gains réalisés (voir repère 1 de la figure 10-8).

Figure 10-8

Test du système avec l'actualisation automatique du cumul des gains



11

Applications Ajax-PHP avec paramètres POST

Jusqu'à maintenant, nous avons réalisé des requêtes Ajax avec la méthode `GET`. Cependant, le format d'URL étant imposé et le nombre de caractères envoyés par cette méthode étant limité, il devient alors vite intéressant d'utiliser la méthode `POST` si vous désirez envoyer des données volumineuses dans un autre format que l'encodage d'URL (XML ou JSON par exemple).

Pour vous initier à ces différentes techniques d'envoi de paramètres avec la méthode `POST`, nous allons maintenant les illustrer en reprenant à chaque fois la même application de la machine à sous que dans le chapitre précédent mais avec cette fois deux paramètres à envoyer (nom et prénom).

Atelier 11-1 : requête asynchrone POST avec un champ texte

Composition du système

Pour bien comprendre les différences entre une application Ajax utilisant la méthode `GET` et une autre utilisant la méthode `POST`, nous allons réaliser, pour commencer, la même application que dans l'atelier 10-4 mais avec deux champs de saisie (nom et prénom) et l'utilisation de la méthode `POST` pour envoyer la requête au serveur.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure est identique à celle de l'atelier 10-4 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous, dont la structure de base avant modification est identique à celle de l'atelier 10-4 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 10-4 mais qui est modifié pour gérer les deux paramètres de la requête `POST` ;

- d'une feuille de styles (`style.css`) identique à celle de l'atelier 10-4 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement est semblable à celui de l'atelier 10-4 hormis le fait que dans cet atelier, il faut saisir le prénom en plus du nom et que l'envoi de la requête va se faire avec la méthode `POST` et non plus `GET` comme dans le chapitre précédent.

Conception du système

Ouvrez la page HTML de l'atelier 10-4 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap11/atelier11-1/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Avant de modifier les scripts, nous devons ouvrir le fichier `index.html` et ajouter un champ de saisie supplémentaire pour le prénom. Pour cela, passez en mode Création et faites glisser un champ de saisie dans le formulaire depuis l'onglet Formulaire de la barre d'outils Insertion située en haut de l'interface de Dreamweaver. Nommez ensuite ce nouveau champ `prenom` à l'aide du panneau Propriétés, après l'avoir sélectionné.

Pour transformer notre requête `GET` en une requête `POST`, nous allons devoir modifier le code de la fonction `jouer()` qui déclenche l'envoi de la requête. Ouvrez pour cela le fichier `fonctionsMachine.js` et localisez la fonction `jouer()` dans le code de la page. Avec la méthode `POST`, il n'est plus utile de se prémunir contre un éventuel problème de cache car il est géré différemment par les navigateurs lorsque la requête est envoyée en `POST`. Nous allons donc supprimer la ligne qui permet de calculer la variable temps et de l'ajouter à la suite du paramètre d'URL. Par contre, comme nous avons désormais un second champ `prenom` à gérer, il faut ajouter sa valeur aux paramètres envoyés en respectant la syntaxe de l'encodage d'URL de sorte à avoir au final une chaîne de caractères semblable à celle de l'exemple ci-dessous :

```
nom=Defrance&prenom=Jean-Marie
```

Nous réalisons cette chaîne par concaténation à l'aide de l'instruction suivante puis nous allons l'affecter à la variable `parametres` :

```
var parametres = "nom="+ codeContenu("nom")+"&prenom="+ codeContenu("prenom");
```

Désormais, la variable `parametres` doit contenir les couples de variables `nom` et `prenom` séparées de leur valeur par un signe égal et reliés entre eux par une esperluette (&).

La méthode `open()` de l'objet XHR doit évidemment être modifiée de sorte à indiquer que nous allons envoyer les paramètres avec `POST` et non plus `GET`. De même, les paramètres n'étant plus envoyés dans l'URL (comme c'était le cas en `GET`) il n'est donc plus utile de les ajouter à la suite du nom du fichier serveur. Les deux premiers arguments de la méthode `open()` doivent donc être modifiés en rapport, ce qui donne l'instruction suivante :

```
objetXHR.open("post","gainAleatoire.php", true);
```

Avec la méthode `POST`, les données sont envoyées dans le corps de la requête et il convient d'ajouter un en-tête supplémentaire nommé `Content-Type` pour indiquer le type du contenu qui va être envoyé au serveur. Habituellement dans un formulaire `POST` traditionnel,

c'est le navigateur qui se charge de transmettre cette information au serveur automatiquement dès que le formulaire est validé. La valeur envoyée comme type de contenu par le navigateur est alors la suivante :

```
application/x-www-form-urlencoded
```

Nous allons donc devoir envoyer la même information dans le cas de notre requête Ajax. Pour cela nous allons utiliser la méthode `setRequestHeader()` de l'objet XHR qui permet de configurer des en-têtes de la requête Ajax en indiquant le nom de l'en-tête à créer dans le premier argument et la valeur à lui affecter dans le second comme l'illustre l'instruction ci-dessous :

```
objetXHR.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
```

Il ne reste plus maintenant qu'à insérer les paramètres à envoyer dans l'argument de la méthode `send()`. Comme nous avons déjà préparé ces paramètres dans la variable `parametres`, il suffit simplement d'indiquer son nom dans l'argument de la fonction.

```
objetXHR.send(parametres);//envoi de la requête
```

Code 11-1 : Code complet de la fonction `jouer()` après nos modifications (les modifications à apporter au code du chapitre précédent sont indiquées en gras) :

```
function jouer() {
  objetXHR = creationXHR();
  var parametres = "nom="+ codeContenu("nom")+"&"+"prenom="+ codeContenu("prenom");
  objetXHR.open("post","gainAleatoire.php", true);
  objetXHR.onreadystatechange = actualiserPage;
  objetXHR.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
  document.getElementById("button").disabled= true;
  document.getElementById("charge").style.visibility="visible";
  objetXHR.send(parametres);//envoi de la requête
}
```

La modification du fichier `fonctionsMachine.js` est terminée, vous pouvez maintenant l'enregistrer et ouvrir le fichier serveur `gainAleatoire.php` afin de le modifier pour gérer le second paramètre `prenom` envoyé dans la requête. Pour cela nous allons devoir ajouter une seconde instruction de récupération de paramètre HTTP dans la variable `$_REQUEST`.

```
if(isset($_REQUEST['nom'])) $nomJoueur=$_REQUEST['nom'];
else $nomJoueur="inconnu";
if(isset($_REQUEST['prenom'])) $prenomJoueur=$_REQUEST['prenom'];
else $prenomJoueur="inconnu";
```

Remarquez au passage que nous avons renommé les variables locales en `$nomJoueur` et `$prenomJoueur` de sorte à pouvoir utiliser `$nom` pour mémoriser la concaténation de ces deux données comme l'illustre le code ci-dessous. Ainsi, désormais nous retournons au navigateur une chaîne de caractères contenant le nom et le prénom du joueur et non uniquement son nom comme dans le chapitre précédent.

```
$nom=$prenomJoueur." ".$nomJoueur;
```

Le reste du code reste inchangé et une fois modifié, le fichier `gainAleatoire.php` doit être semblable au code 11-2.

Code 11-2 : fichier `gainAleatoire.php` :

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
```

```
sleep(2);
if(isset($_REQUEST['nom'])) $nomJoueur=$_REQUEST['nom'];
else $nomJoueur="inconnu";
if(isset($_REQUEST['prenom'])) $prenomJoueur=$_REQUEST['prenom'];
else $prenomJoueur="inconnu";
$gain = rand(0,100);
$nom=$prenomJoueur." ".$nomJoueur;
$resultat=$nom.':'.$gain;
echo $resultat ;
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Dans Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Le fonctionnement du nouveau système doit être semblable à celui de l'atelier 10-4, sauf que cette fois vous devez saisir votre prénom en plus du nom et que les paramètres sont envoyés avec la méthode `POST` et non `GET`.

Pour observer les transferts de données entre le navigateur et le serveur, activez Firebug et cliquez sur l'onglet Console. Effacez les différents enregistrements précédents (s'il y en a) en utilisant la fonction Clear du menu puis cliquez de nouveau sur le bouton JOUER. La requête `POST` du fichier PHP doit apparaître dans la fenêtre de Firebug, cliquez sur le + qui précède ce fichier pour le dérouler, puis sur l'onglet Post pour voir apparaître la valeur du champ nom qui a été envoyée au serveur (voir figure 11-1). Vous pouvez ensuite visualiser les données renvoyées par le serveur en cliquant sur l'onglet Response, mais elles sont semblables à celles que nous avons eu jusqu'à présent hormis que cette fois le prénom du joueur est attaché à son prénom (par exemple `Jean-Marie Defrance : 95`).

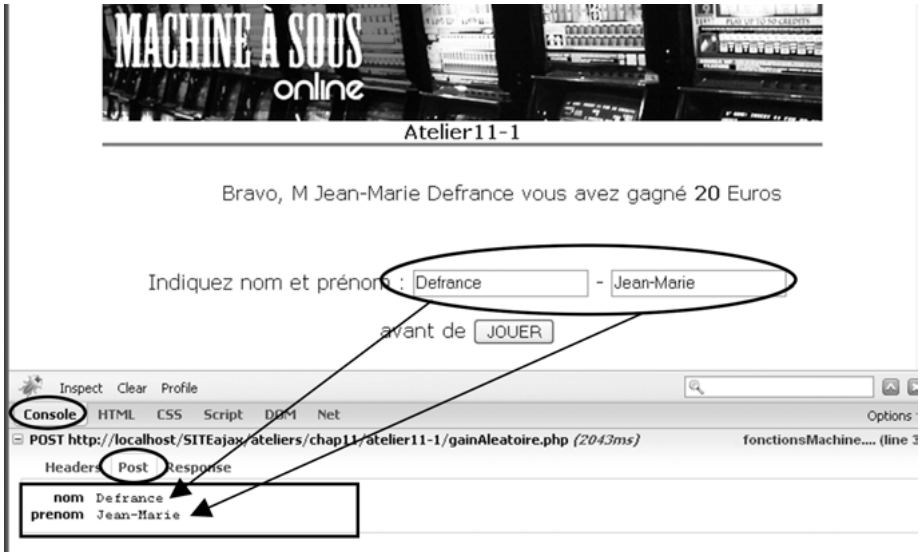


Figure 11-1

Test d'une requête asynchrone POST avec des paramètres au format texte

Atelier 11-2 : requête asynchrone POST avec paramètres en XML

Composition du système

Avec une requête Ajax utilisant la méthode `POST`, nous ne sommes plus limité par la taille des paramètres à envoyer au serveur. De même, le type de données envoyées pouvant être configuré par un en-tête spécifique, nous pouvons maintenant envoyer autre chose que des données encodées en URL (couples de variable/valeur séparées par une esperluette) comme nous l'avons fait jusqu'à présent.

Nous allons donc consacrer ce nouvel atelier à la mise en œuvre d'un système qui permettra d'envoyer au serveur des données au format XML. Notre exemple reste très simple car nous n'envoyons que deux paramètres (le nom et le prénom du joueur) mais par la suite, vous pouvez aussi utiliser cette technique pour envoyer un grand nombre d'informations de la même manière.

Le premier intérêt d'utiliser des paramètres au format XML est de ne plus être limité par la taille des informations transmises au serveur, mais cette technique a aussi l'avantage de hiérarchiser les données envoyées et constitue donc une bonne solution pour intégrer des données complexes dans une requête Ajax.

Cette structure est composée :

- d'une page HTML (`index.html`) identique à celle de l'atelier 11-1 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est identique à celle de l'atelier 11-1 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 11-1 mais qui est modifiée pour gérer le document XML envoyé en paramètre de la requête `POST` ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 11-1 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement est semblable à celui de l'atelier précédent hormis le fait que les paramètres de la requête vont être envoyés au format XML.

Conception du système

Ouvrez la page HTML de l'atelier 11-1 précédent (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap11/atelier11-2/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Pour transformer notre requête POST et envoyer cette fois des données XML, nous allons devoir commencer par modifier le code de la fonction jouer() qui déclenche l'envoi de la requête.

Ouvrez le fichier fonctionsMachine.js et localisez la fonction jouer() dans le code de la page. Nous devons remplacer dans cette fonction l'affectation actuelle de la variable parametres (valeurs de nom et de prenom encodées en URL) par un document XML constitué d'un élément racine <joueur> et de deux éléments enfants <nom> et <prenom>. Dans ce premier atelier sur l'envoi de paramètres en XML, nous allons simplement construire le document XML par concaténation de balises en y intégrant les valeurs saisies dans les champs du formulaire comme contenu des éléments <nom> et <prenom> (voir code 11-3). Cependant, nous allons voir dans le prochain atelier une seconde technique qui consiste à utiliser les méthodes et propriétés du DOM pour créer un arbre XML puis à le sérialiser pour ensuite l'envoyer au serveur.

Code 11-3 : création du document XML qui va être envoyé en paramètre de la requête :

```
var parametresXml =
    "<joueur>" +
      "<nom>" + codeContenu("nom") + "</nom>" +
      "<prenom>" + codeContenu("prenom") + "</prenom>" +
    "</joueur>";
```

La méthode open() de l'objet XHR ne nécessite aucune modification car, comme dans l'atelier 11-1, nous allons envoyer une requête Ajax asynchrone, avec la méthode POST et au même fichier serveur.

```
objetXHR.open("post","gainAleatoire.php", true);
```

En revanche, le type de données envoyées change et il faut donc modifier l'en-tête nommé Content-Type, que nous avions ajouté dans l'atelier précédent, pour indiquer que le type du contenu envoyé avec la requête POST est cette fois du XML :

```
objetXHR.setRequestHeader("Content-Type","text/xml");
```

Comme le transfert est encore réalisé avec POST, les paramètres à envoyer, qu'ils soient encodés en URL ou au format XML, sont toujours insérés dans l'argument de la méthode send() grâce à la variable parametresXml initialisée précédemment ; la configuration de cette méthode reste donc la même :

```
objetXHR.send(parametresXml);//envoi de la requête
```

Code 11-4 : Code complet de la fonction jouer() après nos modifications (les modifications à apporter par rapport au code de l'atelier précédent sont indiquées en gras) :

```
function jouer() {
 objetXHR = creationXHR();
 var parametresXml =
  "<joueur>" +
   "<nom>" + codeContenu("nom") + "</nom>" +
   "<prenom>" + codeContenu("prenom") + "</prenom>" +
  "</joueur>";
 objetXHR.open("post","gainAleatoire.php", true);
 objetXHR.onreadystatechange = actualiserPage;
 objetXHR.setRequestHeader("Content-Type", "text/xml");
```

```
 //gestion du bouton et du chargeur
 document.getElementById("button").disabled= true;
 document.getElementById("charge").style.visibility="visible";
 //envoi de la requête
 objetXHR.send(parametresXml);
}
```

La modification du fichier `fonctionsMachine.js` est terminée, vous pouvez maintenant l'enregistrer et ouvrir le fichier serveur `gainAleatoire.php` pour l'adapter aux nouvelles données envoyées en XML par le moteur Ajax.

Dans ce fichier PHP, nous allons devoir traiter les données du document XML et il faut commencer avant tout par récupérer ses informations. Or, contrairement à l'envoi de données encodées en URL qui pouvaient être récupérées à l'aide de variables HTTP (comme `$_REQUEST`, `$_GET` ou `$_POST` selon la méthode employée), un document XML ne peut pas être formaté de la sorte. En effet, le format XML est complexe et ne peut pas être décomposé dans un tableau de variables HTTP comme le sont les simples données HTTP. Nous devons donc récupérer le document XML complet et sans mise en forme préalable. Pour cela, nous allons utiliser la fonction `file_get_contents()` appliquée au flux d'entrée `php://input` qui permet de lire des données `POST` bruts. De cette manière le document XML est enregistré dans la variable `$parametresXml` sans altérations :

```
$parametresXml = file_get_contents('php://input');
```

Pour traiter le document XML ainsi récupéré, nous allons utiliser les méthodes de l'objet `SimpleXML`. Mais avant cela, il faut commencer par créer un objet XML au moyen de l'instruction ci-dessous :

```
$objetSimpleXML=simplexml_load_string($parametresXml);
```

Une fois l'objet créé, il est facile d'accéder à ses propriétés correspondant aux contenus des éléments du document XML à l'aide des instructions suivantes :

```
$nomJoueur=$objetSimpleXML->nom;
$prenomJoueur=$objetSimpleXML->prenom;
```

Dès que les données sont isolées dans ces variables, il ne reste plus qu'à appliquer la même procédure de concaténation que dans l'atelier précédent pour mettre en forme le résultat avant de le retourner au navigateur.

```
$nom=$prenomJoueur." ".$nomJoueur;
$resultat=$nom.':'.$gain;
```

Après sa modification, le code du fichier `gainAleatoire.php` doit être semblable au code 11-5 ci-dessous.

Code 11- 5 : Instructions du fichier `gainAleatoire.php` :

```
//simulation du temps d'attente du serveur (2 secondes)
sleep(2);
//récupération des paramètres au format XML
$parametresXml = file_get_contents('php://input');
//création d'un objet Simple Xml à partir des paramètres récupérés
$objetSimpleXML=simplexml_load_string($parametresXml);
//récupération du nom du joueur
$nomJoueur=$objetSimpleXML->nom;
//récupération du prenom du joueur
$prenomJoueur=$objetSimpleXML->prenom;
//calcul du nouveau gain entre 0 et 100 Euros
```

```
$gain = rand(0,100);
//concaténation du nom
$nom=$prenomJoueur." ".$nomJoueur;
//mise en forme du résultat avec le nom
$resultat=$nom.':'.$gain;
//envoi de la réponse au navigateur
echo $resultat;
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Dans Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Saisissez vos nom et prénom dans les deux champs du formulaire puis cliquez sur le bouton JOUER. L'application doit alors transmettre le document XML dans lequel ont été intégrées les informations saisies et l'animation doit apparaître pour vous signaler que le traitement est en cours. Au terme du traitement, vos nom et prénom doivent s'afficher dans la fenêtre des résultats suivis du nouveau gain obtenu.

Pour observer les transferts de données entre le navigateur et le serveur, activez Firebug et cliquez sur l'onglet Console. Effacez les différents enregistrements précédents à l'aide de la fonction Clear du menu puis cliquez de nouveau sur le bouton JOUER. L'envoi de la requête `POST` au fichier PHP doit apparaître dans la fenêtre de la console Firebug, cliquez sur le + qui précède ce fichier pour le dérouler, puis sur l'onglet Post pour voir apparaître le document XML qui a été envoyé au serveur (voir figure 11-2). Vous pouvez ensuite visualiser les données renvoyées par le serveur en cliquant sur l'onglet Response mais elles sont semblables à celles que nous avons obtenues jusqu'à présent.

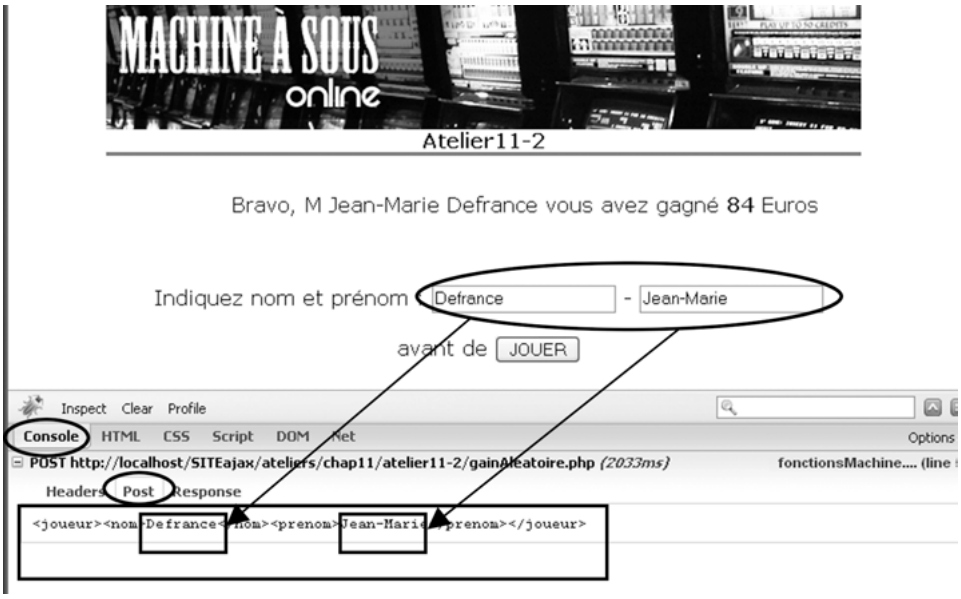


Figure 11-2

Test d'un transfert Asynchrone avec des paramètres XML

Atelier 11-3 : requête asynchrone POST avec paramètres issus d'un arbre DOM XML

Composition du système

Dans l'atelier précédent, nous avons mis en œuvre un système exploitant un document XML pour envoyer des informations au serveur lors de l'émission de la requête mais pour simplifier notre application, le document XML a été créé par une simple concaténation de balises et de leurs contenus.

En pratique, il est souvent utile de pouvoir récupérer les informations d'un arbre XML construit à l'aide de méthodes du DOM. Cela permet par exemple d'enregistrer les actions de l'utilisateur en reliant un événement particulier à un script d'ajout ou de modification d'un nœud de l'arbre XML.

Pour illustrer cette technique, nous vous proposons de mettre en œuvre une application permettant d'utiliser des boutons pour présélectionner le nom du joueur au lieu de devoir saisir ses nom et prénom dans des champs de formulaire comme dans l'atelier précédent.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la base est identique à celle de l'atelier 11-2 mais qui est modifiée afin d'ajouter des boutons pour la présélection des noms et prénoms du joueur ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est identique à celle de l'atelier 11-2 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier 11-2 ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 11-2 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

La page `index.html` contient cette fois deux boutons correspondant à des présélections de deux joueurs différents qui vont se substituer à la saisie des champs actuels du formulaire. Chaque bouton est lié à un gestionnaire d'événement `onclick` qui déclenche un programme de création d'un arbre DOM préconfiguré avec le nom et le prénom du joueur. Lorsque le joueur clique ensuite sur le bouton JOUER, l'arbre précédemment créé est sérialisé puis envoyé au serveur pour traitement. Le reste du système reste identique à celui de l'atelier précédent.

Conception du système

Ouvrez la page HTML de l'atelier 11-2 précédent (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap11/atelier11-3/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Commencez par remplacer les champs de saisie du nom et du prénom par les boutons de présélection. Pour cela, ouvrez la page `index.html` en mode Création dans Dreamweaver et supprimez les deux champs de saisie. À l'aide de l'onglet Formulaire de la barre Inser-

tion, ajoutez ensuite deux boutons dans la zone de formulaire (voir repère 1 de la figure 11-3). Personnalisez ces boutons (renseignez le champ Valeur du panneau des propriétés de chaque bouton, voir repère 3 de la figure 11-3) avec les noms et prénoms des deux joueurs de votre choix, puis attribuez à chaque bouton un nom différent (`nom1` et `nom2` par exemple, à saisir dans le champ du nom du bouton du panneau des Propriétés, voir repère 4 de la figure 11-3). Passez ensuite en mode Code et assurez-vous que les attributs `id` de chaque bouton ont bien été configurés avec `nom1` et `nom2` de sorte à pouvoir les référencer dans le gestionnaire d'événement que nous allons mettre en place par la suite. Les modifications de la page `index.html` sont maintenant terminées : enregistrez-la avant de passer à la suite des modifications.

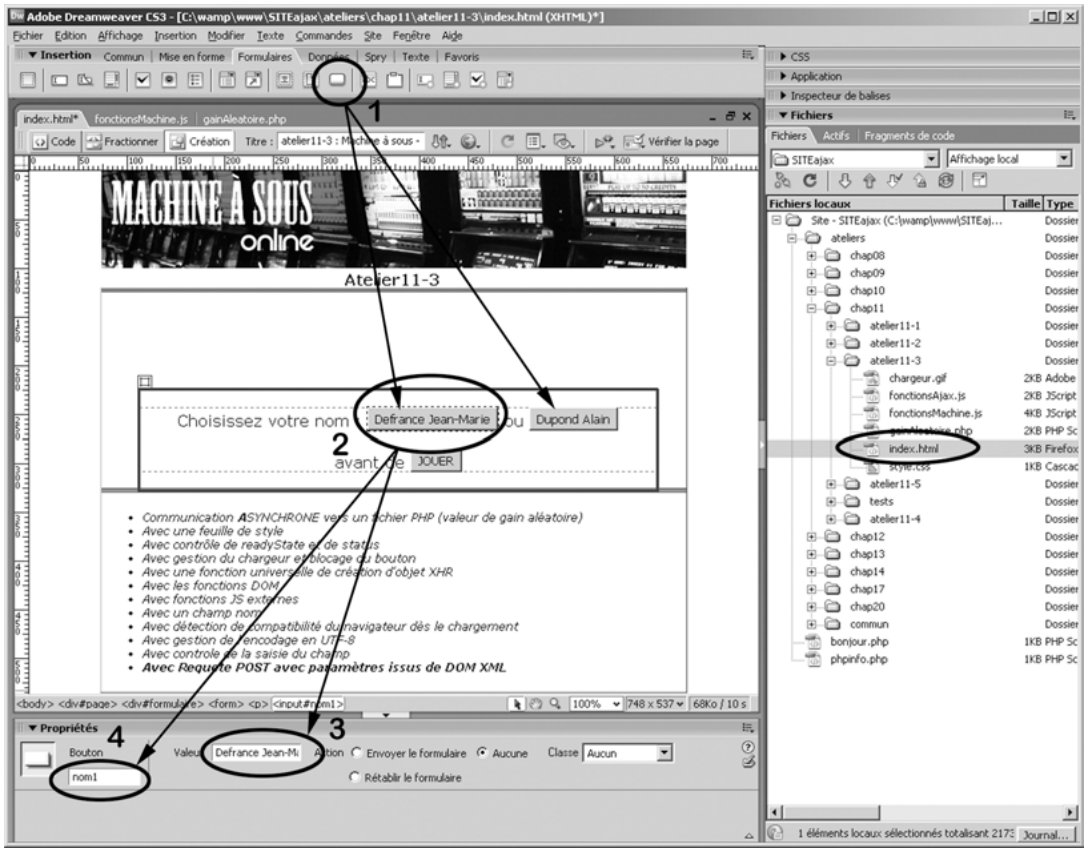


Figure 11-3

Modification de la page index.html : ajout et configuration des boutons de présélection

Maintenant que les boutons de présélection sont en place, nous allons passer à la configuration du gestionnaire d'événement. Pour cela nous allons ouvrir le fichier `fonctionMachine.js` et ajouter les deux déclarations de gestionnaire ci-dessous à la fin de la fonction `testerNavigateur()` afin d'être sûr que les boutons concernés seront déjà chargés au moment de leur configuration.

```
document.getElementById("nom1").onclick=function(){ajouteNom("Defrance","Jean-Marie")};
document.getElementById("nom2").onclick=function(){ajouteNom("Dupond","Alain")};
```

Ces déclarations associent aux deux boutons un événement `onclick` qui déclenche la même fonction `ajouteNom()` mais avec des paramètres différents. En effet, même si la syntaxe de l'attribution de ces fonctions anonymes aux gestionnaires directement dans le code JavaScript peut paraître un peu déstabilisante par rapport à celles que nous avons utilisées jusqu'alors, elle a l'énorme avantage de pouvoir passer des paramètres à la fonction associée à l'événement. Nous pouvons ainsi récupérer ces paramètres dans la fonction appelée comme si nous les avions envoyés depuis une déclaration de gestionnaire équivalente dans une page HTML semblable à celle du code ci-dessous :

```
<input id="nom1" onclick= ajouteNom("Defrance","Jean-Marie"); … >
```

La fonction déclenchée par une action sur l'un des deux boutons doit créer un arbre XML personnalisé avec le nom et prénom de chaque joueur. Pour cela nous allons utiliser les méthodes du DOM pour construire l'arbre branche par branche et créer successivement les éléments `<nom>` et `<prenom>` (avec `createElement()`) puis les nœuds texte (avec `createTextNode()`) contenant les noms et prénoms des deux joueurs que nous allons associer ensuite ensemble avec la méthode `appendChild()`. Enfin, les deux éléments sont ensuite rattachés en tant qu'enfants à un élément racine `<joueur>` en suivant la même procédure que pour les nœuds texte. À noter que l'élément `elementJoueur` est déclaré comme variable globale (sans le préfixe `var`) afin de pouvoir en disposer dans les autres fonctions du moteur Ajax.

Une fois terminée, la fonction de création des documents XML personnalisés à chacun des joueurs est semblable au code 11-6.

Code 11-6 : fonction `ajouteNom()` :

```
function ajouteNom(nom,prenom) {
  //création de l'élement nom avec son noeud texte
  var elementNom=document.createElement("nom");
  var texteNom=document.createTextNode(nom);
  elementNom.appendChild(texteNom);
  //création de l'élement Prénom avec son noeud texte
  var elementPrenom=document.createElement("prenom");
  var textePrenom=document.createTextNode(prenom);
  elementPrenom.appendChild(textePrenom);
  //création du noeud racine joueur
  elementJoueur=document.createElement("joueur");
  elementJoueur.appendChild(elementNom);
  elementJoueur.appendChild(elementPrenom);
 }
```

Nous disposons maintenant du mécanisme de création d'un arbre XML personnalisé mais il nous reste à mettre en place le système de sérialisation de cet arbre et son insertion en tant que paramètres dans la méthode `send()` du moteur Ajax. Pour cela, nous allons modifier la fonction `jouer()` qui déclenche l'envoi de la requête.

Pour sérialiser l'arbre XML `elementJoueur`, nous allons utiliser des méthodes compatibles avec les spécifications du W3C. Nous devons commencer par créer un objet de la classe `XMLSerializer` puis ensuite utiliser l'une de ses méthodes `serializeToString()` permettant de transformer notre arbre en un document XML équivalent mais constitué de chaînes de caractères. Le résultat de la sérialisation est ensuite affecté à la variable `parametresXml` qui est elle-même utilisée comme argument par la méthode `send()` lors de l'envoi des données au serveur.

```
var objetSerializer=new XMLSerializer();
var parametresXml = objetSerializer.serializeToString(elementJoueur);
```

Une fois terminée, la fonction `jouer()` est semblable au code 11-7 ci-dessous.

Code 11-7 : Fonction `jouer()` :

```
function jouer() {
  objetXHR = creationXHR();
  //construction des données des parametres en XML
  var objetSerializer=new XMLSerializer();
  var parametresXml = objetSerializer.serializeToString(elementJoueur);
  //Config. objet XHR
  objetXHR.open("post","gainAleatoire.php", true);
  objetXHR.onreadystatechange = actualiserPage;
  objetXHR.setRequestHeader("Content-Type","text/xml");
  //gestion du bouton et du chargeur
  document.getElementById("button").disabled= true;
  document.getElementById("charge").style.visibility="visible";
  objetXHR.send(parametresXml);//envoi de la requête
 }
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Dans Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Vous devez alors voir les deux boutons de présélection du joueur. Cliquez sur l'un d'entre eux pour choisir votre profil avant de cliquer sur le bouton JOUER. La requête doit alors être envoyée au serveur (l'animation doit apparaître à l'écran pendant le temps de traitement) et le résultat du jeu doit s'afficher au bout de deux secondes, de la même manière que dans l'atelier précédent.

Nous allons maintenant tester notre système avec le navigateur Internet Explorer. Utilisez pour cela la méthode de votre choix : bouton Aperçu de Dreamweaver, nouvelle fenêtre IE ou bouton IE Tab placé en bas de l'écran dans Firefox. Une fois l'application chargée dans IE (ou dans un IE émulé par Firefox), renouvelez la même procédure que précédemment pour tester le système.

On découvre alors que le système ne répond plus lorsqu'on appuie sur le bouton JOUER…

D'autre part, si vous testez maintenant le système sur un serveur distant ne disposant pas de la version 5 de PHP, nous pouvons constater cette fois que notre système ne fonctionne plus, aussi bien avec IE qu'avec Firefox…

Le prochain atelier sera consacré à la résolution de ces deux problèmes pour lesquels nous allons vous présenter des solutions alternatives.

Atelier 11-4 : requête asynchrone POST avec paramètres issus d'un arbre DOM XML multi-navigateurs et compatible PHP 4

Composition du système

Dans l'atelier précédent, nous avons remarqué que le système ne fonctionnait pas dans Wamp 5 avec le navigateur Internet Explorer, de même qu'il était impossible de porter notre application sur un serveur PHP 4 et cela quel que soit le navigateur utilisé.

Le premier problème est lié au fait que le navigateur IE n'est pas conforme au W3C et qu'il ne peut pas instancier la classe `XMLSerializer` comme Firefox le fait. Nous allons donc modifier la procédure de sérialisation de manière à la rendre compatible avec les deux navigateurs.

Le second problème vient de l'utilisation des méthodes de simpleXML qui sont disponibles uniquement depuis la version 5 de PHP. Pour faire fonctionner notre application en PHP 4, nous verrons dans cet atelier comment intégrer un analyseur syntaxique XML en solution alternative à la classe simpleXML.

Cette structure est composée :

- d'une page HTML (`index.html`) identique à celle de l'atelier 11-3 ;
- d'un fichier `JS` (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier `JS` (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant son adaptation à IE est identique à celle de l'atelier 11-3 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 11-3 mais qui est modifiée pour assurer la compatibilité avec le PHP 4 ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 11-3 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système est identique à celui de l'atelier précédent hormis le fait que l'application est compatible avec tous les navigateurs et qu'elle est portable sur un serveur PHP 4.

Conception du système

Ouvrez la page HTML de l'atelier 11-3 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap11/atelier11-4/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Pour résoudre le problème de compatibilité du processus de sérialisation avec IE, nous pourrions mettre en œuvre un détecteur de navigateur et appliquer le processus de sérialisation adapté selon le type de navigateur (IE ou les compatibles W3C comme Firefox). En effet, si IE n'interprète pas la classe `XMLSerializer`, il permet quand même de sérialiser un arbre XML en utilisant la propriété `.xml` appliquée à l'un de ses éléments et nous pourrions utiliser cette technique en solution alternative si le navigateur IE était détecté.

Une autre solution consiste à utiliser une bibliothèque adaptée qui permettrait de sérialiser le document XML sans se préoccuper du type de navigateur. Dans le cadre de cet atelier, nous allons opter pour cette seconde solution et utiliser la bibliothèque zXML.

Bibliothèque zXML

Auteur : Nicholas C. Zakas (Free Software GNU)

Vous pouvez télécharger la bibliothèque zXML à l'adresse ci-dessous :

http://www.nczonline.net/downloads/

Pour mettre en place ce système, commencez par télécharger le fichier `zxml.js` de la bibliothèque zXML et placez-le dans le répertoire `/chap11/atelier11-4/` de notre atelier. Pour disposer de cette bibliothèque dans notre application nous allons ensuite ajouter une balise `<script>` faisant référence à ce fichier dans la balise `<head>` de la page `index.html`.

```
<script type="text/javascript" src="zxml.js" ></script>
```

La bibliothèque zXML permet de disposer sous IE d'une classe identique (ou presque) à celle de Firefox. La seule différence réside dans son nom qui est précédé d'un « z » soit `zXMLSerializer` au lieu de `XMLSerializer` précédemment utilisé avec Firefox. Il convient donc de modifier le nom de cette classe lors de l'instanciation de l'objet `objetSerializer` dans la fonction `jouer()` du fichier `fonctionsMachine.js` en lui ajoutant un « z » comme dans le code ci-dessous.

```
var objetSerializer=new zXMLSerializer();
var parametresXml = objetSerializer.serializeToString(elementJoueur);
```

À noter que la bibliothèque zXML permet aussi de disposer dans Firefox de la propriété `.xml` propre à Internet Explorer. Nous aurions donc pu aussi remplacer les deux lignes de code précédentes par l'instruction ci-dessous : le fonctionnement du système aurait été identique.

```
var parametresXml=elementJoueur.xml;
```

Les modifications nécessaires pour assurer la compatibilité du système pour tous les navigateurs sont terminées. Vous pouvez enregistrer vos fichiers dès maintenant et passer à la phase de test sous IE si vous le désirez.

Dans la partie suivante, nous allons présenter une alternative à l'utilisation de SimpleXML. Au terme des modifications, vous pouvez toujours faire fonctionner votre application dans Wamp5 mais l'intérêt de cette modification est évidemment de la tester sur un serveur PHP 4.

La principe de l'analyseur syntaxique consiste à créer un objet d'analyse XML (à l'aide de `xml_parser_create()`) puis à lui associer trois fonctions d'analyse. La première analyse toutes les balises ouvrantes des éléments du document XML, la seconde, les balises fermantes et enfin la troisième les contenus des éléments. Si dans la première fonction, nous enregistrons dans une variable globale le nom de la balise en cours d'analyse, il est ensuite possible de récupérer cette même variable dans la troisième fonction ce qui nous permet de traiter le contenu selon le nom de la balise concernée. Si on applique cette technique à notre système, nous pouvons ainsi récupérer les informations du joueur contenu dans les balises `<nom>` et `<prenom>` du document XML envoyé par le navigateur.

Pour la mise en œuvre de cet analyseur, commencez par saisir les trois fonctions d'analyse qui sont associées à l'objet analyseur (voir code 11-8 et le repère 1 de la figure 11-4).

Code 11-8 : Les trois fonctions d'analyse :

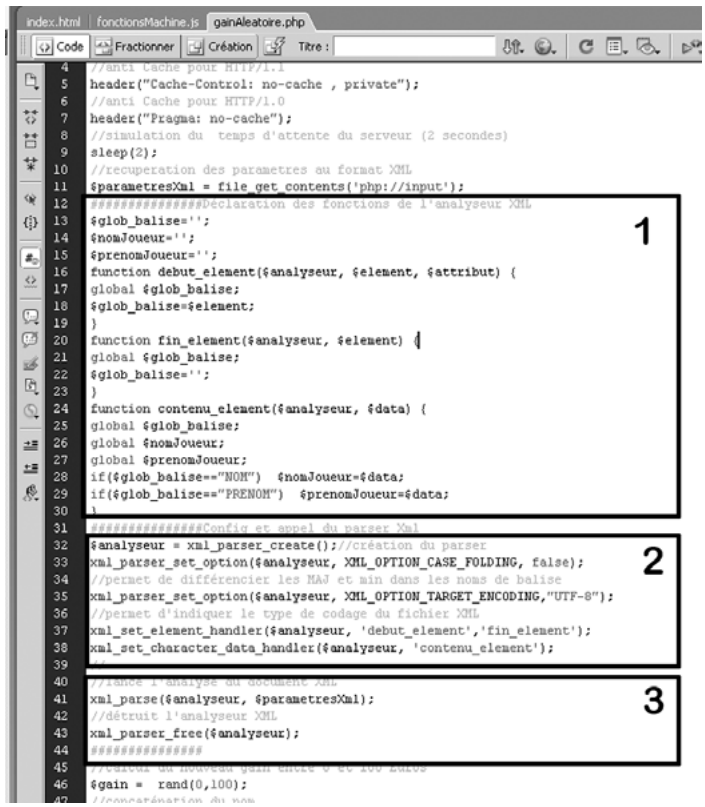
```
$glob_balise='';
$nomJoueur='';
$prenomJoueur='';
function debut_element($analyseur, $element, $attribut) {
  global $glob_balise;
  $glob_balise=$element;
}
```

```
function fin_element($analyseur, $element) {
  global $glob_balise;
  $glob_balise='';
}

function contenu_element($analyseur, $data) {
  global $glob_balise;
  global $nomJoueur;
  global $prenomJoueur;
  if($glob_balise=="NOM") $nomJoueur=$data;
  if($glob_balise=="PRENOM") $prenomJoueur=$data;
}
```

Figure 11-4

Modification de la page gainAleatoire.php : ajout d'un analyseur syntaxique XML compatible PHP 4



Puis créez et configurez l'objet analyseur afin de le lier aux trois fonctions précédentes (voir code 11-9 et le repère 2 de la figure 11-4).

Code 11-9 : Création et configuration de l'analyseur :

```
//création du parser
$analyseur = xml_parser_create();
//permet de différencier les MAJ et min dans les noms de balise
xml_parser_set_option($analyseur, XML_OPTION_CASE_FOLDING, false);
//permet d'indiquer le type de codage du fichier XML
xml_parser_set_option($analyseur, XML_OPTION_TARGET_ENCODING,"UTF-8");
//permet de déclarer les noms des 3 fonctions d'analyse de l'objet
xml_set_element_handler($analyseur, 'debut_element','fin_element');
xml_set_character_data_handler($analyseur, 'contenu_element');
```

Enfin, l'analyse proprement dite est lancée sur le document XML placé dans la variable `$parametresXml` grâce à la méthode `xml_parse()` de l'objet. Une dernière méthode (`xml_parser_free()`) pourra être appelée ensuite au terme du traitement afin de supprimer l'analyseur en cours (voir code 11-10 et le repère 3 de la figure 11-4).

Code 11-10 : Exécution de l'analyse :

```
xml_parse($analyseur, $parametresXml);
xml_parser_free($analyseur);
```

Une fois les modifications effectuées, vous pouvez enregistrer le fichier PHP et passer aux tests dans le navigateur de votre choix et sur un serveur PHP 4 ou 5.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 de Dreamweaver afin de vous assurer que votre application fonctionne toujours avec les navigateurs compatibles W3C. Puis basculez dans un navigateur IE par la méthode de votre choix. Le système devrait fonctionner cette fois de la même manière qu'avec Firefox.

En ce qui concerne la portabilité du fichier PHP, le fait qu'il fonctionne sous PHP 5 atteste déjà qu'il n'y a pas d'erreur de code et il y a de forte chance qu'il fonctionne de la même manière sur un serveur PHP 4. Cependant, si vous le désirez, vous pouvez mettre en place votre application sur un serveur distant PHP 4 pour le tester en situation réelle. Il faudra toutefois s'assurer que l'extension `xml` de PHP est activée sur le serveur pour que l'analyseur puisse fonctionner. Pour vérifier l'activation de cette extension, il suffit de placer un fichier `phpinfo` sur votre serveur distant, de l'afficher depuis un navigateur et de lancer ensuite une recherche dans la page avec le mot-clé « xml ». Vous pourrez ainsi facilement localiser la partie de la page qui affiche l'état de l'extension `xml` et vérifier qu'elle bien activée.

Atelier 11-5 : requête asynchrone POST avec paramètres JSON

Composition du système

Pour clôturer les ateliers de ce chapitre sur les différentes requêtes `POST`, nous vous proposons de réaliser une requête Ajax avec envoi de paramètres au format JSON.

Dans les ateliers précédents, nous avons présenté le format XML comme le format permettant d'envoyer des données sans limitation de taille et disposant d'un hiérarchisation de leur contenu. Toutefois, le format XML souffre aussi d'une lourdeur syntaxique liée au grand nombre de balises nécessaires pour encadrer les données.

Il existe cependant, un nouveau format directement issu de la syntaxe des objets JavaScript qui se place actuellement comme une solution alternative à l'utilisation du XML pour faire des transferts de données entre le client et le serveur. Il s'agit du format JSON (*JavaScript Objet Notation*). En fait, JSON est la notation objet de JavaScript dont la syntaxe permet de représenter des objets et leurs propriétés mais aussi de simples tableaux indicés. Ces types d'objets étant présents dans la grande majorité des langages, il est alors très facile de mettre en place des fonctions de codage et de décodage des données spécifiques à chaque langage, faisant ainsi de JSON un format de transfert universel et léger.

Dans cet atelier, nous allons appliquer le langage JSON pour la première fois dans le transfert des paramètres émis du client vers le serveur lors de l'envoi d'une requête Ajax.

Dans ce contexte, il faut mettre en place des systèmes d'encodage côté client et de décodage côté serveur pour restituer les données afin de les traiter ensuite en PHP.

Même si nous pouvions réaliser ces systèmes directement dans le script, il est beaucoup plus judicieux de faire appel à des bibliothèques externes qui assurent ces fonctions d'encodage et de décodage automatiquement tout en préservant la conformité avec les technologies utilisées (JavaScript côté client et PHP côté serveur par exemple).

Cette structure est composée :

- d'une page HTML (`index.html`) identique à celle de l'atelier 11-4 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant sa modification est identique à celle de l'atelier 11-4 ;
- d'une bibliothèque externe JSON (`json.js`) que nous pouvons télécharger sur le site de l'éditeur `json.org` ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 11-4 mais qui est modifié pour assurer le décodage des données JSON réceptionnées ;
- d'une bibliothèque externe JSON-PHP (`JSON.php`) que nous pouvons télécharger sur le site de l'éditeur ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 11-4 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système est identique à celui de l'atelier précédent hormis le fait que l'envoi et la réception des paramètres de la requête Ajax sont réalisés au format JSON.

Conception du système

Ouvrez la page HTML de l'atelier 11-4 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap11/atelier11-5/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Nous allons commencer par mettre en œuvre le système d'encodage côté client. Pour cela, téléchargez sur votre ordinateur le fichier de la bibliothèque JSON.

Bibliothèque JSON

Vous pouvez télécharger la bibliothèque JavaScript JSON à l'adresse ci-dessous :

http://www.json.org/json.js

Copiez ensuite le fichier `json.js` dans le répertoire de l'atelier actuel `/chap11/atelier11-5/` puis ouvrez le fichier `index.html` avec Dreamweaver et ajoutez une balise `<script>` faisant référence au fichier de la bibliothèque dans la partie `<head>` de la page.

```
<script type="text/javascript" src="json.js"></script>
```

Enregistrez votre page puis passez au fichier `fonctionsMachine.js` pour localiser la fonction `jouer()` dans laquelle nous allons intervenir pour intégrer notre encodeur JSON. Pour coder les paramètres en JSON, il faut commencer par créer un objet JavaScript afin de pouvoir y ajouter les données que nous allons transférer sur le serveur.

```
var objetJS = new Object();
```

Pour ajouter une propriété à un objet, il suffit d'appliquer le nom de la propriété désirée à l'objet en syntaxe pointée et de lui affecter ensuite la valeur correspondante. Dans notre cas, nous allons créer deux propriétés `nom` et `prenom` auxquelles nous allons affecter les valeurs de leur champ de saisie respectif à l'aide de la fonction `codeContenu()` que nous avons déjà développé dans un atelier précédent (pour mémoire cette fonction prend comme argument l'identifiant de l'élément concerné et retourne son contenu après lui avoir appliqué un codage UTF 8).

```
objetJS.nom=codeContenu("nom");
objetJS.prenom=codeContenu("prenom");
```

Une fois l'objet et ses propriétés créés, nous pouvons passer à l'encodage en JSON en appliquant à l'objet la méthode `toJSONString()` de notre bibliothèque externe. Le résultat de l'encodage est alors affecté à la variable `parametres` qui est ensuite insérée en paramètre de la méthode `send()` lors de l'envoi de la requête.

```
var parametres =objetJS.toJSONString();
```

Une fois modifiée, la fonction `jouer()` doit être semblable au code 11-11 ci-dessous.

Code 11-11 : Fonction `jouer()` :

```
function jouer() {
 objetXHR = creationXHR();
 var objetJS = new Object();
 objetJS.nom=codeContenu("nom");
 objetJS.prenom=codeContenu("prenom");
 var parametres =objetJS.toJSONString();
 objetXHR.open("post","gainAleatoire.php", true);
 objetXHR.onreadystatechange = actualiserPage;
 objetXHR.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
 //gestion du bouton et du chargeur
 document.getElementById("button").disabled= true;
 document.getElementById("charge").style.visibility="visible";
 objetXHR.send(parametres);//envoi de la requête
}
```

Enregistrons le fichier `fonctionMachine.js` ainsi modifié et passons maintenant au fichier PHP qui doit réceptionner ces données au format JSON.

Bibliothèque JSON-PHP

Auteur : Michal Migurski

Vous pouvez télécharger la bibliothèque JSON-PHP à l'adresse ci-dessous :

http://pear.php.net/pepr/pepr-proposal-show.php?id=198

Le site de l'auteur de cette bibliothèque se trouve à l'adresse ci-dessous :

http://mike.teczno.com/json.html

Ici aussi, nous allons faire appel à une bibliothèque afin de se décharger de la tâche du décodage JSON. Il existe de nombreuses bibliothèques qui proposent ce genre de

fonctionnalités mais dans cet ouvrage nous allons utiliser la bibliothèque JSON-PHP qui a l'avantage d'être très simple à installer.

Téléchargez le kit de la bibliothèque JSON-PHP et copiez ensuite le fichier `JSON.php` dans le répertoire de l'atelier en cours. Ouvrez maintenant le fichier `gainAleatoire.php` et effacez toutes les instructions situées après la récupération du flux d'entrée dans la variable `$parametres` jusqu'à l'instruction du calcul du gain. Saisissez ensuite l'instruction ci-dessous de sorte à pouvoir disposer de la bibliothèque JSON-PHP dans notre fichier.

```
require_once('JSON.php');
```

Pour décoder les données au format JSON, nous devons commencer par créer un objet JSON à l'aide de la classe `Services_JSON`.

```
$objetJSON = new Services_JSON();
```

Une fois l'objet créé, il ne reste plus qu'à lui appliquer sa méthode `decode()` afin de récupérer un autre objet PHP `$objetJoueur` semblable à celui initialement créé avec JavaScript avant l'envoi de la requête et récupéré dans la variable `$paramètres JSON`.

```
$objetJoueur = $objetJSON->decode($parametresJSON);
```

Maintenant que nous disposons d'un objet PHP comportant la même structure et les mêmes données que l'objet JS précédemment créé côté client, il est très facile de récupérer les informations initiales grâce à ses propriétés en ajoutant à l'objet un accesseur (le symbole `->`) suivi du nom des propriétés comme dans le code ci-dessous.

```
$nomJoueur=$objetJoueur->nom;
$prenomJoueur=$objetJoueur->prenom;
```

Les deux paramètres nécessaires au traitement étant maintenant récupérés, le reste du programme est semblable aux ateliers précédents (voir code 11-12).

Code 11-12 : Instructions du fichier `gainAleatoire.php` :

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
sleep(2);
$parametresJSON = file_get_contents('php://input');
require_once('JSON.php');
$objetJSON = new Services_JSON();
$objetJoueur = $objetJSON->decode($parametresJSON);
$nomJoueur=$objetJoueur->nom;
$prenomJoueur=$objetJoueur->prenom;
$gain = rand(0,100);
$nom=$prenomJoueur." ".$nomJoueur;
$resultat=$nom.':'.$gain;
echo $resultat ;
```

Nous venons d'utiliser la bibliothèque externe JSON-PHP pour récupérer les données au format JSON côté serveur, mais il existe désormais une extension `json` intégrée à PHP et disponible pour les versions de PHP ultérieures à 5.2. Pour exploiter cette nouvelle extension, il suffit alors d'utiliser les méthodes `json_decode()` ou `json_encode()`. Avec cette extension, le code du système de décodage est évidemment beaucoup plus simple et ne nécessite plus de disposer de la bibliothèque JSON-PHP. Pour illustrer son utilisation, nous vous proposons d'appliquer cette seconde technique à l'exemple de notre atelier. Le nouveau contenu du fichier `gainAleatoire.php` serait alors semblable au code 11-13.

Code 11-13 : Instructions du fichier `gainAleatoire.php` pour PHP 5.2 ou + :

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
sleep(2);
$parametresJSON = file_get_contents('php://input');
$objetJoueur=json_decode($parametresJSON);
$nomJoueur=$objetJoueur->nom;
$prenomJoueur=$objetJoueur->prenom;
$gain = rand(0,100);
$nom=$prenomJoueur." ".$nomJoueur;
$resultat=$nom.':'.$gain;
echo $resultat ;
```

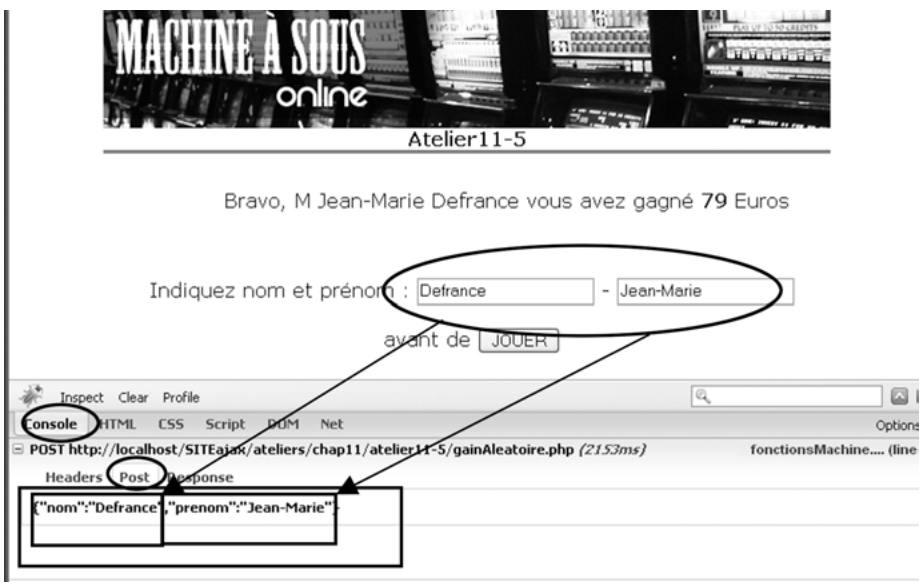
Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 de Dreamweaver. Saisissez votre nom et prénom dans les deux champs du formulaire puis cliquez sur le bouton JOUER. L'application doit alors transmettre les paramètres JSON au serveur et le chargeur doit apparaître pour vous signaler que le traitement est en cours. Au terme du traitement, vos nom et prénom doivent s'afficher dans la fenêtre des résultats suivis du nouveau gain obtenu comme dans l'atelier précédent.

Pour observer les transferts de données entre le navigateur et le serveur, activez Firebug et cliquez sur l'onglet Console. Effacez les différents enregistrements précédents à l'aide de la fonction Clear du menu puis cliquez de nouveau sur le bouton JOUER. L'envoi de la requête `POST` au fichier PHP doit apparaître dans la fenêtre de Firebug, cliquez sur le petit « + » qui précède ce fichier pour le dérouler puis sur l'onglet Post pour voir apparaître les paramètres au format JSON qui ont été envoyés au serveur (voir figure 11-5). Vous pouvez ensuite visualiser les données renvoyées par le serveur en cliquant sur l'onglet Response mais elles sont semblables à celles que nous avons eues jusqu'à présent.

Figure 11-5

Test de la requête Ajax avec paramètres au format JSON



12

Applications Ajax-PHP avec réponses HTML, XML, JSON et RSS

Dans les différents ateliers que nous avons réalisés jusqu'à présent, nous avons toujours utilisé le format texte pour récupérer les réponses du serveur. Cependant, d'autres formats de données peuvent être utilisés pour effectuer ces transferts afin de récupérer des données préformatées (avec des fragments de HTML par exemple) mais surtout plus volumineuses et hiérarchisées (avec JSON, XML ou encore RSS).

Aussi, nous vous proposons dans ce chapitre d'étudier, sur la base de la même application de la machine à sous, les principaux formats que vous pourrez utiliser dans vos futures applications Ajax-PHP. L'objectif de ces ateliers n'étant pas d'ajouter des fonctionnalités supplémentaires à l'application mais au contraire de comparer sur une application identique ce qui change entre ces techniques pour effectuer le transfert de la réponse du serveur dans des formats différents.

Atelier 12-1 : requête avec réponse en HTML

Composition du système

Nous allons commencer par étudier le transfert d'une réponse au format HTML. Celui-ci peut être intéressant lorsqu'on veut remplacer directement toute une zone de la page Web par une nouvelle structure HTML différente.

Toutefois, ce n'est pas le cas de ce premier exemple, car nous allons nous contenter de remplacer le conteneur `info` par un fragment HTML identique mais dont les résultats (nom et gain) y ont été préalablement insérés. Ainsi, si l'envoi de la requête ne change pas de celui utilisé pour l'atelier 10-4 (requête `GET` avec paramètres), le fichier PHP renvoie cette fois un fragment HTML (identique à celui de la balise `<div>` d'identifiant `info`) dans lequel ont déjà pris place les valeurs des résultats (le nom et le gain). Une fois réceptionné par le moteur Ajax, celui-ci remplace le fragment HTML actuel de la page

Web par celui réceptionné dans la réponse du serveur pour que les nouveaux résultats soient visibles à l'écran de la même manière que dans l'atelier précédent.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure est identique à celle de l'atelier 10-4 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est identique à celle de l'atelier 10-4 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 10-4 mais qui va être modifiée pour gérer la réponse HTML ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 10-4 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système est identique à celui de l'atelier 10-4 que nous allons prendre comme base de départ pour effectuer nos modifications hormis le fait que, cette fois, la réponse du serveur est constituée d'un fragment HTML.

Conception du système

Ouvrez la page HTML de l'atelier 10-4 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap12/atelier12-1/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Pour gérer le transfert de la réponse en HTML, nous allons commencer par modifier le code du fichier PHP `gainAleatoire.php` afin qu'il renvoie un fragment de HTML et non plus les données texte séparées par le symbole « : » comme dans l'atelier 10-4. Pour cela, ouvrez le fichier PHP et supprimez (ou commentez) l'avant dernière instruction qui concatène les deux données résultats avec le symbole « : » puis remplacez-la par le code ci-dessous :

```
$resultat='Bravo, M <span id="gagnant">'.$nom.'</span> vous avez gagné
➥<span id="resultat">'.$gain.'</span> Euros';
```

Ainsi, ce fragment de code HTML enregistré dans la variable `$resultat` est directement renvoyé comme résultat au navigateur. Les deux informations `$nom` et `$gain` ayant déjà pris place dans les balises `<span>` correspondantes. Le résultat étant maintenant du HTML, il convient de modifier l'en-tête `Content-type` pour lui affecter la valeur `text/html` au lieu de `text/plain` (voir code 12-1).

Code 12-1 : fichier `gainAleatoire.php` après modification :

```
//indique que le type de la réponse sera du HTML
header("Content-Type: text/html ; charset=utf-8");
//anti cache pour HTTP/1.1
header("Cache-Control: no-cache , private");
```

```
//anti cache pour HTTP/1.0
header("Pragma: no-cache");
//simulation du temps d'attente du serveur (2 secondes)
sleep(2);
//récupération du paramètre nom
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
//calcul du nouveau gain entre 0 et 100 euros
$gain = rand(0,100);
//mise en forme HTML du résultat
$resultat='Bravo, M <span id="gagnant">'.$nom.'</span> vous avez gagné
➥<span id="resultat">'.$gain.'</span> Euros';
//envoi de la réponse au navigateur
echo $resultat;
```

Passez maintenant coté client et ouvrez le fichier `fonctionsMachine.js`, puis localisez la fonction `actualiserPage()`. Comme cette fonction récupère l'attribut `responseText` de l'objet XHR contenant la réponse du serveur, nous allons devoir la modifier pour tenir compte du fait que la réponse est désormais un fragment de code HTML et non plus les deux données (`nom` et `gain`) séparées par le caractère « : ».

La fonction `split()` n'étant plus nécessaire, la récupération du résultat est donc réduite à une simple affectation de la variable `nouveauResultat` par l'attribut `responseText` de l'objet XHR.

```
var nouveauResultat = objetXHR.responseText;
```

Pour remplacer le fragment de code HTML de l'élément `info` par le résultat précédemment récupéré, nous n'allons pas utiliser ici notre fonction `remplacerContenu()` que nous avions développée avec des méthodes du DOM dans un atelier précédent. En effet, celle-ci est prévue pour gérer exclusivement le texte d'un élément et non un fragment de code HTML pour lequel ses balises, dans ce cas, ne seraient plus interprétées par le navigateur et s'afficheraient comme un simple texte à l'écran.

En solution alternative pour réaliser le remplacement du fragment HTML (ce ne sera pas toujours le cas), nous allons faire appel à l'attribut `innerHTML` qui, lui, n'a pas cet inconvénient. Pour appliquer cet attribut à l'élément concerné, il faut donc au préalable le référencer avec la méthode `getElementById()`. Il suffit ensuite d'affecter à l'attribut `innerHTML` de l'élément, le fragment HTML mémorisé dans la variable `nouveauResultat` pour que le remplacement soit effectué.

```
var elementInfo = document.getElementById("info");
elementInfo.innerHTML=nouveauResultat;
```

Une fois modifié, le code de la fonction `actualiserPage()` doit être semblable au code 12-2 ci-dessous.

Code 12-2 : partie de la fonction `actualiserPage()` après modification :

```
function actualiserPage() {
if (objetXHR.readyState == 4) {//test si le résultat est disponible
  if (objetXHR.status == 200) {
   var nouveauResultat = objetXHR.responseText;//récup du résulat
   var elementInfo = document.getElementById("info");
   elementInfo.innerHTML=nouveauResultat;
```

```
    //affiche la zone info
    document.getElementById("info").style.visibility="visible";
    //gestion du bouton et du chargeur
    document.getElementById("button").disabled= false;
    document.getElementById("charge").style.visibility="hidden";
   }
  }
 }
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Sous Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Le fonctionnement du nouveau système doit être exactement le même que celui de l'atelier 10-4, hormis le fait que le transfert des résultats est effectué cette fois à l'aide d'un fragment HTML.

Pour visualiser les données renvoyées par le serveur lors de la réponse, nous vous conseillons d'activer Firebug et de cliquer sur l'onglet Console. Commencez par cliquer sur le bouton Clear, de sorte à effacer les éventuels enregistrements antérieurs. Saisissez ensuite votre nom et cliquez sur bouton JOUER. Une nouvelle requête envoyée à `gainAleatoire.php` doit alors apparaître dans la fenêtre de l'onglet Console. Cliquez sur le + qui précède le nom du fichier pour le déplier puis sur l'onglet Response. Vous devriez voir le fragment HTML renvoyé par le serveur (voir figure 12-1) qui remplacera toute la zone de résultat du système (balise `<div>` d'identifiant `info`).

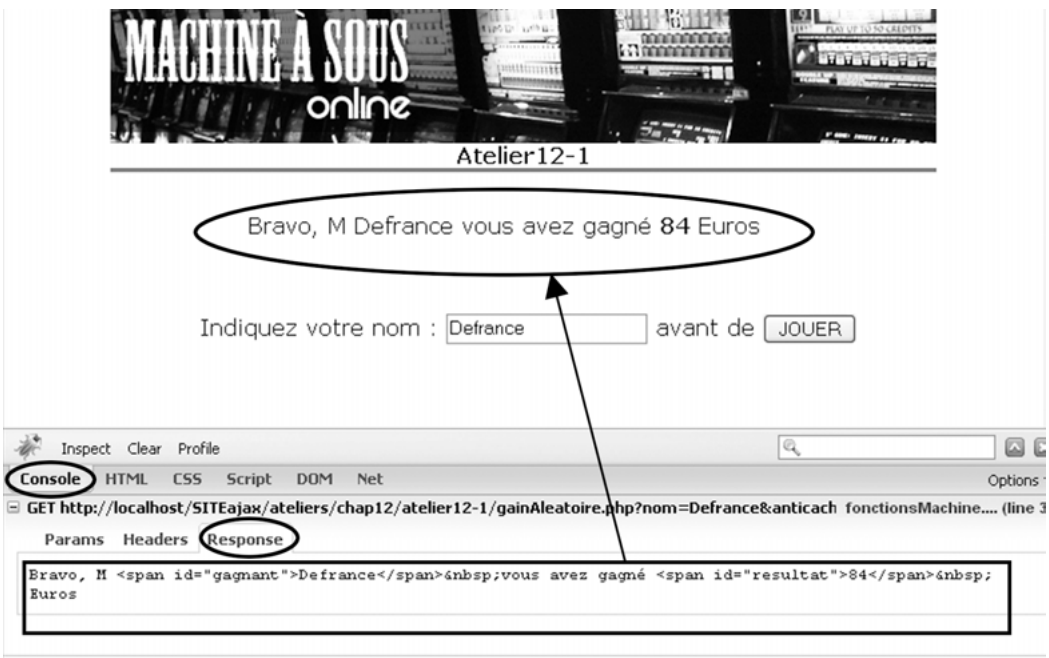


Figure 12-1

Test du transfert d'une réponse HTML

Atelier 12-2 : requête avec réponse en XML

Composition du système

Nous allons maintenant nous intéresser au format XML pour effectuer le transfert de la réponse du serveur. Hormis le fait que le nombre d'informations renvoyées pourra maintenant être beaucoup plus important, le format XML permet aussi de hiérarchiser les données, ce qui se révèle souvent bien pratique pour récupérer des résultats complexes ou issus d'une base de données par exemple.

Dans cet atelier, nous ne renvoyons que les valeurs du gain et du nom du gagnant comme dans les ateliers précédents, mais il vous sera très facile par la suite d'adapter cette technique à des transferts de données plus importants.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure est identique à celle de l'atelier 12-1 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est identique à celle de l'atelier 12-1 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 12-1 mais qui va être modifié pour gérer la réponse XML ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 12-1 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système est strictement identique à celui de l'atelier 12-1 que nous venons de réaliser sauf que dans cet atelier le transfert de la réponse est au format XML.

Conception du système

Ouvrez la page HTML de l'atelier 12-1 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap12/atelier12-2/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Pour adapter le système au transfert d'une réponse en XML, nous allons de nouveau modifier le code du fichier PHP `gainAleatoire.php` afin qu'il renvoie cette fois un document XML. Pour cela, ouvrez le fichier PHP et remplacez l'avant-dernière instruction par le code ci-dessous :

```
$resultat='<?xml version="1.0" encoding="utf-8"?>';
$resultat.= "<resultats><nom>".$nom. "</nom><gain>".$gain. "</gain></resultats>";
```

Ces lignes de code permettent de construire le document XML résultat constitué du prologue XML habituel suivi de la structure hiérarchique des balises dans lesquelles ont été insérées les nouvelles valeurs du nom et du gain. Nous disposons ainsi dans la variable `$resultat` d'un document XML bien formé (revoir si besoin les ressources sur le XML dans la partie 4 de cet ouvrage pour savoir ce qui caractérise un document bien

formé) dans lequel nous avons un nœud racine `<resultats>` et deux autres nœuds enfants `<nom>` et `<gain>`.

Enfin, le résultat étant maintenant au format XML, il faut donc de nouveau modifier l'entête `Content-type` pour lui affecter cette fois la valeur `text/xml`.

Code 12-3 : fichier `gainAleatoire.php` après modification :

```
//indique que le type de la réponse sera du HTML
header("Content-Type: text/xml ; charset=utf-8");
//anti cache pour HTTP/1.1
header("Cache-Control: no-cache , private");
//anti cache pour HTTP/1.0
header("Pragma: no-cache");
//simulation du temps d'attente du serveur (2 secondes)
sleep(2);
//récupération du paramètre nom
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
//calcul du nouveau gain entre 0 et 100 euros
$gain = rand(0,100);
//mise en forme XML du résultat
$resultat='<?xml version="1.0" encoding="utf-8"?>';
$resultat.= "<resultats><nom>".$nom. "</nom><gain>".$gain. "</gain></resultats>";
//envoi de la réponse au navigateur
echo $resultat;
```

Pour vous assurer que le fichier PHP est correctement configuré et que le document XML qui va être renvoyé au navigateur est bien formé, nous vous conseillons dans un premier temps de tester individuellement le fichier `gainAleatoire.php` sur le Web local (ouvrez le fichier PHP dans Dreamweaver et appuyez sur la touche F12). Une fois appelée dans le navigateur, la page doit afficher l'arbre du document XML (s'il est bien formé) avec la valeur « inconnu » dans la balise `<nom>` car aucune requête `GET` ne lui précise encore le nom du joueur. Vous pouvez alors simuler une requête client en ajoutant `?nom=Defrance` par exemple à la suite du nom de la page dans la barre d'adresse (voir figure 12-2). Le nom du joueur doit alors remplacer la valeur « inconnu » dans la balise `<nom>`.

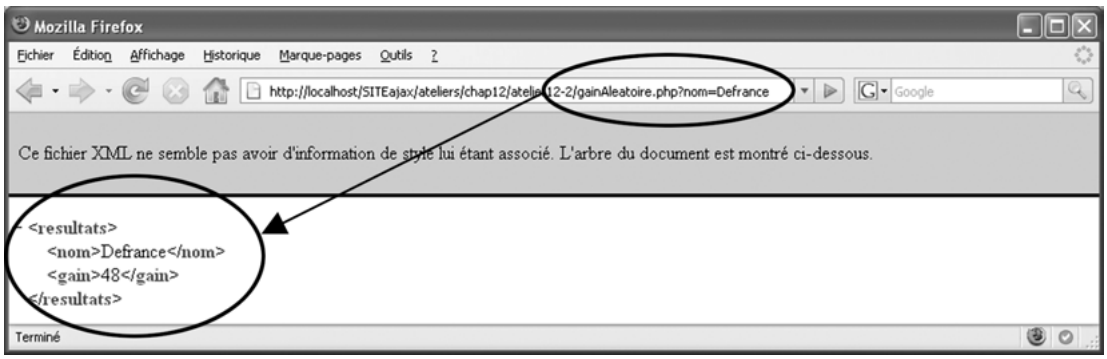


Figure 12-2

Test du fichier PHP vérifiant que le document XML renvoyé au navigateur est bien formé.

Il faut maintenant modifier le fichier `fonctionsMachine.js` afin que le moteur Ajax réceptionne comme il se doit ce document XML. Dans la fonction `actualiserPage()` nous allons commencer par modifier l'attribut de l'objet XHR utilisé pour récupérer la réponse. En effet, il ne s'agit plus maintenant de données texte, comme pour tous les transferts des réponses que nous avons réalisés jusqu'à présent, mais de données XML. Si nous désirons utiliser les méthodes du DOM XML, l'attribut `responseXML` de l'objet XHR doit alors remplacer l'attribut `responseText`. Les données récupérées par l'attribut `responseXML` sont alors transférées dans un arbre XML et peuvent ainsi être manipulées directement par les méthodes DOM spécifiques à la gestion des documents XML.

```
var nouveauResultatXML = objetXHR.responseXML;
```

Avant de récupérer les valeurs du gain et du nom du joueur, nous allons créer un pointeur `racineResultats` qui permet de référencer le nœud racine du document initial `<resultats>`.

```
var racineResultats = nouveauResultatXML.firstChild;
```

Une fois cette référence mise en place, il suffit d'utiliser les méthodes du DOM pour parcourir l'arbre XML afin de récupérer les valeurs des nœuds texte des éléments `<nom>` et `<gain>`.

```
var gainTxt=racineResultats.childNodes[1].firstChild.nodeValue;
var gagnantTxt=racineResultats.childNodes[0].firstChild.nodeValue;
```

Il ne reste plus ensuite qu'à affecter ces nouvelles valeurs aux balises `<span>` d'identifiant `resultat` et `gagnant` avec la fonction `remplacerContenu()` comme l'illustre ce code ci-dessous.

```
remplacerContenu("resultat",gainTxt);
remplacerContenu("gagnant",gagnantTxt);
```

Code 12-4 : représentation partielle de la fonction `actualiserPage()` après modification :

```
function actualiserPage() {
if (objetXHR.readyState == 4) {//teste si le résultat est disponible
  if (objetXHR.status == 200) {
    //récup du résultat dans un arbre XML
    var nouveauResultatXML = objetXHR.responseXML;
    //création d'un pointeur racine du résultat
    var racineResultats = nouveauResultatXML.firstChild;
    //récup des valeurs des éléments nom et gain dans l'arbre
    var gainTxt=racineResultats.childNodes[1].firstChild.nodeValue;
    var gagnantTxt=racineResultats.childNodes[0].firstChild.nodeValue;
    //actualisation des résultats
    remplacerContenu("resultat",gainTxt);
    remplacerContenu("gagnant",gagnantTxt);
    //affiche la zone info
    document.getElementById("info").style.visibility="visible";
    //gestion du bouton et du chargeur
    document.getElementById("button").disabled= false;
    document.getElementById("charge").style.visibility="hidden";
  }
}
}
```

Il existe cependant une autre alternative pour récupérer les données dans l'arbre XML résultat. Celle-ci exploite la méthode `getElementsByTagName()` qui permet de récupérer la liste (dans un tableau de variables) de tous les éléments dont le nom de la balise est

identique à celui qui est passé en argument de la méthode. Ainsi, l'instruction suivante récupère un tableau de tous les éléments portant le nom `gain` :

```
var gainTableau=nouveauResultatXML.getElementsByTagName("gain");
```

Si vous désirez ensuite accéder à l'une des valeurs de ce tableau, il faut alors mentionner son indice entre crochets comme ci-dessous.

```
var gainElement=nouveauResultatXML.getElementsByTagName("gain")[0];
```

ou encore :

```
var gainElement= gainTableau[0];
```

Dans notre exemple, nous n'avons qu'un seul élément de nom `gain` et il est donc facilement récupérable par l'indice 0 mais si nous en avions plusieurs, il serait alors judicieux d'utiliser une boucle afin de pouvoir traiter rapidement tous les éléments du tableau comme l'illustre le code ci-dessous.

```
for(var i=0; i<gainTableau.length;i++) {
  alert(gainTableau[i]);//traitement des éléments
}
```

Dans le cadre de notre atelier, si nous appliquons cette nouvelle méthode pour récupérer les valeurs des résultats, les nouveaux codes de la fonction seraient alors les suivants (les instructions du code 12-5 ci-dessous devraient alors remplacer les lignes en gras du code 12-4).

Code 12-5 : solution alternative avec la méthode `getElementsByTagName()` :

```
//récup du résultat dans un arbre XML
var nouveauResultatXML = objetXHR.responseXML;
//récupération des valeurs dans l'arbre XML
var gainTxt=nouveauResultatXML.getElementsByTagName("gain")[0].firstChild.nodeValue;
var gagnantTxt=nouveauResultatXML.getElementsByTagName
➥("nom")[0].firstChild.nodeValue;
//actualisation des résultats
remplacerContenu("resultat",gainTxt);
remplacerContenu("gagnant",gagnantTxt);
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Sous Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Le fonctionnement du nouveau système doit être exactement le même que celui de l'atelier 12-1, hormis le fait que le transfert des résultats est effectué cette fois à l'aide d'un document XML.

Pour visualiser les données renvoyées par le serveur lors de la réponse, nous vous conseillons d'activer Firebug et de cliquer sur l'onglet Console. Commencez par cliquer sur le bouton Clear de sorte à effacer d'éventuels enregistrements antérieurs. Saisissez ensuite votre nom et cliquez sur le bouton JOUER. Une nouvelle requête envoyée à `gainAleatoire.php` doit alors apparaître dans la fenêtre de l'onglet Console. Cliquez sur le

+ qui précède le nom du fichier pour le déplier puis sur l'onglet Response. Vous devriez voir le document XML renvoyé par le serveur (voir figure 12-3).

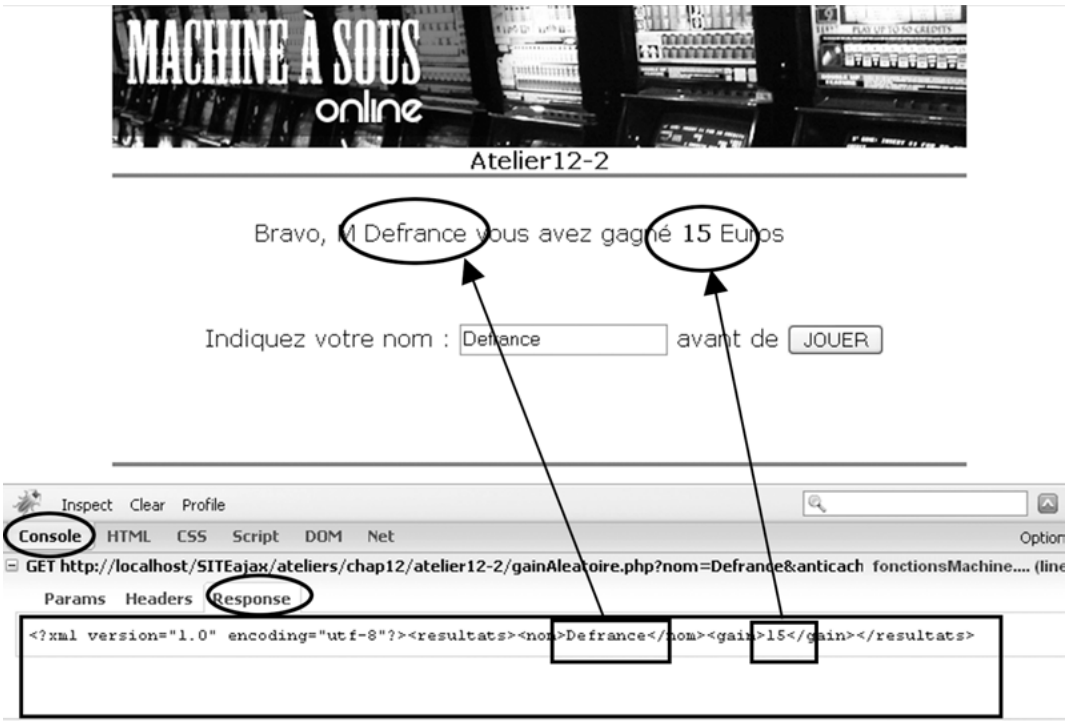


Figure 12-3

Test du transfert d'une réponse XML

Atelier 12-3 : requête avec réponse en JSON sans bibliothèques externes

Composition du système

Nous avons déjà utilisé le format JSON dans le cadre de l'atelier 11-5 pour l'envoi des paramètres de la requête vers le serveur. Nous allons maintenant nous intéresser à l'utilisation de JSON pour formater les données renvoyées dans la réponse du serveur. Pour le codage et le décodage des données, nous pourrions utiliser dès maintenant des bibliothèques externes comme nous l'avons fait pour l'atelier 11-5, mais nous avons préféré nous en passer dans un premier temps afin de vous démontrer que c'est possible de travailler sans, mais cette solution n'est pas sans poser de problèmes.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure est identique à celle de l'atelier 12-2 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;

- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est identique à celle de l'atelier 12-2 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 12-2 mais qui est modifiée pour gérer la réponse XML ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 12-2 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système est strictement identique à celui de l'atelier 12-2 que nous venons de réaliser hormis le fait que le transfert de la réponse du serveur est réalisé au format JSON.

Conception du système

Ouvrez la page HTML de l'atelier 12-2 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap12/atelier12-3/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Pour adapter le système au transfert d'une réponse encodée en JSON, nous allons commencer par modifier le fichier PHP `gainAleatoire.php` afin de coder les données de la réponse avant qu'elles ne soient renvoyées au navigateur.

Pour établir une structure de réponse JSON équivalente à celle de l'atelier précédent (nom du joueur et son gain), nous allons nous appuyer sur le document XML envoyé précédemment. Celui-ci est constitué d'un élément racine `<resultats>` et de deux éléments enfants `<nom>` et `<gain>` contenant chacun les données à transmettre au navigateur comme l'illustre le code de l'exemple ci-dessous.

Code 12-6 : rappel de la structure XML du résultat de l'atelier 12-2 :

```
<resultats>
   <nom>Defrance</nom>
   <gain>87</gain>
</resultats>
```

Si nous devions concevoir un objet JavaScript qui permettrait de recréer une structure équivalente à ce document XML, il serait alors constitué d'un objet principal contenant une seule propriété dont la clé serait nommée `resultats` dans laquelle nous aurions un autre objet contenant les propriétés du nom et du gain (voir code 12-7).

Code 12-7 :

```
objetJSON=
{
"resultats":
    {
        "nom":"Defrance",
        "gain":87
    }
} ;
```

Le format JSON, basé sur la syntaxe des objets JavaScript, reprend une partie de cette syntaxe mais sans le nom de la variable ni l'opérateur d'affectation (=) et le point virgule qui clôture l'instruction (revoir si besoin la syntaxe des objets JS dans le chapitre 19 consacré au JavaScript dans la partie 4 sur les ressources technologiques).

Nous pouvons donc déduire que la structure JSON équivalente à l'exemple précédent est semblable à celle du code 12-8 ci-dessous.

Code 12-8 :

```
{
"resultats":
    {
        "nom":"Defrance",
        "gain":87
    }
}
```

Pour matérialiser en PHP cette nouvelle structure de données dans une chaîne de caractères qui est envoyée à l'application cliente et y intégrer les valeurs issues du traitement serveur (nom réceptionné par le client dans la variable `$nom` et montant aléatoire dans la variable `$gain`) nous allons utiliser une série de concaténation comme l'illustre le code ci-dessous (l'ensemble étant sauvegardé dans la variable `$resultat`).

```
$resultat='{"resultats":{"nom":"'.$nom.'","gain":'.$gain.'}}';
```

Ainsi, la chaîne de caractères de la réponse du serveur est semblable à celle de l'exemple du code 12-8. La réponse étant maintenant prête à être renvoyée au navigateur, la procédure d'encodage peut donc se résumer à cette seule ligne d'instruction. Après cette modification, le fichier PHP doit être semblable au code 12-9 ci-dessous.

Code 12-9 : fichier `gainAleatoire.php` après modification :

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
sleep(2);
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
$gain = rand(0,100);
$resultat='{"resultats":{"nom":"'.$nom.'","gain":'.$gain.'}}';
echo $resultat;
```

Nous allons maintenant passer côté client pour mettre en place le décodage JSON de cette réponse afin de pouvoir actualiser les données de la page de l'application en conséquence. Ouvrez pour cela le fichier `fonctionsMachine.js` et localisez la fonction `actualiserPage()`.

Le format JSON s'appuyant sur la syntaxe des objets JavaScript (comme nous venons de le voir), il est maintenant très simple de transformer la réponse du serveur (actuellement sous forme d'une chaîne de caractères semblable à celle de l'exemple du code 12-8) en un objet JavaScript équivalent qui devrait reprendre la forme de l'objet JavaScript à partir duquel nous sommes parti pour élaborer la syntaxe de la chaîne de caractères JSON (voir code 12-7).

Pour convertir cette chaîne en un objet, nous pouvons simplement utiliser la fonction JavaScript `eval()` en passant comme argument la chaîne à convertir.

```
var objetJSON=eval('('+nouveauResultat+')');
```

Vous remarquez dans l'instruction ci-dessus que l'argument passé à la fonction `eval()` est lui-même encadré par une deuxième série de parenthèses, ceci afin d'indiquer qu'il s'agit d'une expression à évaluer et non d'une instruction à exécuter.

Une fois l'objet JSON récréé, il suffit d'utiliser la syntaxe pointée qui permet de lire les propriétés d'un objet JavaScript afin d'obtenir les valeurs du gain et le nom du joueur.

```
var gainTxt=objetJSON.resultats.gain;
var gagnantTxt=objetJSON.resultats.nom;
```

Après ces modifications, la fonction `actualiserPage()` devrait être semblable au code 12-10 ci-dessous.

Code 12-10 :

```
function actualiserPage() {
if (objetXHR.readyState == 4) {
 if (objetXHR.status == 200) {
  var nouveauResultat = objetXHR.responseText;
  var objetJSON=eval('('+nouveauResultat+')');
  var gainTxt=objetJSON.resultats.gain;
  var gagnantTxt=objetJSON.resultats.nom;
  remplacerContenu("resultat",gainTxt);
  remplacerContenu("gagnant",gagnantTxt);
  document.getElementById("info").style.visibility="visible";
  //gestion du bouton et du chargeur
  document.getElementById("button").disabled= false;
  document.getElementById("charge").style.visibility="hidden";
 }
}
}
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Sous Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Le fonctionnement du nouveau système doit être exactement le même que celui de l'atelier 12-2 précédent, hormis le fait que le transfert des résultats est cette fois effectué à l'aide de données au format JSON.

Pour visualiser les données renvoyées par le serveur lors de la réponse, nous vous conseillons d'activer Firebug et de cliquer sur l'onglet Console. Saisissez ensuite votre nom et cliquez sur le bouton JOUER. Une nouvelle requête envoyée à `gainAleatoire.php` doit alors apparaître dans la fenêtre de l'onglet Console. Cliquez sur le + qui précède le nom du fichier pour le déplier puis sur l'onglet Response. Vous devriez y voir la réponse au format JSON renvoyée par le serveur (voir figure 12-4).

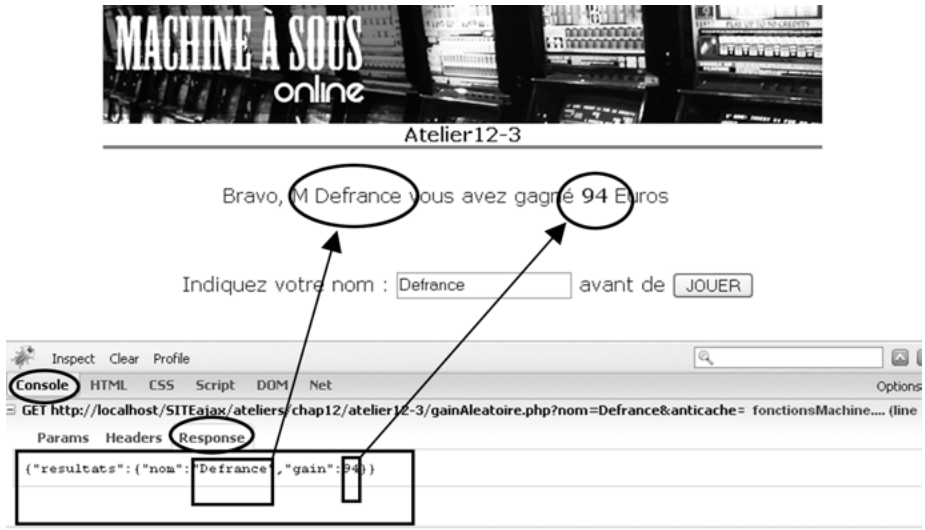


Figure 12-4

Test du transfert d'une réponse JSON

Atelier 12-4 : requête avec réponse en JSON avec bibliothèques externes

Composition du système

Dans l'atelier précédent, nous avons présenté une solution très simple pour effectuer le transfert d'une réponse serveur au format JSON.

Cependant, l'utilisation de la fonction `eval()` côté client présente des problèmes de sécurité. En effet, aucun contrôle n'est effectué afin de vérifier que le contenu de la réponse ne contient pas des scripts malveillants, ce qui pourrait avoir des conséquences regrettables pour votre site…

De même, côté serveur nous avons réalisé nous-même l'élaboration de la réponse au format JSON. Cela a été une tâche relativement facile à développer mais imaginez que la réponse soit beaucoup plus complexe ou encore que la structure de cette réponse soit amenée à changer d'une requête à l'autre. Il deviendrait alors très compliqué d'élaborer une réponse JSON correspondante.

La solution à ces deux problèmes consiste à utiliser des bibliothèques externes (en JS côté client et en PHP côté serveur) qui prennent en charge les tâches d'encodage ou de décodage au format JSON. Nous avons déjà présenté ces deux bibliothèques dans l'atelier 11-5 dans lequel nous les avons exploitées pour envoyer les paramètres d'une requête au format JSON du navigateur vers le serveur. Nous allons maintenant les utiliser dans le sens inverse, afin d'effectuer le transfert de la réponse du serveur vers le navigateur au format JSON.

Cette structure est composée :

- d'une page HTML (`index.html`) identique à celle de l'atelier 12-3 ;

- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant sa modification est identique à celle de l'atelier 12-3 ;
- d'une bibliothèque externe Json (`json.js`) que nous avons déjà utilisée dans l'atelier 11-5 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celui de l'atelier 12-3 mais qui va être modifié pour assurer l'encodage automatique des données JSON à envoyer ;
- d'une bibliothèque externe JSON-PHP (`JSON.php`) que nous avons déjà utilisée dans l'atelier 11-5 ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 12-3 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système est strictement identique à celui de l'atelier 12-3 que nous venons de réaliser hormis le fait que l'encodage (côté serveur) et le décodage (côté client) au format JSON sont réalisés à l'aide de bibliothèques externes.

Conception du système

Ouvrez la page HTML de l'atelier 12-3 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap12/atelier12-4/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Commençons par copier les bibliothèques `json.js` et `JSON.php` utilisées dans l'atelier 11-5 dans le même répertoire que celui de l'atelier actuel.

Ouvrez ensuite le programme `gainAleatoire.php` et ajoutez l'instruction `require_once()` afin de pouvoir disposer de la bibliothèque JSON-PHP dans ce programme serveur.

```
require_once('JSON.php');
```

Avant d'encoder les données, il faut d'abord commencer par créer un objet PHP (constitué de deux tableaux associatifs dans le cas de cet atelier) et ses valeurs correspondant à la structure désirée.

Code 12-11 : création de l'objet PHP :

```
$tableauNomGain=array(
          "nom"=>$nom,
          "gain"=>$gain);
$tableauResultat = array("resultats"=>$tableauNomGain);
```

Créez ensuite un objet JSON à l'aide de la classe `Services_JSON()`.

```
$objetJSON = new Services_JSON();
```

Une fois les deux objets créés, nous pouvons alors passer à l'encodage à l'aide de la méthode `encode()` de l'objet JSON (l'objet PHP à encoder étant passé en argument de cette même méthode).

```
$resultat = $objetJSON->encode($tableauResultat);
```

La variable `$resultat` contient désormais une chaîne de caractères formatée en JSON qui devrait être semblable à celle que nous avons réalisée par concaténation dans l'atelier précédent (revoir si besoin le code 12-8).

Une fois les modifications effectuées, le fichier PHP devrait être semblable au code 12-12.

Code 12-12 : fichier `gainAleatoire.php` après modification :

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
sleep(2);
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
$gain = rand(0,100);
require_once('JSON.php');
$tableauNomGain=array(
        "nom"=>$nom,
        "gain"=>$gain);
$tableauResultat = array("resultats"=>$tableauNomGain);
$objetJSON = new Services_JSON();
$resultat = $objetJSON->encode($tableauResultat);
echo $resultat;
```

Comme dans l'atelier 11-5, nous vous signalons que l'extension `json` est désormais intégrée au PHP 5.2 et versions ultérieures. Il est donc possible de se passer de la bibliothèque JSON-PHP dans ce cas. Le script correspondant à cette seconde alternative est alors semblable à celui du code 12-13.

Code 12-13 : fichier `gainAleatoire.php` pour PHP 5.2 et plus exclusivement :

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
sleep(2);
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
$gain = rand(0,100);
$tableauNomGain=array(
        "nom"=>$nom,
        "gain"=>$gain);
$tableauResultat = array("resultats"=>$tableauNomGain);
$resultat = json_encode($tableauResultat);
echo $resultat;
```

Les modifications du programme serveur sont maintenant terminées et vous pouvez enregistrer le fichier `gainAleatoire.php`, nous allons maintenant nous intéresser au décodage JSON côté client et ouvrir pour cela les fichiers `index.html` et `fonctionsMachine.js`.

Dans le fichier `index.html`, ajoutez une balise `<script>` de sorte à faire référence à notre bibliothèque `json.js`.

```
<script type="text/javascript" src="json.js"></script>
```

Localisez ensuite la fonction de rappel du moteur Ajax nommée `actualiserPage()` dans le fichier `fonctionsMachine.js`. Cette fonction étant appelée lors de la réception de la réponse par le navigateur, c'est ici que nous allons mettre en place le processus de décodage des données JSON. Pour cela, nous allons appliquer la méthode `parseJSON()` de la bibliothèque à la réponse retournée par le serveur (réponse actuellement enregistrée dans

la variable `nouveauResultat`). Cette instruction aura pour effet de décoder la chaîne de caractères JSON de la réponse et de créer un objet JSON en rapport (dans `objetJSON`).

```
var objetJSON=nouveauResultat.parseJSON();
```

Une fois l'objet recréé, il est alors facile d'accéder à ses propriétés pour récupérer les informations nécessaires à l'actualisation de la page de l'application (à savoir le gain et le nom du joueur).

```
var gainTxt=objetJSON.resultats.gain;
var gagnantTxt=objetJSON.resultats.nom;
```

Après ces modifications, la fonction `actualiserPage()` devrait être semblable au code 12-14 :

Code 12-14 : fonction `actualiserPage()` :

```
function actualiserPage() {
if (objetXHR.readyState == 4) {
 if (objetXHR.status == 200) {
  var nouveauResultat = objetXHR.responseText;
  var objetJSON=nouveauResultat.parseJSON();
  var gainTxt=objetJSON.resultats.gain;
  var gagnantTxt=objetJSON.resultats.nom;
  remplacerContenu("resultat",gainTxt);
  remplacerContenu("gagnant",gagnantTxt);
  document.getElementById("info").style.visibility="visible";
  //gestion du bouton et du chargeur
  document.getElementById("button").disabled= false;
  document.getElementById("charge").style.visibility="hidden";
 }
}
}
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Sous Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Le fonctionnement du nouveau système est le même que celui de l'atelier 12-3 précédent, hormis le fait que l'encodage et le décodage au format JSON sont réalisés à l'aide de bibliothèques externes.

Atelier 12-5 : requête avec réponse en XML et conversion JSON

Composition du système

Dans le dernier atelier, nous avons étudié une solution opérationnelle pour transférer une réponse serveur au format JSON. Cependant, dans certains cas, il n'est pas possible d'accéder au code serveur pour configurer le format de la réponse en JSON (c'est notamment le cas de la plupart des Web services qui mettent à votre disposition un flux de données en XML et non en JSON).

Si, malgré tout, vous désirez quand même traiter ces flux de données en JSON, il est alors possible de mettre en place un convertisseur qui permet de transformer le flux XML en JSON. Cet atelier vous présente une illustration de cette technique.

Cette structure est composée :

- d'une page HTML (`index.html`) identique à celle de l'atelier 12-4 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant sa modification est identique à celle de l'atelier 12-4 ;
- d'une bibliothèque externe de conversion XML vers JSON (`xml2json.js`) ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celui de l'atelier 12-2 (renvoi d'une réponse en XML) ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 12-4 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système est semblable à celui de l'atelier 12-4 que nous venons de réaliser hormis le fait que le transfert de la réponse est effectué en XML et qu'il est ensuite converti en JSON dès son arrivée sur le poste client.

Conception du système

Ouvrez la page HTML de l'atelier 12-4 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap12/atelier12-5/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier sauf le fichier `gainAleatoire.php` qui est issu de l'atelier 12-2 (réponse serveur en XML).

Pour réaliser la conversion, nous allons utiliser une bibliothèque externe nommée `xml2json`. Commencez donc par télécharger cette bibliothèque sur votre ordinateur puis copiez ensuite le fichier `xml2json.js` dans le répertoire de l'atelier en cours.

Bibliothèque xml2json

Auteur : Thomas Frank (free software licence GNU)

Vous pouvez télécharger la bibliothèque xml2json à l'adresse ci-dessous :

http://www.thomasfrank.se/xml_to_json.html

Ouvrez le fichier `index.html` pour y ajouter une balise `<script>` afin de faire référence à la bibliothèque `xml2json.js`.

```
<script type="text/javascript" src="xml2json.js"></script>
```

Le fichier PHP étant issu de l'atelier 12-2, il est déjà configuré pour renvoyer un document XML contenant les données `nom` et `gain` (pour mémoire, la structure de ce document est semblable à celle de l'exemple du code 12-6).

Par contre, nous allons devoir modifier la fonction de rappel du moteur Ajax de sorte a intégrer le script de conversion. Ouvrez le fichier `fonctionsMachine.js` et localisez la fonction `actualiserPage()`. La première chose à mettre en place est la récupération du document XML envoyé en réponse par le serveur. Contrairement à ce que l'on pourrait penser, nous n'allons pas récupérer ce document XML avec la propriété `responseXML` mais avec `responseText`. En effet, dans notre cas nous désirons disposer du document XML sous la forme d'une chaîne de caractères et non pas sous la forme d'un arbre DOM XML.

```
var nouveauResultat = objetXHR.responseText;
```

Dès que nous disposons du document XML dans la variable `nouveauResultat`, nous pouvons lui appliquer la méthode `parse()` de la bibliothèque externe xml2json qui exécute la conversion.

```
var objetJSON=xml2json.parser(nouveauResultat);
```

Une fois la conversion effectuée, nous disposons du résultat dans un objet JSON et nous pouvons ainsi facilement lire ses propriétés en utilisant la syntaxe pointée appliquée à l'objet comme l'illustre le code ci-dessous.

```
var gainTxt=objetJSON.resultats.gain;
var gagnantTxt=objetJSON.resultats.nom;
```

La nouvelle fonction `actualiserPage()` devrait être maintenant semblable à celle du code 12-15.

Code 12-15 :

```
function actualiserPage() {
if (objetXHR.readyState == 4) {
 if (objetXHR.status == 200) {
   var nouveauResultat = objetXHR.responseText;
   var objetJSON=xml2json.parser(nouveauResultat);
   var gainTxt=objetJSON.resultats.gain;
   var gagnantTxt=objetJSON.resultats.nom;
   remplacerContenu("resultat",gainTxt);
   remplacerContenu("gagnant",gagnantTxt);
   document.getElementById("info").style.visibility="visible";
   //gestion du bouton et du chargeur
   document.getElementById("button").disabled= false;
   document.getElementById("charge").style.visibility="hidden";
  }
}
}
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Sous Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Le fonctionnement du nouveau système est le même que celui de l'atelier 12-4 précédent, hormis le fait que le transfert de la réponse est réalisé au format XML et qu'une conversion JSON est ensuite réalisée sur le poste client à l'aide de la bibliothèque externe xml2json.

Atelier 12-6 : requête avec réponse RSS

Composition du système

Dans l'atelier 12-2, nous vous avons déjà présenté un système exploitant une réponse du serveur au format XML. Nous vous proposons maintenant de voir une application spécifique de ce système en réalisant un lecteur très simple de flux RSS.

Les flux RSS (*Really Simple Syndication*) permettent à des serveurs fournisseurs d'informations de diffuser facilement des articles qui pourront être ensuite repris par divers sites Web ou lus directement avec des applications comme le gestionnaire de messagerie.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure est identique à celle de l'atelier 12-2 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est identique à celle de l'atelier 12-2 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier 12-2 ;
- d'un nouveau fichier serveur PHP (`proxyRss.php`) ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 12-2 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système de la machine à sous est strictement identique à celui de l'atelier 12-2. Nous allons simplement ajouter un tableau HTML en bas de l'écran affichant les différentes actualités que nous allons récupérer sur le site du fournisseur d'informations. Chaque item comporte un titre et est configuré pour être cliquable afin d'afficher la page d'origine de l'information.

Conception du système

Ouvrez la page HTML de l'atelier 12-2 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap12/atelier12-6/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Nous allons commencer par ajouter le tableau HTML qui va réceptionner les nouvelles dans la page `index.html`. Pour cela, ouvrez le fichier `index.html` et placez le code 12-16 en bas de la page juste après la balise de fermeture de la zone `<div>` d'identifiant `page`.

Code 12-16 : balises du tableau HTML destiné à afficher les actualités du flux RSS :

```
<div id="nouvelles">
 <table width="400" border="1" align="center" cellspacing="0">
 <tbody id="tableRss" >
  <tr>
   <td> LISTE DE NOUVELLES RSS </td>
```

```
  </tr>
 </tbody>
 </table>
</div>
```

Afin d'être en conformité avec le W3C, nous avons ajouté une balise `<tbody>` qui regroupe les différentes lignes du corps du tableau (voir l'encadré ci-dessous pour plus d'information sur la balise `tbody`). De même, nous avons configuré un identifiant `tableRss` pour cette même balise afin de pouvoir y ajouter par la suite les futures lignes qui contiendront les titres des actualités.

Balises thead, tbody et tfoot

Ces balises permettent de structurer les données dans l'élément `table`. Ainsi, la balise `thead` permet de regrouper les informations concernant l'en-tête du tableau comme un titre ou le nom des colonnes par exemple. La balise `tbody`, quant à elle, permet, de regrouper les différentes lignes du corps du tableau et enfin la balise `tfoot` regroupe des éventuelles remarques ou légendes.

Le véritable intérêt de cette nouvelle organisation est de permettre de visualiser les différentes lignes de données du corps avec un scroll si la taille de la table est importante. Ainsi, vous pouvez faire défiler les lignes contenues dans la balise `tbody` alors que les lignes des balises `thead` et `tfoot` restent toujours visibles.

Enregistrez la page `index.html` et ouvrez maintenant le fichier `fonctionsMachine.js`. Copiez les deux fonctions `jouer()` et `actualiserPage()` et collez-les en bas du fichier afin de partir de cette base pour construire le nouveau moteur Ajax pour la gestion du lecteur RSS. Renommez la première fonction `lectureRss()` et la deuxième `afficheRss()`, puis sélectionnez les deux fonctions et lancez un Rechercher/Remplacer pour changer le nom de l'objet XHR actuel en `objetXHR2`.

Modifiez ensuite le second argument de la méthode `open()` en le remplaçant par `proxyRss.php` qui permet d'assurer la liaison avec le serveur fournisseur d'information. Modifiez aussi le nom de la déclaration de la fonction de rappel pour qu'elle corresponde avec celui de la seconde fonction du moteur soit `afficheRss`.

```
objetXHR2.open("get","proxyRss.php", true);
objetXHR2.onreadystatechange = afficheRss;
```

Une fois modifiée, la fonction `lectureRss()` doit être semblable au code 12-17.

Code 12-17 : fonction `lectureRss()` :

```
function lectureRss() {
 objetXHR2 = creationXHR();
 objetXHR2.open("get","proxyRss.php", true);
 objetXHR2.onreadystatechange = afficheRss;
 objetXHR2.send(null);
 setTimeout("lectureRss()",6000);
}
```

Pour assurer le rafraîchissement des actualités d'une manière périodique, nous allons configurer en bas de la fonction du moteur Ajax une instruction `setTimeOut()` qui permet d'appeler la fonction dans laquelle elle se trouve toutes les 6 secondes. Pour déclencher

cette temporisation, il faut appeler la fonction une première fois. Pour cela, nous allons ajouter un appel de `lectureRss()` dans la fonction `testerNavigateur()` qui est appelée à la fin du chargement de la page `index.html` (voir code 12-18).

Code 12-18 : ajout de l'appel de `lectureRss()` :

```
function testerNavigateur() {
 …
 lectureRss();
}
```

Proxy et serveurs mandataires

Les applications Ajax ne peuvent pas diffuser directement des informations issues d'un autre serveur Web que celui sur lequel elles sont installées. Ces restrictions de sécurité sont imposées par les navigateurs mais peuvent être contournées en utilisant un serveur relais qui se charge de récupérer les informations sur le site externe. L'application Ajax peut ensuite communiquer avec le serveur relais et disposer ainsi des informations mises à sa disposition.

Passons maintenant côté serveur pour créer le fichier PHP qui fait office de proxy (voir encadré) entre le client et le serveur fournisseur d'information. Créez un nouveau fichier PHP et enregistrez-le sous le nom `proxyRss.php`. Configurez ses en-têtes pour bloquer la mise en cache et pour signaler que le contenu de la réponse renvoyée est au format XML.

```
header("Content-Type: text/xml");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
```

Pour accéder au fichier XML du flux RSS, nous allons utiliser simplement la fonction `readfile()` qui permet d'afficher tout le contenu du fichier XML à l'écran. Pour vos tests, nous vous conseillons d'appeler directement ce fichier depuis le navigateur pour vous assurer que la source XML du RSS est bien formée. Dans le cadre de cet ouvrage, nous avons utilisé un flux RSS du site `www.infos-du-net.com` mais vous pouvez bien sûr utiliser un autre flux RSS 2.0 si vous le désirez.

```
readfile("http://www.infos-du-net.com/feeds/1602-rss2-autres.xml");
```

Le fichier `proxyRss.php` est maintenant terminé et devrait être semblable au code 12-19.

Code 12-19 : fichier `proxyRss.php` :

```
header("Content-Type: text/xml");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
readfile("http://www.infos-du-net.com/feeds/1602-rss2-autres.xml");
```

Avant de commencer à développer la fonction de rappel du moteur Ajax, nous vous invitons à faire un premier test du fichier `ProxyRss.php` que nous venons de réaliser dans un navigateur (voir figure 12-5). Si le document XML du flux RSS est bien formé, le navigateur doit afficher les différentes balises des informations des actualités d'une manière hiérarchique.

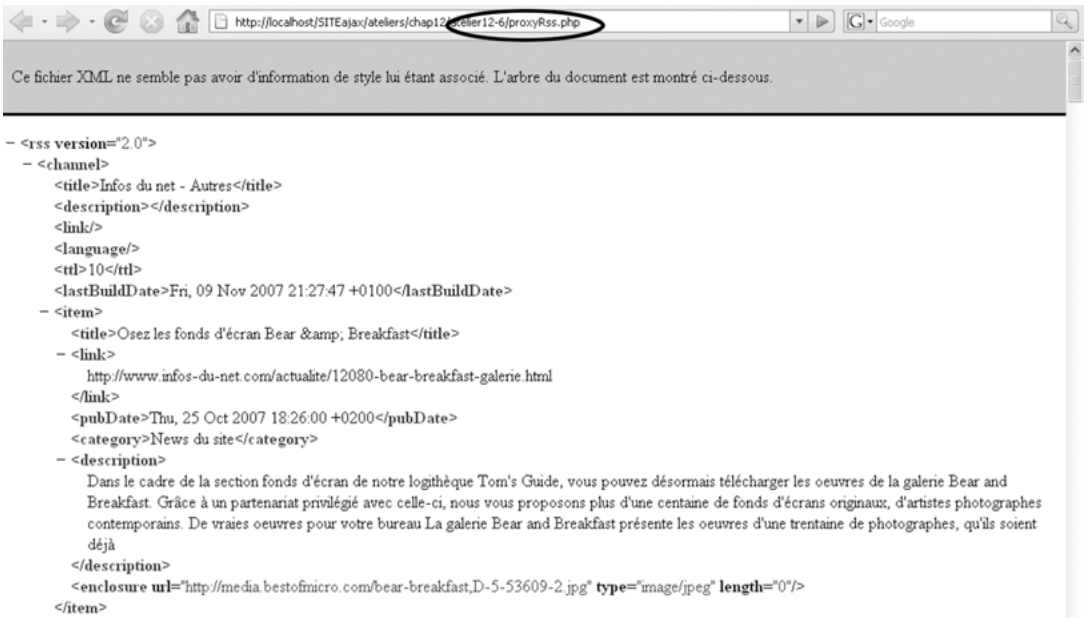


Figure 12-5

Test du fichier proxyRss.php appelé directement

Il existe plusieurs formats de flux RSS mais dans le cadre de cet atelier, nous allons utiliser un format RSS 2.0. Son contenu comprend toujours un en-tête et une liste d'items. Si nous représentions une structure minimale du document, nous aurions un document semblable au code 12-20.

Code 12-20 : structure minimale d'un flux RSS 2.0 :

```
<rss version="2.0">
<channel>
 <title>Infos du net - Autres</title>
 <description>Actualité de l'Internet</description>
 <link/>
  <item>
   <title>Google lance gmail</title>
   <description>…</description>
   <link>http://www.google.com</link>
  </item>
  <item>
   <title>…</title>
   <description>…</description>
   <link>…</link>
  </item>
<channel>
</rss>
```

Revenons maintenant côté client pour configurer la fonction de rappel du moteur Ajax qui permet de récupérer et mettre en forme les nouvelles du flux RSS.

L'objectif de la fonction de rappel est de récupérer le contenu des balises `title` et `link` de chaque item (voir code 12-20). Ces informations devront ensuite être intégrées dans le tableau situé en bas de la page `index.html` de sorte que l'utilisateur puisse accéder à la page d'origine d'une actualité en cliquant sur son titre.

Ouvrez de nouveau le fichier `fonctionMachine.js` et localisez la fonction `afficheRss()`. Configurez une instruction pour récupérer le document XML dans une variable nommée `nouveauRss`.

```
var nouveauRss=objetXHR2.responseXML;
```

Créez ensuite un pointeur sur la liste des éléments `<item>` afin de pouvoir ensuite accéder à chaque item en modifiant l'indice du tableau de variable `listeItem[1]`, `listeItem[2]`...

```
var listeItem=nouveauRss.getElementsByTagName("item");
```

Nous allons ensuite créer une boucle `for()` afin de parcourir tous les items du document RSS. À chaque tour de boucle, nous allons récupérer le titre et le lien de l'item pour ensuite les communiquer en paramètres à une fonction `nouvelleLigne` qui permet de créer une ligne supplémentaire du tableau à chaque itération.

```
for(i=0;i<listeItem.length;i++){
 var titre=listeItem[i].getElementsByTagName("title")[0].firstChild.nodeValue;
 var lien=listeItem[i].getElementsByTagName("link")[0].firstChild.nodeValue;
 nouvelleLigne(titre,lien);
}//fin du for
```

La fonction `afficheRss()` complète doit ensuite être semblable au code 12-21 ci-dessous.

Code 12-21 : fonction `afficheRss()` :

```
function afficheRss() {
if (objetXHR2.readyState == 4) {
 if (objetXHR2.status == 200) {
  var nouveauRss=objetXHR2.responseXML;
  var listeItem=nouveauRss.getElementsByTagName("item");
  for(i=0;i<listeItem.length;i++){
   var titre= listeItem[i].getElementsByTagName("title")[0].firstChild.nodeValue;
   var lien= listeItem[i].getElementsByTagName("link")[0].firstChild.nodeValue;
   nouvelleLigne(titre,lien);
  }//fin du for
 }//fin du if status
}//fin du if readyState
}//fin de function
```

Nous devons maintenant créer la fonction `nouvelleLigne()` qui est appelée à chaque tour de la boucle `for()` dans la fonction `afficheRss()`. Cette nouvelle fonction doit récupérer le titre et le lien de l'actualité afin de construire une ligne de tableau supplémentaire qui permettra à l'utilisateur de cliquer sur le titre pour accéder à la page d'origine de la nouvelle.

Pour construire cette ligne, nous allons utiliser les méthodes du DOM. Nous commençons par créer les éléments `tr`, `td`, `a` et `text` à l'aide de la méthode `createElement()`.

```
var nouveauTR=document.createElement('tr');
var nouveauTD=document.createElement('td');
```

```
var nouveauTXT=document.createTextNode(titre);
var nouveauA=document.createElement('a');
```

Une fois la balise `a` créée, nous pouvons configurer ses attributs `href` et `target` à l'aide de la méthode `setAttribute()`.

```
nouveauA.setAttribute('href',lien);
nouveauA.setAttribute('target','_blank');
```

Il faut ensuite attacher successivement tous les éléments créés précédemment en respectant leurs hiérarchies à l'aide de la méthode `appendChild()`.

```
nouveauA.appendChild(nouveauTXT);
nouveauTD.appendChild(nouveauA);
nouveauTR.appendChild(nouveauTD);
```

Enfin l'élément `tr` de la ligne est rattaché au tableau par le biais de la balise `tbody` (dont l'identifiant est `tableRss`).

```
document.getElementById("tableRss").appendChild(nouveauTR);
```

Une fois terminée, la fonction `nouvelleLigne()` doit être semblable au code 12-22.

Code 12-22 : fonction `nouvelleLigne()` :

```
function nouvelleLigne(titre,lien) {
  var nouveauTR=document.createElement('tr');
  var nouveauTD=document.createElement('td');
  var nouveauTXT=document.createTextNode(titre);
  var nouveauA=document.createElement('a');
  nouveauA.setAttribute('href',lien);
  nouveauA.setAttribute('target','_blank');
  nouveauA.appendChild(nouveauTXT);
  nouveauTD.appendChild(nouveauA);
  nouveauTR.appendChild(nouveauTD);
  document.getElementById("tableRss").appendChild(nouveauTR);
}
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

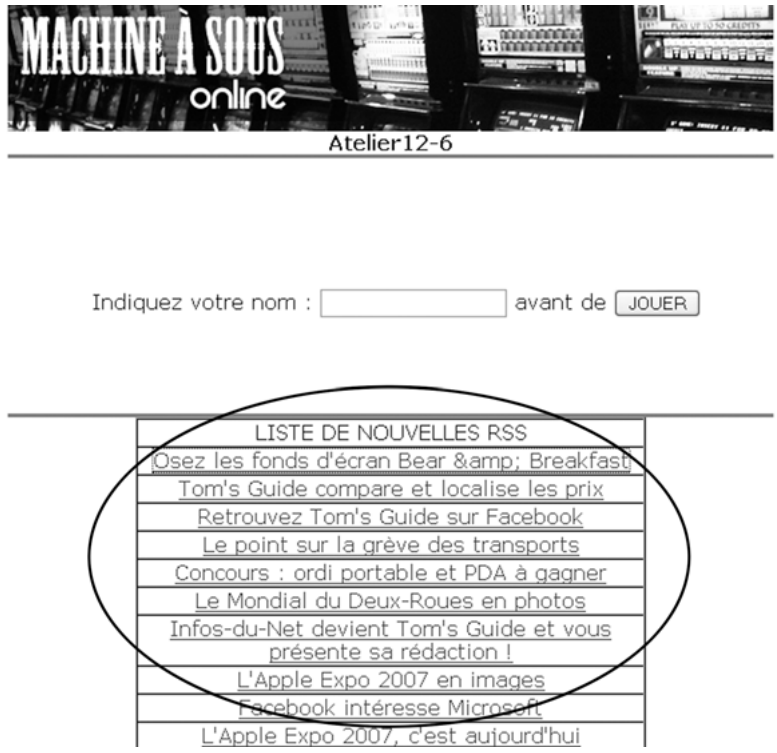
Test du système

Sous Dreamweaver, ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12. Le fonctionnement du jeu de la machine à sous restera inchangé par rapport à celui de l'atelier 12-2 mais avec cette fois un tableau des actualités du flux RSS doit apparaître en bas de l'écran. Dès le chargement de la page, la fonction de lecture du flux RSS doit être appelée et afficher toutes les actualités disponibles sur des lignes différentes du tableau HTML (voir figure 12-6). Ces informations seront ensuite réactualisées toutes les 6 secondes.

Si vous cliquez sur l'un des titres du tableau, une nouvelle fenêtre doit s'ouvrir et afficher la page d'origine de l'actualité.

Figure 12-6

Affichage des nouvelles du flux RSS



13

Applications Ajax-PHP-MySQL

Dans la pratique, la plupart des systèmes stockent leurs informations dans une base de données. Les applications Ajax-PHP n'échappent pas à cette règle, et nous vous proposons dans ce chapitre d'étudier les différentes techniques qui permettent ce type d'interactions entre l'application client et la base de données.

Pour illustrer ces techniques, nous allons reprendre notre application de machine à sous en ligne à laquelle nous allons associer de nouvelles fonctionnalités qui nécessitent la mise en œuvre de différents systèmes de communication avec la base de données.

Atelier 13-1 : vérification instantanée de la saisie dans une base de données

Composition du système

La vérification instantanée (ou presque) de données au fil de la saisie est une illustration typique de l'usage que l'on peut faire d'une application Ajax-PHP-MySQL. Pour l'appliquer à notre exemple de machine à sous, nous vous proposons de créer un système qui permet de vérifier à chaque nouveau caractère saisi si le nom du joueur est bien enregistré dans la base de données des joueurs du club.

Tant que la requête de vérification SQL à la base de données n'identifiera pas précisément le nom du joueur, un message informant l'utilisateur que son nom est inconnu s'affichera à l'écran et le bouton JOUER restera bloqué. En revanche, dès que le nom est reconnu, l'identifiant de l'enregistrement est alors renvoyé par le serveur à l'application, ce qui va déclencher l'affichage d'un message informant l'utilisateur qu'il a été identifié. Le bouton JOUER est ensuite débloqué et le joueur peut utiliser l'application comme dans les ateliers précédents.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure est identique à celle de l'atelier 10-4 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;

- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est identique à celle de l'atelier 10-4 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 10-4 ;
- d'un second fichier serveur PHP (`nomVerification.php`) qui va être créé dans cet atelier ;
- d'un fichier PHP de connexion à la base de données (`connexionMysql.php`) qui va être créé dans cet atelier ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 10-4 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du jeu est identique à celui de l'atelier 10-4 que nous allons prendre comme base de départ pour effectuer nos modifications. Cependant, la mise en place d'une nouvelle fonctionnalité permet d'interroger la base de données à chaque saisie d'un nouveau caractère dans le champ du nom et d'informer l'utilisateur dès qu'il est identifié. Le bouton JOUER reste bloqué tant que le nom du joueur n'est pas identifié.

Conception du système

Ressources sur les bases de données MySQL

Avant de passer à la conception de ce système, il est fortement conseillé d'avoir quelques notions sur la structure et le fonctionnement d'une base de données. Si ce n'est pas votre cas, nous vous invitons à lire au préalable le chapitre 22 dédié aux bases de données MySQL que vous pouvez trouver dans la partie 4 de cet ouvrage.

Ouvrez la page HTML de l'atelier 10-4 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap13/atelier13-1/`. Copiez ensuite les autres fichiers de l'atelier 10-4 dans ce nouveau dossier.

Avant de dialoguer avec la base de données, nous devons commencer par la créer. Pour cela, nous allons utiliser le gestionnaire de base de données phpMyAdmin, accessible depuis le manager de Wamp5 (deuxième item dans le menu du manager en partant du haut). Une fois le gestionnaire ouvert, la partie droite de l'écran affiche une zone intitulée Création d'une base. Saisissez le nom de la nouvelle base dans ce champ, soit `machineasous` et cliquez sur le bouton Créer.

Alternative à la création de la base

Si vous êtes déjà initié à l'utilisation de phpMyAdmin et que vous désirez créer rapidement la base de données `joueurs`, vous pouvez utiliser le fichier de sauvegarde SQL placé dans un répertoire du même nom dans le dossier de cet atelier.

Pour connaître la procédure de restitution de la base à l'aide de ce fichier, reportez-vous à la fin du chapitre 22 sur le langage SQL dans la partie 4 de cet ouvrage.

Dans la nouvelle fenêtre du gestionnaire, vous devez alors voir une nouvelle zone intitulée Création d'une nouvelle table. Saisissez le nom de la première table dans ce champ, soit `joueurs`, indiquez `3` pour le nombre de champs et cliquez sur le bouton Créer pour lancer la création de la table.

Un troisième écran doit alors apparaître (voir figure 13-1), chaque ligne correspondant aux caractéristiques d'un des futurs champs de la table. Le premier est dédié à la clé primaire de la table : une clé primaire est un champ obligatoire dont le contenu doit être unique et sert de clé de sélection pour accéder à un enregistrement précis. Le second est un champ texte correspondant au nom du joueur et le troisième à son prénom. Dans ce premier atelier, nous n'aurons pas l'usage de ce troisième champ, mais nous le créons quand même maintenant en prévision de sa future utilisation.

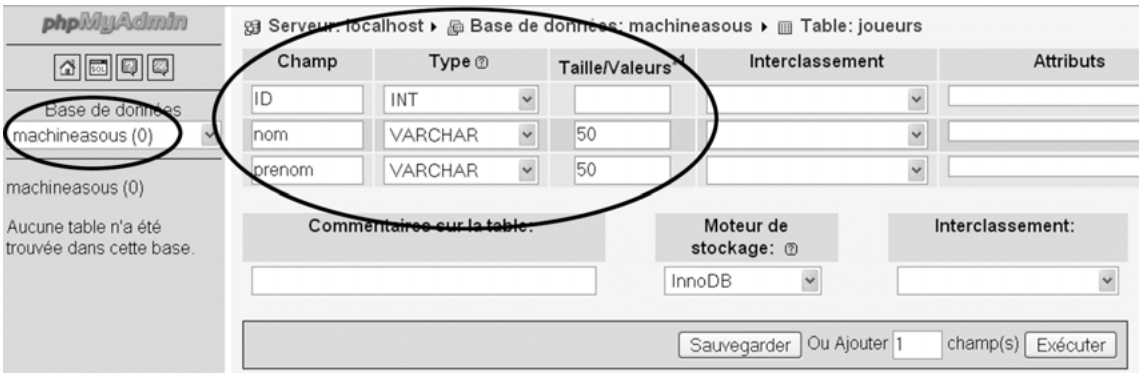


Figure 13-1

Création de la table joueurs (partie de gauche)

Après cette rapide présentation de la structure de notre nouvelle table, passons maintenant à sa configuration. La première colonne, nommée Champ, permet de nommer les futures colonnes de la table. Saisissez `ID` pour la première ligne afin d'identifier la colonne de la clé primaire. Puis saisissez `nom` et `prenom` en tête des deux autres lignes. La deuxième colonne, nommée Type permet de choisir le typage que doivent respecter les différentes valeurs du champ en rapport. Ainsi, pour celui de la clé primaire `ID`, nous choisissons `INT` pour indiquer que le type de la clé primaire est un nombre entier. Pour les deux autres champs, nous choisissons en revanche `VARCHAR` pour indiquer qu'il s'agit ici de chaînes de caractères. Certains types, comme `INT` suffisent amplement et ne nécessitent aucun complément de valeur, mais ce n'est pas le cas de `VARCHAR` pour lequel vous devez obligatoirement indiquer le nombre maximum de caractères que peut contenir la chaîne. Dans notre exemple, nous allons indiquer `50` pour les deux champs, ce qui nous semble amplement suffisant pour mémoriser le nom ou le prénom du joueur. Déplaçons nous maintenant sur la droite de l'écran à l'aide du curseur horizontal (voir figure 13-2). Pour la colonne nommée Extra, sélectionnez l'option `auto_increment` pour permettre l'auto-incrémentation de la valeur de la clé primaire à chaque insertion d'un nouvel enregistrement. Pour les deux autres lignes (`nom` et `prenom`), laissez leur menu déroulant tel quel sans sélectionner d'option. Il reste maintenant à préciser le champ `ID` qui va faire office de clé primaire. Pour cela, nous allons cocher le bouton radio de la colonne Primaire de cette première ligne. Les deux autres colonnes (Unique et Index) permettent de préciser qu'un champ peut être unique ou qu'il est utilisé en tant qu'index pour accéder aux

enregistrements dans les futures utilisations du système. Une clé primaire étant à la fois unique et index, il est donc inutile de cocher ces deux autres options pour le champ `ID`.

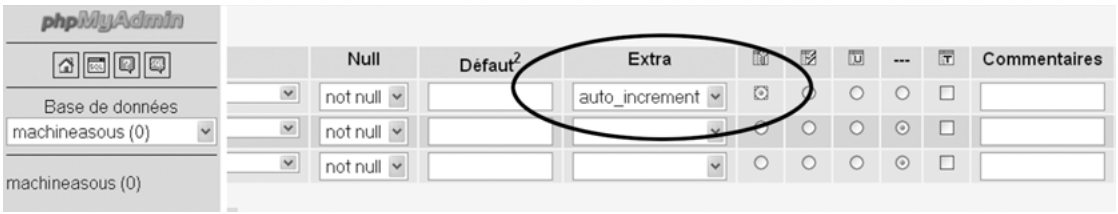


Figure 13-2

Création de la table joueurs (partie de droite)

> **Éviter les doublons pour les noms**
>
> Si l'application se limitait uniquement à la gestion des joueurs par leur nom, il faudrait alors cocher le bouton radio Unique du champ nom pour éviter d'avoir deux fois le même nom de joueur dans la table. Cependant, nous allons voir dans un prochain atelier que nous sommes amenés à gérer des joueurs de la même famille. Dans ce nouveau contexte, chaque enregistrement doit être identifié par son couple nom-prénom et il n'est donc plus nécessaire de cocher la case Unique pour le champ nom.

Il est inutile de configurer les autres colonnes de cet exemple très simple mais sachez tout de même que la colonne Interclassement permet de définir l'encodage utilisé s'il devait être différent de celui de la table ; la colonne Attributs permet d'indiquer, dans le cas de nombres, s'ils sont signés ou non ; la colonne Null permet de rendre optionnel un champ lors de l'insertion d'un nouvel enregistrement et enfin le champ Défaut, comme son nom l'indique, permet de proposer une valeur par défaut si celle-ci n'est pas renseignée.

La configuration du tableau `joueurs` est maintenant terminée. Pour valider vos choix, vous devez maintenant cliquer sur le bouton Sauvegarder. Attention : ne cliquez pas sur le bouton Exécuter, ce qui aurait comme effet de créer une ligne de champ supplémentaire. Un message doit ensuite s'afficher, confirmant que la création de la table a bien été effectuée (voir message ci-dessous).

```
Table `machineasous`.`joueurs` a été créé(e).
```

La table est donc créée mais ne contient pas encore de données. Pour ajouter de nouveaux enregistrements, cliquez sur l'onglet Insérer en haut de l'écran. Un nouveau formulaire apparaît alors vous proposant de saisir deux nouveaux enregistrements dans cette table. Même si tous les champs sont accessibles, il n'est pas nécessaire de renseigner celui de la clé primaire `ID` car nous l'avons configuré pour être auto-incrémenté. La valeur du champ sera ainsi automatiquement renseignée lors de l'insertion de chaque enregistrement.

Pour les besoins de notre atelier, nous allons vous demander de saisir deux noms et prénoms de joueur en vous référant à ceux renseignés dans la figure 13-3. Vous pouvez évidemment saisir les noms de votre choix mais si vous désirez que cela corresponde avec les visuels de nos tests, nous conseillons de saisir ces mêmes informations dans un premier temps.

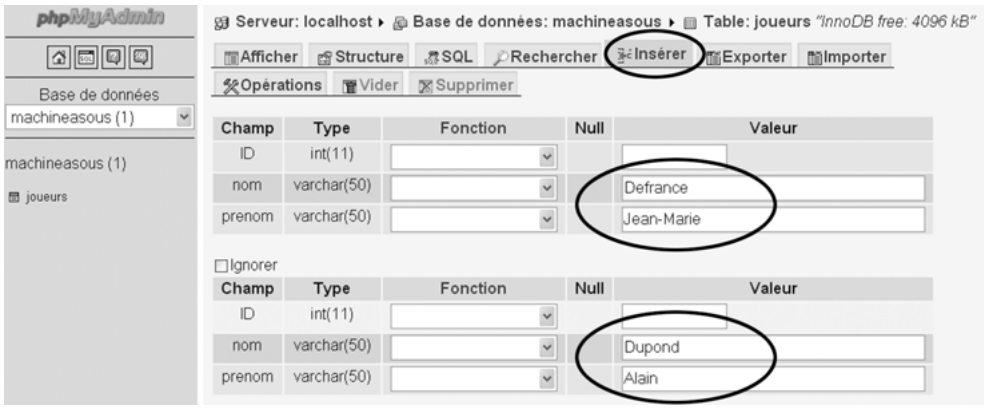


Figure 13-3

Insertion de deux utilisateurs dans la table joueurs

Cliquez cette fois sur le bouton Exécuter pour confirmer l'insertion de ces deux nouveaux utilisateurs. Un message doit alors s'afficher vous indiquant que les enregistrements ont bien été ajoutés à la table (voir code ci-dessous). La syntaxe de la requête SQL doit, quant à elle, s'afficher en dessous.

```
Nombre d'enregistrements insérés : 2
```

Notre base est maintenant prête à être utilisée, si toutefois vous désirez avoir un aperçu de son contenu avant de quitter le gestionnaire, vous pouvez le faire en cliquant sur l'onglet Afficher (voir figure 13-4).

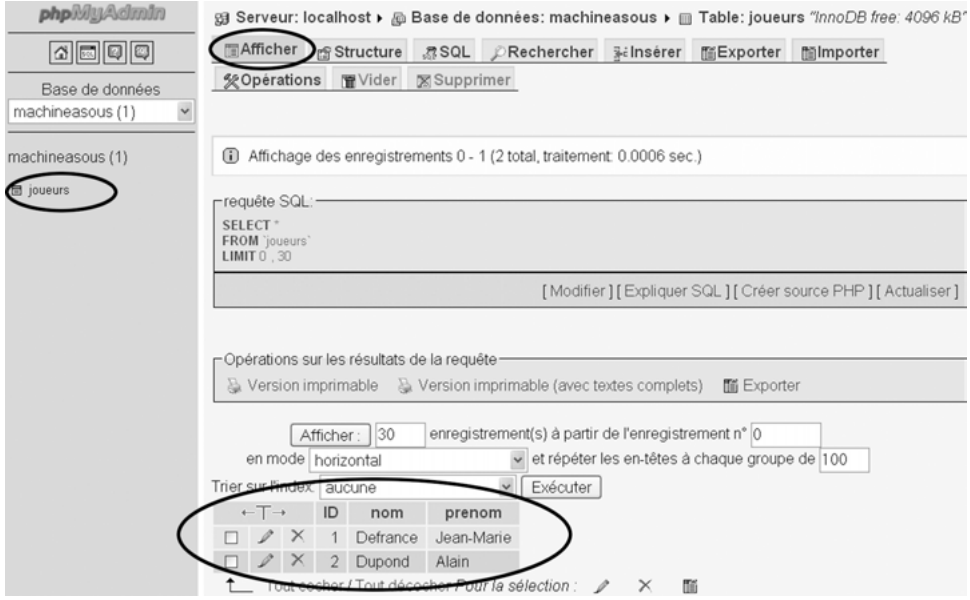


Figure 13-4

Contenu de la table joueurs

Vous pouvez aussi remarquer qu'une icône représentant la nouvelle table est désormais présente dans la partie de gauche du gestionnaire. Celle-ci vous permettra par la suite d'accéder directement à la table `joueurs` (si vous cliquez sur l'icône, vous afficherez le contenu de la table alors que si vous cliquez sur son nom, c'est sa structure qui s'affichera). Cependant, rappelez-vous que quel que soit l'écran que vous visualisez dans phpMyAdmin, vous avez la possibilité de revenir rapidement à une page « tableau de bord » qui rassemble dans le même écran toutes les actions possibles (ajout, suppression, modification du contenu comme de la structure des tables) et cela sur toutes les futures tables de votre base. Pour cela, il suffit de cliquer sur le nom de la base de données situé dans la partie gauche du gestionnaire (voir repère 1 de la figure 13-5).

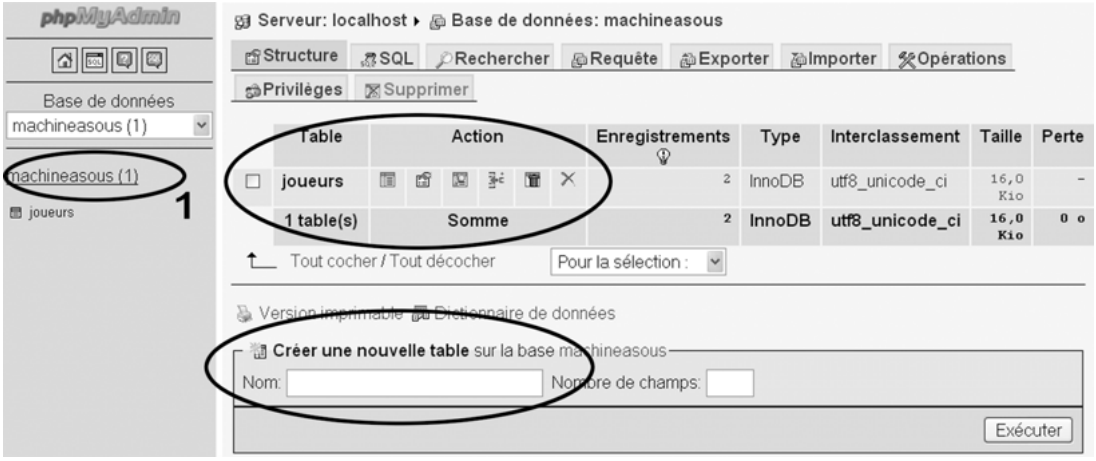


Figure 13-5

En cliquant sur le nom de la base dans la partie gauche de l'écran, vous accédez rapidement au tableau récapitulatif des différentes actions que vous pouvez effectuer sur les différentes tables de la base.

> **Utilisation de phpMyAdmin**
>
> Si par la suite vous désirez faire des modifications de la structure de cette table ou de son contenu, vous pouvez trouver les procédures relatives à ces actions dans le chapitre 22 consacré à SQL de la partie 4 de cet ouvrage.

Passons maintenant aux modifications des fichiers. Pour pouvoir accéder à cette base, il est nécessaire de créer une connexion entre le serveur Web (où se trouvent les fichiers PHP) et le serveur de base de données. En production, un nom d'utilisateur de la base et le mot de passe en rapport sont obligatoires pour s'identifier à la base de données, pour des raisons de sécurité évidentes. Cependant, dans le cadre de nos ateliers, nous allons utiliser le nom d'utilisateur créé par défaut lors de l'installation de Wamp. Cet utilisateur générique se nomme `root` et ne nécessite pas de mot de passe (pour créer un utilisateur spécifique, reportez-vous au chapitre sur le SQL). Une autre information dont nous allons avoir besoin pour créer cette connexion est le nom du domaine sur lequel le serveur de base de données est installé. En général, ce dernier est installé sur la même machine que le serveur Web et il suffit d'indiquer `localhost` pour que le serveur de base de données soit localisé. Cependant, sachez que selon votre hébergeur vous pouvez avoir des organisations différentes et il faut alors utiliser le domaine sur lequel se trouve la base de données pour réaliser une connexion. Ces trois informations vont ainsi nous permettre

de créer un identifiant de connexion qui devra être exploité par la suite dans les différents fichiers PHP pour lesquels nous aurons besoin d'accéder à la base de données. Aussi, est-il judicieux de les stocker dans un fichier indépendant des autres fichiers PHP afin de pouvoir modifier ces paramètres en un seul endroit au cas ou vous devriez transférer votre système sur un autre serveur Web.

Nous allons donc créer un fichier `connexionMysql.php` dans lequel nous allons centraliser ces informations ainsi que la fonction de création de l'identifiant (`$connexionMysql`) et le nom de la base de données (`$base`) qui sont, eux aussi, communs à tous les programmes PHP qui nécessitent une connexion à la base. Le contenu de ce fichier de connexion est semblable au code 13-1 et nous vous invitons à le créer dès maintenant.

Code 13-1 : `connexionMysql.php`, fichier de connexion à la base de données :

```
# Nom du fichier="connexionMysql.php"
$serveur = "localhost";
$base = "machineasous";
$utilisateur = "root";
$motedepasse = "";
$connexionMysql = mysql_connect($serveur, $utilisateur, $motdepasse);
```

Ressources sur des fonctions PHP dédiées à MySQL

Pour obtenir plus d'informations sur les fonctions PHP dédiées à MySQL, vous pouvez vous reporter au tableau 21-18 du chapitre consacré à PHP.

Pour mettre en œuvre cette nouvelle fonctionnalité de vérification du nom du joueur, nous allons créer un nouveau fichier PHP. Celui-ci se nommera `nomVerification.php` et comportera au début les mêmes instructions que le précédent fichier `gainAleatoire.php` destinées au blocage du cache et à la récupération de la variable `nom` envoyée dans l'URL par la requête Ajax (voir code 13-2).

Saisissons à la suite de cette partie les instructions spécifiques à notre application. Nous devons commencer par faire référence au fichier de connexion pour l'inclure dans le programme et disposer ainsi des variables qui y sont déclarées.

```
require_once('connexionMysql.php');
```

Il faut ensuite sélectionner la base sur laquelle nous désirons intervenir à l'aide de la fonction `mysql_select_db()` en rappelant la variable `$base` qui contient son nom (mémorisé précédemment dans le fichier de connexion).

```
mysql_select_db($base);
```

Dans notre exemple, nous désirons savoir si le nom envoyé en paramètre par la requête Ajax (`$nom`) est présent dans le champ `nom` de la table `joueurs` que nous avons précédemment créée. Nous allons donc créer une requête SQL en rapport en y intégrant la variable `$nom` dans la clause `WHERE`. Le résultat de cette dernière est mémorisé dans une variable `$requeteSQL` pour une meilleure lisibilité du code. Elle est ensuite soumise au serveur à l'aide de la fonction `mysql_query()` qui renverra un pointeur en mémoire sur une liste de résultats (`$reponseSQL`), si toutefois le nom en question a été trouvé.

```
$requeteSQL="SELECT ID FROM joueurs WHERE nom='".$nom."'";
$reponseSQL = mysql_query($requeteSQL);
```

Le pointeur `$reponseSQL` n'étant pas exploitable directement, il est nécessaire de transférer les résultats ainsi pointés en mémoire dans un tableau de variables afin de pouvoir

accéder aux différents champs de l'enregistrement correspondant à la requête SQL (dans notre cas, nous n'avons que le champ `ID` dans cet enregistrement car la requête SQL a été configurée de la sorte : `SELECT ID`). Pour réaliser cette transformation, nous allons utiliser une fonction PHP de type `mysql_fetch_xxx()` soit `mysql_fetch_array()` qui a l'avantage de créer un double tableau, à la fois associatif (avec pour clé les noms des champs) et indicé (mieux adapté en cas de parcours par une boucle).

```
$enregistrement=mysql_fetch_array($reponseSQL);
```

La variable `$enregistrement` est donc un tableau dans lequel nous pouvons récupérer les résultats en précisant simplement le nom du champ désiré en tant que clé du tableau associatif. Par exemple, pour récupérer l'identifiant du joueur qui a été reconnu, nous pouvons utiliser l'instruction suivante :

```
$ID=$enregistrement["ID"];
```

Cependant, avant de mémoriser cet identifiant, il faut effectuer un test afin de s'assurer qu'il existe bien un résultat et dans le cas où le joueur n'a pas encore été reconnu, retourner la valeur zéro en guise d'identifiant. Nous allons réaliser cette vérification en exploitant la fonction `mysql_num_rows()` qui est censée contenir le nombre d'enregistrements renvoyés par le serveur de base de données en réponse à la requête SQL. Si celui-ci est supérieur à zéro, nous affectons l'identifiant de l'enregistrement retourné à la variable `$ID`, sinon nous affectons la valeur `0` à cette même variable.

```
if(mysql_num_rows($reponseSQL)>0)
  $ID=$enregistrement["ID"];
else
  $ID=0;
echo $ID ;
```

La création du nouveau fichier PHP est désormais terminée, vous pouvez l'enregistrer. Son contenu doit être semblable à celui du code 13-2.

Code 13-2 : `nomVerification.php`, fichier permettant la vérification du nom du joueur dans la base de données :

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
require_once('connexionMysql.php');
mysql_select_db($base);
$requeteSQL="SELECT ID FROM joueurs WHERE nom='".$nom."'";
$reponseSQL = mysql_query($requeteSQL);
$enregistrement=mysql_fetch_array($reponseSQL);
if(mysql_num_rows($reponseSQL)>0)
  $ID=$enregistrement["ID"];
else
  $ID=0;
echo $ID ;
```

Passons maintenant côté client pour créer le nouveau moteur Ajax qui va solliciter ce fichier `nomVerification.php`. Pour cela, ouvrez le fichier `fonctionsMachine.js` et localisez la fonction `detecterNavigateur()`. Pour mémoire, cette fonction (initialement créée pour identifier si le navigateur supporte les requêtes Ajax dès l'ouverture de la page) est appelée dès le chargement de la page dans le navigateur. Nous allons donc l'exploiter pour y

insérer la déclaration du gestionnaire d'événement qui va prendre en charge la détection de la saisie des caractères dans le champ `nom`. Pour cela, nous allons placer à la fin de cette fonction un gestionnaire d'événement `onkeyup` appliqué au champ de saisie (l'élément `nom`) afin de lui associer l'appel de la fonction `nomVerifier` à chaque fois que cet événement se produira.

```
document.getElementById("nom").onkeyup=nomVerifier;
```

Dans cette fonction, nous allons devoir configurer un moteur Ajax afin de créer un nouvel objet XHR et envoyer la requête Ajax au fichier `nomVerification.php`, avec comme paramètre le nom en cours de saisie à chaque fois qu'un nouveau caractère est ajouté. La structure de ce script est semblable à celle de la fonction `jouer()` qui exerce la même fonction dans le premier moteur Ajax déjà en place. Nous allons donc copier son contenu dans notre nouvelle fonction `nomVerifier()` puis lancer une action Rechercher/Remplacer (localiser au contenu de la fonction) pour remplacer le nom de l'objet `objetXHR` par `objetXHR2`.

Simplification de la fonction jouer()

Avec l'ajout de la nouvelle fonctionnalité de vérification bloquante lors de la saisie du nom, le test de contrôle de la présence d'un nom dans le champ placé en début de la fonction `jouer()` n'est désormais plus nécessaire. Vous pouvez donc simplifier cette fonction en supprimant les instructions correspondant à cette fonctionnalité devenue inutile.

Nettoyez ensuite ce code des instructions qui ne sont plus nécessaires dans le contexte de ce nouveau moteur (comme le contrôle de la saisie du nom ou la gestion de l'animation du chargeur, par exemple). Modifiez ensuite le nom de la fonction de rappel en le remplaçant par `afficherReponse` ainsi que le nom du fichier PHP vers lequel sera envoyée la requête en le remplaçant par `nomVerification.php`. Une fois les mises à jour effectuées, la nouvelle fonction devrait être semblable au code 13-3 ci-après.

Code 13-3 : fonction `nomVerifier()` du fichier `fonctionsMachine.js` :

```
function nomVerifier()
 objetXHR2 = creationXHR();
 var temps = new Date().getTime();
 var parametres = "nom="+ codeContenu("nom") +
                  "&anticache="+temps ;
 objetXHR2.open("get","nomVerification.php?"+parametres, true);
 objetXHR2.onreadystatechange = afficherReponse;
 document.getElementById("button").disabled= true;
 objetXHR2.send(null); // Envoi de la requête
}
```

Il nous reste maintenant à créer la fonction de rappel liée à ce second moteur Ajax. Cette fois nous allons copier la fonction de rappel `actualiserPage()` afin de la prendre comme base de départ pour la nouvelle fonction de rappel de ce second moteur Ajax. Dès que la nouvelle fonction est copiée, renommez-le `afficherReponse()`. Le but de cette fonction étant d'informer l'utilisateur qu'il est reconnu et de débloquer le bouton JOUER dès qu'il est identifié, nous allons tester la valeur de l'identifiant (`ID`) renvoyée par le serveur (et récupérée dans la variable `nouveauResultat`) pour conditionner ces actions. Le bloc ainsi défini doit alors contenir les instructions ci-dessous :

```
if(nouveauResultat!=0) {
  remplacerContenu("message", "Joueur identifié");
  document.getElementById("button").disabled= false;
}
```

Dans le cas contraire, un message différent doit être affiché afin d'indiquer que le nom saisi est inconnu et le bouton JOUER doit être bloqué. Pour réaliser ces actions, le bloc alternatif contient les instructions suivantes :

```
else {
  remplacerContenu("message", "Joueur inconnu");
  document.getElementById("button").disabled= true;
}
```

Une fois les modifications de cette fonction de rappel réalisées, celle-ci devrait être semblable au code 13-4 (auxquelles vont s'ajouter des instructions secondaires sur la mise en forme du texte ou son affichage).

Code 13-4 : fonction `afficherReponse()` du fichier `fonctionsMachine.js` :

```
function afficherReponse() {
if (objetXHR2.readyState == 4) {
 if (objetXHR2.status == 200) {
  var nouveauResultat = objetXHR2.responseText;
    if(nouveauResultat!=0) {
     document.getElementById("message").style.visibility="visible";
     remplacerContenu("message", "Joueur identifié");
     document.getElementById("message").style.color="green";
     document.getElementById("button").disabled= false;
    }else{
     document.getElementById("message").style.visibility="visible";
     remplacerContenu("message", "Joueur inconnu");
     document.getElementById("message").style.color="red";
     document.getElementById("button").disabled= true;
    }
 }
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Améliorations possibles du système

Dans notre exemple, la table ne contient que deux enregistrements et nous sommes en local. Cependant, imaginez que votre club de jeux remporte un franc succès et que vous deviez gérer 1 000 voire 10 0000 joueurs … Le temps de recherche dans la base de données serait alors plus long et pourrait freiner considérablement la réactivité du système, notamment au début de la frappe pour les noms de quelques lettres qui n'ont aucune chance de correspondre à un nom possible de joueur. Pour améliorer la réactivité du système dans ce contexte, nous pourrions alors mettre en place un test côté client qui vérifierait que le nom à soumettre est supérieur à 3 caractères par exemple, sans quoi la requête Ajax ne serait pas émise. Vous trouverez ci-dessous un exemple de code qui permettrait d'ajouter cette fonctionnalité en début de la fonction `verifNom()`.

Code 13-5 : option pour la fonction `verifNom()` :

```
function verifierNom() {
 var taille=document.getElementById("nom").value.length;
 if(taille<4) {
   document.getElementById("message").style.visibility="visible";
   remplacerContenu("message", "nombre de caractères insuffisant");
```

```
      document.getElementById("message").style.color="red";
      return false;
    }
…
}
```

Un autre problème qui pourrait apparaître si la base devenait trop volumineuse, entraînant du même coup un temps de réponse du serveur plus long, serait de saturer le système avec un grand nombre de requêtes si l'utilisateur saisissait rapidement tous les caractères de son nom. Dans ce cas, la solution consiste à mettre en place un système de déclenchement des requêtes de vérification, non plus à chaque caractère saisi, mais toutes les secondes par exemple. Pour mettre en œuvre ce système, une solution simple consiste par exemple à remplacer le gestionnaire d'événement actuel `onkeyup` du champ `nom` par un `onfocus` dans la fonction `detecterNavigateur()`. De cette manière, il serait possible de déclencher le contrôle périodique dès que l'utilisateur clique dans le champ, puis d'ajouter un temporisateur à la fin de la fonction `verifierNom()` qui rappelle la fonction `verifierNom()` toutes les secondes, par exemple.

Code 13-6 : solution alternative pour le gestionnaire d'événement :

```
function detecterNavigateur() {
…
document.getElementById("nom").onfocus=verifierNom;
}
```

Code 13-7 : insertion d'un temporisateur pour déclencher la fonction toutes les secondes :

```
function verifierNom() {
…
setTimeout("verifierNom()",1000); // Timer de 1 seconde
}
```

Enfin, si nous partons du principe qu'il s'agit de noms de joueur, ceux-ci ne devraient pas contenir de chiffre par exemple. Il serait alors judicieux d'ajouter un système de contrôle des caractères saisis côté client. De même, il serait aussi intéressant d'en profiter pour filtrer toutes les autres touches du clavier qui ne doivent pas être utilisées comme les flèches de déplacement ou les touches Suppr ou Insert, par exemple. Dans ce cas, la détection du caractère pourrait être mise en place au début de la fonction comme l'illustre l'exemple de code suivant qui permet de filtrer les touches numériques du clavier.

Code 13-8 : exemple de filtre bloquant les touches numériques du clavier :

```
function verifierNom(event) {
 event = window.event||event;
 // Récupération du code de la touche
 var codeTouche= event.keyCode;
 if(codeTouche>=48 && codeTouche<=57)
 {
 // Mettre ici l'action si le caractère est un chiffre
 return false;
 }
 // Pour information, le caractère de la touche peut
 // être récupéré avec l'instruction suivante :
 var touche = String.fromCharCode(codeTouche);
…
}
```

Test du système

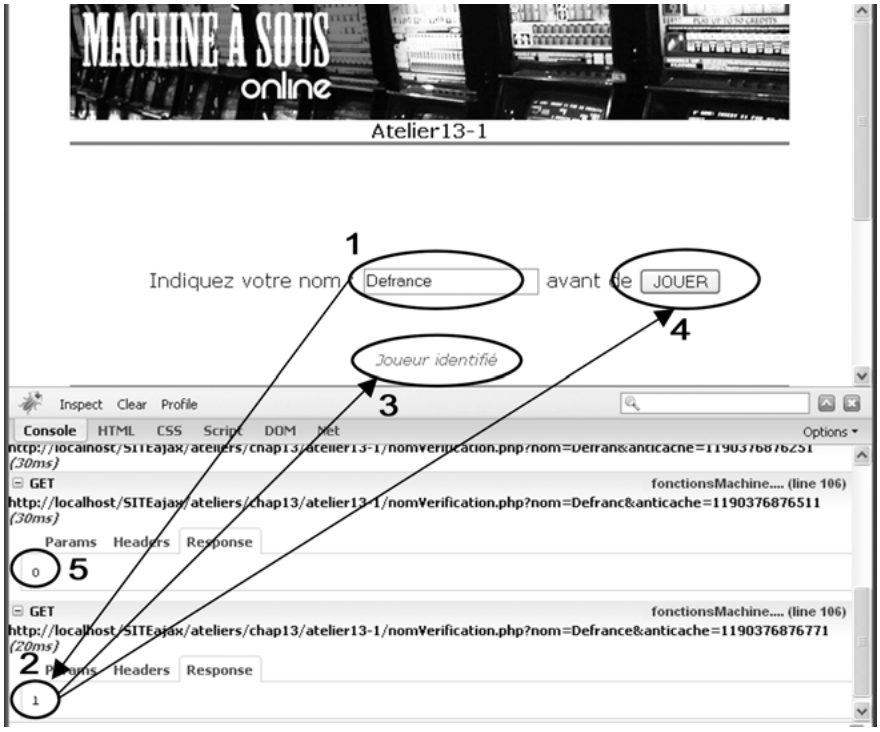
Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Pour les tests, nous vous invitons à saisir l'un des deux noms déjà présents dans la table `joueurs` (Defrance ou Dupond). Dès l'insertion de la première lettre dans le champ de saisie, un message « joueur inconnu » doit s'afficher dans la zone de message située en dessous du champ de saisie et le bouton JOUER doit devenir grisé afin d'indiquer qu'il est bloqué. Continuez cependant à saisir le nom, jusqu'à sa dernière lettre. Le message doit alors changer et indiquer « joueur identifié » et le bouton JOUER doit de nouveau redevenir actif.

Vous pouvez ensuite appuyer sur le bouton JOUER et utiliser le système comme dans les ateliers précédents. Le nom du joueur et le montant de son nouveau gain doivent alors apparaître à l'écran.

Pour visualiser les données renvoyées par le serveur lors de la réponse de chacune des requêtes générées par un nouveau caractère, nous vous conseillons d'activer Firebug et de cliquer sur l'onglet Console. Si nous prenons comme exemple le moment où vous venez de saisir le dernier caractère d'un des deux noms de la base, vous devriez voir en bas de la fenêtre les deux dernières requêtes avant la validation du nom. Cliquez sur le signe + situé devant les noms de ces deux derniers fichiers pour les dérouler et cliquez ensuite sur leur onglet Response. Vous devriez voir dans cette fenêtre la valeur `0` pour l'avant dernière requête (le nom du joueur n'étant pas encore identifié, voir repère 5 de la figure 13-6) et la valeur `1` pour la dernière s'il s'agit de Defrance (sinon c'est la valeur `2` pour Dupond puisqu'il s'agit de leur clé primaire, voir repère 2 de la figure 13-6). Vous pouvez aussi cliquer sur les onglets Params de ces deux requêtes pour constater que dans la dernière, le nom complet a été envoyé (`Defrance`, par exemple) alors que dans l'avant dernière, il manquait encore la dernière lettre (`Defranc`, par exemple).

Figure 13-6

Test du système de vérification du nom du joueur en cours de saisie



Atelier 13-2 : insertion dans la base de données issues d'un formulaire

Composition du système

Dans ce deuxième atelier, nous allons adapter le premier moteur Ajax et son script serveur afin de pouvoir mémoriser les différents gains des joueurs dans une nouvelle table de la base. Pour vérifier que l'insertion a bien été effectuée, nous allons également ajouter un troisième paramètre à la réponse retournée par le serveur afin de permettre l'affichage d'un message d'erreur à la place du montant du gain habituel si toutefois un problème survenait.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure est identique à celui de l'atelier 13-1 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est identique à celle de l'atelier 13-1 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 13-1, modifiée pour pouvoir gérer la commande d'insertion SQL dans la base de données ;
- d'un fichier serveur PHP pour la vérification du nom (`nomVerification.php`), identique à celui de l'atelier 13-1 ;
- d'un fichier PHP de connexion à la base de données (`connexionMysql.php`), identique à celui de l'atelier 13-1 ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 13-1 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système est semblable à celui de l'atelier 13-1 que nous venons de réaliser à ceci près que dans cet atelier, les gains de chaque jeu sont enregistrés dans la base de données. En cas de problème avec le serveur de base de données, un message d'erreur s'affichera alors à l'écran.

Conception du système

Ouvrez la page HTML de l'atelier 13-1 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap13/atelier13-2/`. Copiez ensuite les autres fichiers de l'atelier 13-1 dans ce nouveau dossier.

Pour mémoriser les montants des gains des différents joueurs nous allons devoir créer une seconde table `gains` dans la base `machineasous`. Celle-ci est liée à la première (`joueurs`) par un champ particulier que nous nommons « clé étrangère » dans le dialecte des bases de données, mais qui n'est, ni plus ni moins, que la copie de la clé primaire du joueur auquel est attribué le gain de l'enregistrement (ceci afin d'assurer la liaison entre l'enregistrement du gain et celui du joueur concerné). Ainsi, si nous réalisons une structure minimaliste de cette table, elle peut ne comporter que quatre champs : sa clé primaire

(`ID` : champ obligatoire pour toutes les tables, auto-incrémenté dans notre cas) ; le montant du gain à enregistrer (`montant`) ; la date et l'heure (`date`) de l'enregistrement et la clé étrangère (`joueursID` : copie du champ `ID` de la table `joueurs`) qui relie l'enregistrement à celui d'un joueur particulier de la table `joueurs`.

Pour créer cette nouvelle table, nous allons de nouveau utiliser le gestionnaire phpMyAdmin (cliquez sur l'entrée du même nom dans le manager de Wamp pour ouvrir le gestionnaire). Une fois ouvert, nous sélectionnons cette fois la base `machineasous` que nous avons créé précédemment dans le menu de gauche. Le nom de la table `joueurs` et ses différents boutons donnant accès aux actions du gestionnaire sur cette table doivent apparaître à droite. Saisissez le nom `gains` dans le champ Créer une nouvelle table, et `4` pour le nombre de champs puis validez en cliquant sur le bouton Créer.

Nous nous retrouvons de nouveau avec le même genre de formulaire que pour la création de la table `joueurs` (revoir si besoin l'atelier précédent pour plus d'explications sur le rôle de chacun de ces champs). Saisissez les quatre noms et leur éventuelles valeurs associées dans les deux premières colonnes de ce formulaire en vous référant aux informations de la figure 13-7. À noter que dans le cas de cette table `gains`, les champs sont de type entier (`INT`) sauf pour la date (qui est de type `DATETIME`). En effet, les clés étrangère et primaire doivent être de même type (le champ `ID` de la table `joueurs` étant de type `INT`, il est donc logique que la clé étrangère en rapport soit, elle aussi, de type `INT`). De même, le montant correspondant à une valeur entière comprise entre 0 et 100 (sans décimale), son champ devra, lui aussi, être de type `INT`.

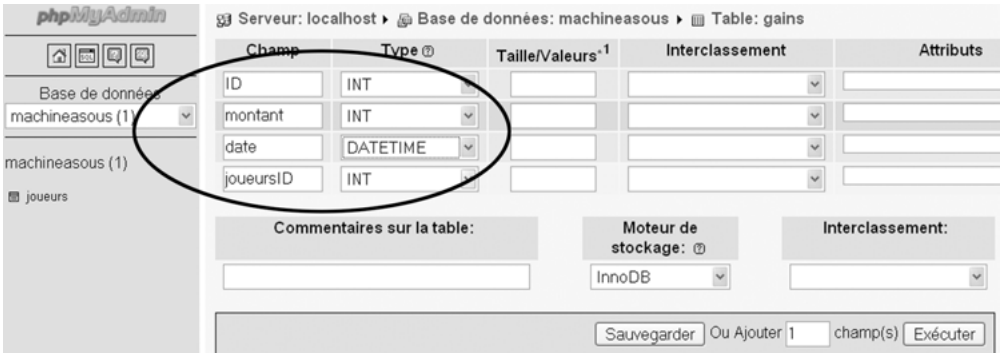


Figure 13-7

Formulaire de création de la table gains (partie de gauche)

Déplacez-vous maintenant dans la partie de droite du formulaire à l'aide du curseur horizontal. Dans la colonne Extra, sélectionnez l'option `auto_increment` (comme pour la précédente table `joueurs`) pour la clé primaire (première ligne) et cochez le bouton radio Primaire sur la même ligne pour indiquer que `ID` est la clé primaire de la table (voir figure 13-8). Validez votre configuration en cliquant sur le bouton Sauvegarder pour créer la nouvelle table `gains` correspondante.

Passons maintenant à l'éditeur Dreamweaver afin de modifier les fichiers pour répondre aux nouvelles fonctionnalités de cet atelier.

Pour lier l'enregistrement du gain avec celui du joueur correspondant, nous allons devoir disposer de la clé primaire du joueur. Pour mémoire, celle-ci est renvoyée dans la réponse du second moteur Ajax, dès que le nom du joueur est identifié dans la table `joueurs`.

Nous allons donc récupérer cette information pour l'exporter dans le processus d'insertion en l'envoyant en paramètre supplémentaire dans la requête du premier moteur Ajax (celui qui est déclenché par le bouton JOUER).

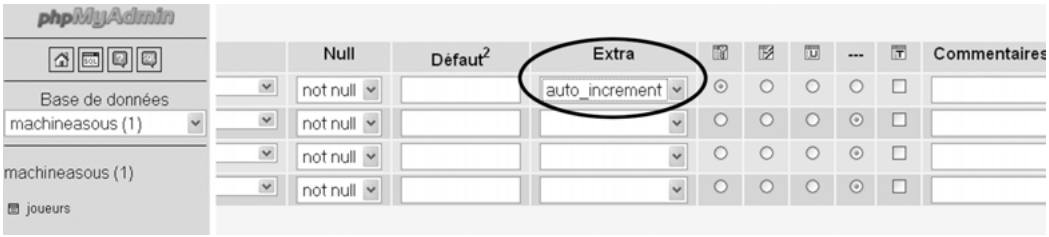


Figure 13-8

Formulaire de création de la table gains (partie de droite)

Commencez par ouvrir le fichier `fonctionsMachine.js` puis localisez la première instruction après le test du statut dans la fonction `afficherReponse()` du deuxième moteur Ajax. Pour le moment, la variable `nouveauResultat` qui réceptionne l'identifiant du joueur (dès qu'il est reconnu) est déclarée en variable locale (avec `var`) et son utilisation est donc limitée au bloc de la fonction. Nous allons modifier cela en supprimant le mot-clé `var` afin de rendre la variable globale et ainsi pouvoir en disposer dans le reste du programme. La nouvelle instruction après modification doit être semblable à la ligne suivante :

```
nouveauResultat = objetXHR2.responseText ;
```

Nous allons maintenant récupérer cette information dans la fonction `jouer()` pour l'ajouter dans les paramètres d'URL déjà présents pour la requête Ajax. Pour cela, nous allons modifier l'instruction qui construit la variable `parametres` afin d'ajouter la variable `nouveauResultat` (désormais disponible dans cette fonction car elle est maintenant globale) dans la chaîne des paramètres d'URL précédée de `&ID=` pour être conforme au format d'URL comme indiqué dans l'instruction ci-dessous.

```
var parametres = "nom="+ codeContenu("nom") + "&ID="+ nouveauResultat
➥+ "&anticache="+temps ;
```

Ainsi, la requête Ajax du premier moteur envoie désormais l'identifiant de l'enregistrement du joueur en plus de son nom. Nous n'avons cependant pas terminé nos modifications côté client car nous désirons aussi afficher un message d'information à la place du message habituel indiquant le montant du gain dans le cas où des problèmes de connexion avec la base de données surviendraient, empêchant l'enregistrement du gain.

Nous reviendrons sur ce fichier JavaScript par la suite. Passons maintenant au fichier PHP afin de gérer la requête SQL d'insertion qui doit être déclenchée à la réception de la requête Ajax initiée par une action du joueur sur le bouton JOUER.

Ouvrez pour cela le fichier `gainAleatoire.php` et commencez par ajouter un script de réception HTTP de la variable `ID` semblable à celui que nous avons déjà mis en place pour la variable `nom`.

```
if(isset($_REQUEST['ID'])) $ID=$_REQUEST['ID'];
else $ID=0;
```

Ajoutez l'instruction d'inclusion du fichier de connexion Mysql après l'instruction du calcul du gain de manière à disposer de ses informations dans le fichier.

```
require_once('connexionMysql.php');
```

Sélectionnez ensuite la base `machineasous` (mémorisée dans la variable `$base`) à l'aide de l'instruction `mysql_select_db()`.

```
mysql_select_db($base);
```

Passons maintenant à la création de la commande SQL qui va permettre d'ajouter le montant du gain dans la table `gains` (revoir si besoin les syntaxes SQL dans le chapitre 22 de la partie 4 de cet ouvrage).

```
 $commandeSQL="INSERT INTO gains SET montant='".$gain."', joueursID='".$ID."' ";
```

Dans cette commande, nous allons renseigner le montant du gain (calculé aléatoirement comme d'habitude) et la clé primaire du joueur auquel ce gain doit être associé (mémorisé dans le champ de la clé étrangère `joueursID` de la table `gains`). Cette dernière information est envoyée en paramètre dans l'URL de la requête Ajax suite aux modifications que nous avons réalisées précédemment. Une fois que la commande proprement dite est mémorisée dans la variable `$commandeSQL`, il ne reste plus qu'à la soumettre au serveur à l'aide de la fonction `mysql_query()`.

```
$reponseSQL = mysql_query($commandeSQL);
```

La réponse retournée par le serveur de base de données est égale à `true` (soit `1`) si l'insertion a été correctement effectuée et à `false` dans le cas contraire. Il suffit donc ensuite de récupérer cette variable pour l'inclure en troisième paramètre dans la réponse retournée au navigateur.

```
$resultat=$nom.':'.$gain.':'.$reponseSQL;
```

Les modifications de ce fichier PHP sont terminées, vous pouvez l'enregistrer. Il doit maintenant être semblable au code 13-9.

Code 13-9 : `gainAleatoire.php`, après modifications :

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
sleep(2);
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
if(isset($_REQUEST['ID'])) $ID=$_REQUEST['ID'];
else $ID=0;
$gain = rand(0,100);
require_once('connexionMysql.php');
mysql_select_db($base);
$commandeSQL="INSERT INTO gains SET montant='".$gain."', joueursID='".$ID."' ";
$reponseSQL = mysql_query($commandeSQL);
$resultat=$nom.':'.$gain.':'.$reponseSQL;
echo $resultat ;
```

Revenons maintenant côté client pour terminer nos modifications. En effet, il nous reste à exploiter ce troisième paramètre pour conditionner l'affichage du montant du gain habituel si sa valeur est égale à 1 ou afficher un message indiquant un problème dans le cas contraire.

La récupération des valeurs de la réponse serveur s'effectue de la même manière que dans les ateliers précédents hormis le fait que nous avons maintenant trois valeurs à récupérer au lieu de deux. À noter que la conversion de la chaîne du résultat (au format `nom:gain:indicateur`) en un tableau de variables s'effectue aussi avec la méthode `split(":")`, comme dans les ateliers précédents.

Une fois que l'indicateur de réussite de l'insertion SQL est accessible par une variable du tableau (`nouveauResultat[2]`), nous allons utiliser une structure de test `if()` conditionnée par l'égalité de cet indicateur avec la valeur `1` (valeur correspondant à une insertion effectuée sans problème) pour afficher le résultat du jeu. Dans le cas contraire, un message d'erreur est affiché à la place du résultat (voir structure `else` dans le code 13-10).

```
var nouveauResultat = objetXHR.responseText.split(":");
if(nouveauResultat[2]==1) {
… } else {
… }
```

À noter que nous avons ajouté juste après le test une instruction de réinitialisation du contenu de l'élément `info` avec la même structure qu'au chargement initial de la page (voir code 13-10) afin que le système puisse de nouveau fonctionner normalement suite à une erreur SQL. En effet, une telle erreur aurait pour incidence de remplacer tout le contenu de l'élément `info` par le message d'erreur et empêcherait ainsi un fonctionnement normal du système une fois le problème résolu.

Une fois modifiée, la fonction `actualiserPage()` doit être semblable au code 13-10.

Code 13-10 : fonction `actualiserPage()`, après modifications :

```
function actualiserPage() {
if (objetXHR.readyState == 4) {
 if (objetXHR.status == 200) {
  var nouveauResultat = objetXHR.responseText.split(":");
  if(nouveauResultat[2]==1) {
     var elementInfo = document.getElementById("info");
     elementInfo.innerHTML='Bravo, M <span id="gagnant"></span> vous avez gagné
     ➥<span id="resultat"></span> Euros';
     elementInfo.style.color="black";
     // Actualisation du résultat
     remplacerContenu("resultat", decodeURI(nouveauResultat[1]));
     // Actualisation du nom
     remplacerContenu("gagnant", decodeURI(nouveauResultat[0]));
     // Affichage de la zone info
  }else{
     var elementInfo = document.getElementById("info");
     elementInfo.innerHTML="Problème technique : gains non enregistrés" ;
     elementInfo.style.color="red";
    }
   document.getElementById("info").style.visibility="visible";
   document.getElementById("button").disabled= false;
   document.getElementById("charge").style.visibility="hidden";
 }
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Comme dans l'atelier précédent, saisissez un des deux noms enregistrés dans la table `joueurs` (Defrance ou Dupond). Lorsque le message « joueur identifié » apparaît, cliquez sur le bouton JOUER. Le message habituel indiquant le montant du gain doit alors s'afficher s'il n'y a pas de problème de connexion avec le serveur de base de données et le gain du jeu doit être ajouté dans la table `gains`.

Si vous le désirez, vous pouvez simuler un problème de connexion en changeant le nom de l'utilisateur ou celui de la base dans le fichier `connexionMysql.php`. Si vous renouvelez vos tests, un message d'erreur doit s'afficher à la place du montant du gain.

Pour visualiser les données renvoyées par le serveur lors de la réponse, nous vous conseillons d'activer Firebug et de cliquer sur l'onglet Console. Cliquez sur le signe + qui précède les deux derniers noms de fichier pour les dérouler. Le dernier enregistrement (voir repère 3 de la figure 13-9) correspond à l'appel du script serveur `gainAleatoire.php` et on peut voir dans la fenêtre Response les trois paramètres renvoyés par le serveur (si l'insertion dans la base s'est bien passée, le troisième paramètre doit être égal à 1, voir repère 4 de la figure 13-9). L'avant dernier enregistrement, quant à lui, correspond au dernier appel du fichier `nomVerification.php` (voir repère 1 de la figure 13-9). La réponse du serveur de cette requête, indiquant que le nom du joueur a été identifié dans la base, doit être différente de 0 (soit 1 pour Defrance et 2 pour Dupond, voir repère 2 de la figure 13-9).

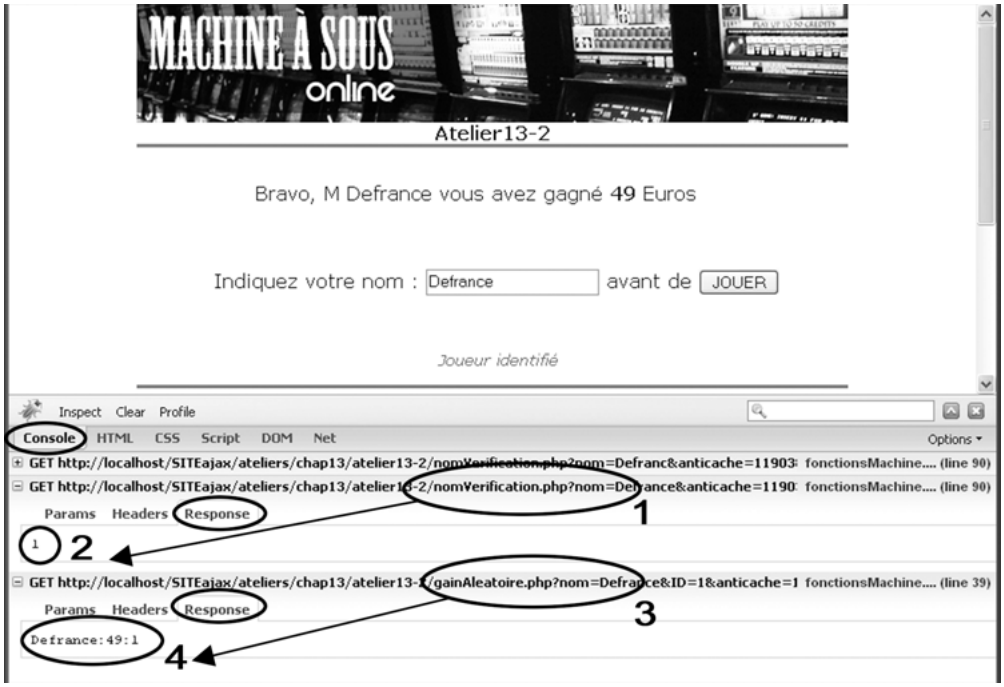


Figure 13-9

Test du système d'insertion des gains dans la base

Enfin, si vous désirez vérifier que les bonnes valeurs de gain ont bien été attribuées au bon joueur, vous devez à nouveau utiliser phpMyAdmin pour afficher le contenu de la table `gains` (voir figure 13-10).

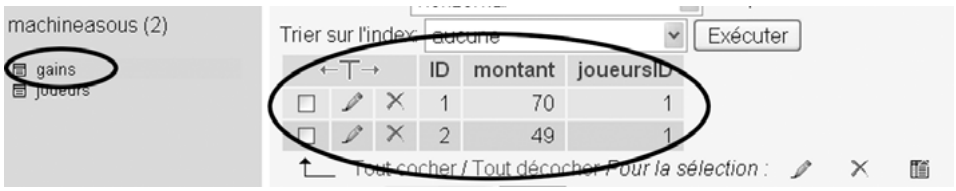


Figure 13-10

Table gains après deux insertions pour le joueur ayant la clé primaire 1

Atelier 13-3 : récupération d'une liste de données dans la base de données

Composition du système

Pour illustrer la récupération d'une liste d'enregistrements par une application Ajax, nous vous proposons maintenant de développer un système qui affiche automatiquement l'historique des gains d'un joueur dans un tableau HTML dès que le nom du joueur est identifié.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure de base avant les modifications est semblable à celle de l'atelier 13-2 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est identique à celle de l'atelier 13-2 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier 13-2 ;
- d'un fichier serveur PHP pour la vérification du nom (`nomVerification.php`), identique à celui de l'atelier 13-2 ;
- d'un nouveau fichier serveur PHP créé dans cet atelier (`gainListe.php`) qui renvoie la liste des gains du joueur ;
- d'un fichier PHP de connexion à la base de données (`connexionMysql.php`), identique à celui de l'atelier 13-2 ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 13-2 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système sera semblable à celui de l'atelier 13-2 que nous venons de réaliser à l'exception près que dans cet atelier, un tableau HTML contenant l'historique des gains, ainsi que la date et l'heure d'enregistrement, s'affichera en bas de l'écran.

Pour mettre en œuvre cette nouvelle fonctionnalité, nous allons créer un troisième moteur Ajax qui va appeler un fichier PHP spécifique qui sollicitera à son tour la base de données avec une requête SQL pour que le navigateur puisse récupérer la liste des enregistrements désirés au format JSON. Une fois les données disponibles côté client, nous allons utiliser les méthodes DOM pour créer à la volée un tableau HTML intégrant les informations récupérées dans le document JSON.

Conception du système

Ouvrez la page HTML de l'atelier 13-2 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap13/atelier13-3/`. Copiez ensuite les autres fichiers de l'atelier 13-2 dans ce nouveau dossier.

Commencez par ouvrir le fichier `index.html` afin d'ajouter la structure vide du tableau qui réceptionnera par la suite les enregistrements des gains et leur date. Pour cela, ajoutez la structure suivante en bas de la page en prenant soin de configurer un identifiant nommé `tableListe` dans la balise `<tbody>` de cette structure.

Code 13-11 : structure du tableau de l'historique des gains du joueur :

```
<table width="400" border="1" align="center" cellspacing="0" >
 <tbody id="tableListe">
    <tr>
     <td>Pas encore de gain</td>
     <td>?</td>
    </tr>
 </tbody>
</table>
```

Balises thead, tbody et tfoot

Pour plus dinformations sur ces balises, reportez-vous à l'encadré qui leur est consacré dans l'atelier 12-6.

Profitez que le fichier `index.html` soit ouvert pour y ajouter une balise `<script>` en référence à la bibliothèque `json.js` que nous allons utiliser par la suite dans `fonctionsMachine.js`.

```
<script type="text/javascript" src="json.js"></script>
```

Ouvrez ensuite le fichier `fonctionsMachine.js`. Le déclenchement du moteur Ajax que nous allons créer étant initié par l'identification du nom du joueur, nous allons ajouter un appel à la future fonction du nouveau moteur (`demandeGains()`) au début de la fonction de rappel du moteur qui gère actuellement la vérification du nom du joueur, soit `afficheReponse()` (voir code 13-12).

Code 13-12 : ajout de l'appel de la fonction `demandeGains()` dans la fonction `afficherReponse()` :

```
function afficherReponse() {
if (objetXHR2.readyState == 4) {
```

```
 if (objetXHR2.status == 200) {
  nouveauResultat = objetXHR2.responseText;
   if(nouveauResultat!=0) {
      demandeGains();
…
```

Pour que le tableau de l'historique soit actualisé après chaque jeu du joueur, nous devons aussi ajouter un appel à cette même fonction `demandeGains()` dans la fonction de rappel `actualiserPage()` du premier moteur (celui qui gère l'affichage du résultat du jeu à l'écran). Le début de cette fonction, une fois modifiée, doit être semblable au code 13-13.

Code 13-13 : ajout de l'appel de la fonction `demandeGains()` dans la fonction `actualiserPage()` :

```
function actualiserPage() {
if (objetXHR.readyState == 4) {
 if (objetXHR.status == 200) {
  var nouveauResultat = objetXHR.responseText.split(":");
  if(nouveauResultat[2]==1) {
    demandeGains();
…
```

Pour éviter d'avoir à réécrire tout le code du nouveau moteur, nous allons prendre comme base de départ les instructions des deux fonctions du second moteur Ajax (copiez l'intégralité de ces deux fonctions et collez le tout à la fin du fichier).

Renommez la première fonction de ce nouveau moteur en `demandeGains` de manière à ce que cela corresponde à l'appel déjà placé dans les fonctions `afficheReponse()` et `actualiserPage()` (voir code 13-12 et 13-13).

```
function demandeGains() {
```

Utilisez ensuite la fonctionnalité Rechercher/Remplacer de Dreamweaver pour modifier le nom de l'objet XHR dans les deux fonctions de ce nouveau moteur (mettre `objetXHR3` à la place de `objetXHR2`, voir codes 13-14 et 13-17).

Afin de pouvoir sélectionner les gains du joueur, les paramètres de la future requête doivent contenir son identifiant qui a été préalablement sauvegardé dans la variable `nouveauResultat` lors de la vérification de son nom. Il convient donc de modifier la construction de la variable `parametre3` correspondante.

```
var parametres3 = "ID="+ nouveauResultat +
                  "&anticache="+temps ;
```

Le deuxième argument de la méthode `open()` doit aussi être actualisé afin de cibler cette fois le fichier `gainListe.php` (fichier que nous allons créer par la suite).

```
objetXHR3.open("get","gainListe.php?"+parametres3, true);
```

De même, le nom de la fonction de rappel de ce troisième moteur doit aussi être changé. Définissons par exemple `afficheGains` comme nom de cette future fonction de rappel.

```
objetXHR3.onreadystatechange = afficheGains;
```

La méthode `send()` reste, quant à elle, inchangée. Une fois que toutes les modifications sont effectuées, la fonction `demandeGains()` doit être semblable au code 13-14.

Code 13-14 :

```
function demandeGains() {
 objetXHR3 = creationXHR();
 var temps = new Date().getTime();
 var parametres3 = "ID="+ nouveauResultat +
         "&anticache="+temps ;
 objetXHR3.open("get","gainListe.php?"+parametres3, true);
 objetXHR3.onreadystatechange = afficheGains;
 objetXHR3.send(null);
}
```

Avant de développer la fonction de rappel `afficheGains` qui va de pair avec cette première fonction, nous allons créer le fichier PHP `gainListe.php` qui va réceptionner les paramètres dans la requête Ajax et qui va renvoyer la réponse au format JSON après avoir interrogé la base de données. Nous reviendrons ensuite sur le fichier client pour terminer nos modifications.

Ouvrez le fichier `nomVerification.php` qui nous servira de base pour élaborer le nouveau fichier PHP. Enregistrez-le sous son nouveau nom, soit `gainListe.php`.

Commencez par modifier l'instruction de récupération HTTP de la variable `nom` en la remplaçant par `ID` (`ID` étant le paramètre envoyé par le nouveau moteur Ajax que nous venons de mettre en place).

```
if(isset($_REQUEST['ID'])) $ID=$_REQUEST['ID'];
else $ID=0;
```

Changez ensuite la clause `WHERE` de la requête SQL afin de l'adapter à ce nouveau paramètre `ID` (pour sélectionner l'enregistrement par rapport à cette clé étrangère). De même, modifiez le nom de la table dans laquelle est réalisée la recherche en remplaçant la table actuelle par `gains`.

```
$requeteSQL= "SELECT montant, date FROM gains WHERE joueursID='".$ID."' ORDER BY ID DESC ";
```

Cette requête retourne ainsi les deux colonnes `montant` et `date` pour chaque enregistrement de la table `gains` dont la clé étrangère `joueursID` est égale à la clé primaire du joueur qui a été préalablement identifié par le système de vérification du nom côté client.

L'objectif du programme qui suit est de convertir au format JSON la série d'enregistrements retournée par le serveur de base de données à la réception de la requête SQL. Après la conversion, nous devrions avoir un document JSON semblable à l'exemple suivant (voir code 13-15) :

Code 13-15 : exemple de réponse JSON générée par le programme (correspondant à 3 enregistrements) :

```
{ "gains" : [
   { "montant":"37","date":"2007-09-26 02:01:17" }
  ,{ "montant":"70","date":"2007-09-26 01:59:52" }
  ,{ "montant":"27","date":"2007-09-26 01:59:43" }
] }
```

Pour faciliter la compréhension de ce programme (voir code 13-16), nous avons mis en gras les informations qui doivent être intégrées dans le document JSON renvoyé au navigateur.

Code 13-16 : programme de conversion des jeux d'enregistrements issus de la base de données en un document JSON correspondant :

```
$debut = true;
$nbColonnes=mysql_num_fields($reponseSQL);
echo "{\"gains\":[";
if (mysql_num_rows($reponseSQL)){
  while ($ligne = mysql_fetch_array($reponseSQL)) {
    if ($debut){
      echo "{";
      $debut = false;
     } else {
       echo ",{";
     }
    for($j=0;$j<$nbColonnes;$j++){
      $colonne=mysql_field_name($reponseSQL,$j);
      echo "\"".$colonne."\":\"". utf8_encode($ligne[$colonne])."\"";
      if ($j != $nbColonnes-1) echo ",";
    } // Fin de la boucle for
  echo "}";
  } // Fin de la boucle while
} // Fin de la boucle if
echo "]}";
```

Dans le document JSON, les couples nom colonne/valeur des données de la base sont représentés par des séries au format `"nomColonne":"valeur"`. Chaque enregistrement est composé de plusieurs séries séparées par une virgule. L'exemple du code 13-15, présente trois enregistrements (représentés sur trois lignes différentes) chacun composé de deux séries.

La première instruction du début du programme permet d'initialiser une variable `$debut` qui va être utilisée ultérieurement pour détecter s'il s'agit du premier enregistrement ou non. L'affichage de la virgule qui précède l'accolade du début de l'enregistrement étant par la suite conditionné par le test `if($debut)`, la virgule s'affiche ainsi au début de toutes les séries sauf de la première.

La seconde instruction permet de mémoriser dans la variable `$nbColonnes` le nombre de colonnes de chaque enregistrement retourné par le serveur de base de données. Cette variable est ensuite exploitée dans la boucle `for()` qui affiche autant de séries au format `"nomColonne":"valeur"` qu'il y a de colonne (dans notre exemple, nous n'avons que deux colonnes : `montant` et `date`, voir l'exemple ci-dessous).

```
"montant":"37","date":"2007-09-26 02:01:17"
```

Afin de ne pas avoir de virgule après la dernière série d'un même enregistrement, une structure de test `if()` intégrée à la boucle conditionne son affichage.

```
if ($j != $nbColonnes-1) echo ",";
```

Le début du document est toujours identique (même s'il n'y aucun enregistrement, dans ce cas le contenu des [ ] est vide).

```
echo "{\"gains\":[";
```

La boucle `while()` permet d'afficher autant de lignes qu'il y a d'enregistrements dans la réponse de la base de données. Ainsi, dans l'exemple du code 13-15, il y a trois enregistrements, la boucle `while()` parcourt trois fois son corps de boucle et affiche trois lignes.

À noter que le corps de boucle `while()` contient aussi la boucle `for()` qui permet de créer autant de séries qu'il y a de colonnes et cela pour chaque ligne d'enregistrement.

```
while ($ligne = mysql_fetch_array($reponseSQL)) {
…
} // Fin de la boucle while
```

Enfin, le document JSON est clôturé par les deux caractères `]}` grâce à l'instruction ci-dessous :

```
echo "]}";
```

Une fois toutes les instructions modifiées, enregistrez votre fichier PHP et revenez au fichier `fonctionsMachine.js` pour créer la fonction de rappel `afficheGains` qui va traiter le document JSON renvoyé par le programme PHP que nous venons de décrire.

Après les tests `if()` communs aux autres fonctions de rappel, nous allons récupérer la chaîne de caractères correspondant au document JSON dans la variable `listeJSON` au moyen de l'instruction ci-dessous.

```
listeJSON = objetXHR3.responseText;
```

La conversion de cette chaîne en objet JavaScript est réalisée à l'aide de la méthode `parseJSON()` disponible grâce à la bibliothèque `json.js` installée préalablement (revoir le début de cette procédure pour l'ajout de la balise dans le fichier `index.html` faisant référence à la bibliothèque `json.js`).

```
   objetJSON3=listeJSON.parseJSON();
```

Avant d'extraire les différents contenus de cet objet pour construire le tableau HTML, nous allons créer une variable `tableListe` qui référence la balise `<tbody>` dont l'identifiant est `"tableListe"`. Cette balise correspond au corps du tableau dans lequel nous allons intégrer l'historique des gains par la suite.

```
var tableListe=document.getElementById("tableListe");
```

Afin de pouvoir réinitialiser le contenu de cette table à chaque actualisation des données, nous allons utiliser la fonction `supprimerContenu()` déjà créée et qui se trouve dans la bibliothèque `fonctionsAjax.js`.

```
supprimerContenu(tableListe);
```

La boucle `for()` suivante permet de parcourir tous les enregistrements contenus dans l'objet JavaScript et d'en extraire le montant et la date à chaque tour de boucle. Une fois les données d'un enregistrement récupérées, nous allons utiliser une fonction (détaillée ci-après) pour créer une nouvelle ligne dans le tableau HTML afin d'y intégrer les données correspondantes.

```
for(i=0;i<objetJSON3.gains.length;i++){
   var montant=objetJSON3.gains[i].montant;
   var date=objetJSON3.gains[i].date;
   nouvelleLigne(tableListe,date,montant);
}
```

Une fois ces modifications effectuées, la fonction de rappel complète devrait alors être semblable au code 13-17.

Code 13-17 : fonction `afficheGains()` :

```
function afficheGains() {
```

```
if (objetXHR3.readyState == 4) {
 if (objetXHR3.status == 200) {
   listeJSON = objetXHR3.responseText;
   objetJSON3=listeJSON.parseJSON();
   var tableListe=document.getElementById("tableListe");
   supprimerContenu(tableListe);
   for(i=0;i<objetJSON3.gains.length;i++){
     var montant=objetJSON3.gains[i].montant;
     var date=objetJSON3.gains[i].date;
     nouvelleLigne(tableListe,date,montant);
   }
 }
}
```

La fonction de création des lignes du tableau HTML exploite les méthodes du DOM. Trois arguments lui sont passés, à savoir le nom du tableau auquel doit être rattachée la nouvelle ligne et les deux textes qui viennent s'insérer dans les deux cellules de la ligne.

```
function nouvelleLigne(tab,text1,text2) {
```

La première instruction de la fonction permet de créer un nouvel élément `tr`.

```
var nouveauTR=document.createElement('tr');
```

Nous allons ensuite avoir deux ensembles de trois instructions qui permettent de créer les éléments et le contenu des cellules de la ligne.

```
var nouveauTD1=document.createElement('td');
var nouveauTXT1=document.createTextNode(text1);
nouveauTD1.appendChild(nouveauTXT1);
//---------------------------------
var nouveauTD2=document.createElement('td');
var nouveauTXT2=document.createTextNode(text2);
nouveauTD2.appendChild(nouveauTXT2);
```

Une fois ces deux cellules générées (`nouveauTD1` et `nouveauTD2`), les deux instructions suivantes permettent de les rattacher à la ligne créée au début de la fonction (élément `nouveauTR`).

```
nouveauTR.appendChild(nouveauTD1);
nouveauTR.appendChild(nouveauTD2);
```

Il suffit ensuite de rattacher l'élément `nouveauTR` au corps du tableau (soit l'élément `tbody` passé en paramètre dans le premier argument de la fonction : `tab`) qui va recevoir l'historique des gains à l'aide de l'instruction suivante :

```
tab.appendChild(nouveauTR);
```

Une fois terminée, la fonction doit être semblable au code 13-18.

Code 13-18 : fonction `nouvelleLigne()` :

```
function nouvelleLigne(tab,text1,text2) {
  var nouveauTR=document.createElement('tr');
  //---------------------------------------
  var nouveauTD1=document.createElement('td');
  var nouveauTXT1=document.createTextNode(text1);
  nouveauTD1.appendChild(nouveauTXT1);
  var nouveauTD2=document.createElement('td');
```

```
    var nouveauTXT2=document.createTextNode(text2);
    nouveauTD2.appendChild(nouveauTXT2);
    //-----------------------------------------
    nouveauTR.appendChild(nouveauTD1);
    nouveauTR.appendChild(nouveauTD2);
    //-----------------------------------------
    tab.appendChild(nouveauTR);
}
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Comme dans l'atelier précédent, saisissez un des deux noms enregistrés dans la table (Defrance ou Dupond). Dès que le joueur est identifié par le système de vérification du nom (le message « joueur identifié » doit s'afficher) la requête Ajax du nouveau moteur que nous venons de mettre en place dans cet atelier doit être envoyée au serveur et les données récupérées dans la réponse JSON doivent s'afficher dans le tableau HTML situé en bas de l'écran (voir figure 13-11).

Si vous cliquez maintenant sur le bouton JOUER, la requête du premier moteur Ajax est envoyée comme d'habitude au serveur et le nouveau gain est enregistré dans la base de données, de même la réponse de cette requête retourne les informations du résultat qui sont affichées dans la page grâce au script de la fonction de rappel de ce premier moteur. Cependant, comme nous avons aussi ajouté à cette fonction de rappel un appel à notre nouveau moteur Ajax (affichage de l'historique), le résultat affiché dans le tableau HTML est actualisé et une nouvelle ligne contenant le dernier gain du joueur devrait être ajouté à l'historique.

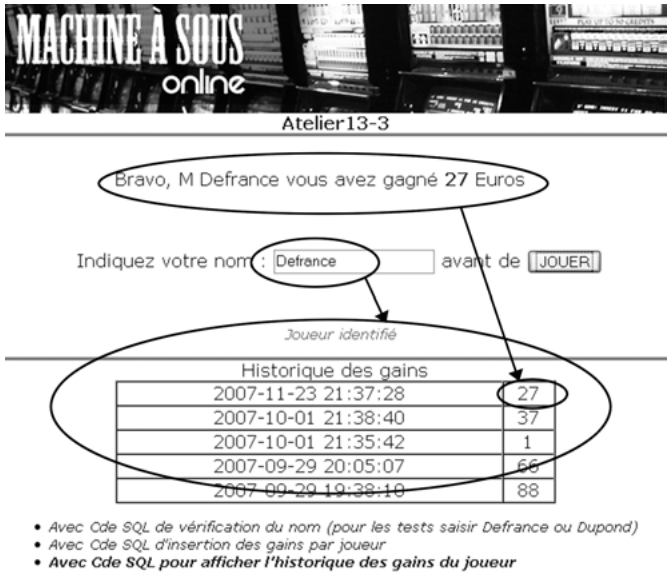


Figure 13-11

Test du système d'affichage de l'historique des gains du joueur

Atelier 13-4 : double menu déroulant dynamique

Composition du système

Un double menu déroulant dynamique se caractérise par le fait que les options du second menu sont créées dynamiquement suite au choix de l'utilisateur dans le premier menu. On peut ainsi créer un système de sélection des codes postaux dans le premier menu qui déclenche l'insertion des différentes villes ayant le même code postal dans le second menu. Une autre utilisation courante du double menu dynamique consiste à faciliter le choix de la couleur d'un article dans un site marchand en affichant dans le second menu les différentes couleurs disponibles pour l'article sélectionné dans le premier menu.

Les applications de ce type de système sont nombreuses et variées, aussi nous vous proposons dans cet atelier d'illustrer sa mise en œuvre en l'appliquant à notre machine à sous en ligne dans laquelle nous allons aider le joueur à sélectionner son identité. Pour cela, le premier menu affichera une liste de tous les noms disponibles dans la table `joueurs`. Une fois que l'utilisateur aura fait son choix dans cette première liste, une application Ajax interrogera la base de données pour récupérer tous les prénoms associés au nom sélectionné afin de les insérer dans le second menu du système.

Cette structure est composée :

- d'une page PHP (`index.php`) dont la structure de base avant modification est semblable à celle de la page `index.html` de l'atelier 13-3 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est identique à celle de l'atelier 13-3 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier 13-3 ;
- d'un fichier serveur PHP (`gainListe.php`) identique à celui de l'atelier 13-3 ;
- d'un nouveau fichier serveur PHP (`prenomsListe.php`) ayant comme base de départ le fichier `gainListe.php` de l'atelier 13-3 ;
- d'un fichier PHP de connexion à la base de données (`connexionMysql.php`) identique à celui de l'atelier 13-3 ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 13-3 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

L'application de cet atelier diffère du précédent par son système d'identification du joueur. En effet, ce dernier n'aura plus à saisir son nom comme dans l'atelier précédent, mais devra sélectionner successivement son nom puis son prénom dans un double menu déroulant dynamique. Les autres fonctionnalités (hormis le système de vérification du nom devenu inutile) seront conservées et fonctionneront de la même manière que dans l'atelier précédent.

Conception du système

Avant même de commencer à modifier le code des fichiers de ce nouvel atelier, nous devons d'abord ajouter de nouveaux utilisateurs de la même famille que les deux joueurs déjà présents (Defrance et Dupond) afin de disposer d'une base de données test comportant plusieurs joueurs ayant le même nom. Dans notre démonstration nous allons utiliser la liste des joueurs de la figure 13-12. Nous vous conseillons de saisir les mêmes informations si vous désirez pouvoir comparer vos propres résultats avec ceux des figures de ce chapitre.

Pour ajouter ces nouveaux joueurs, vous allez utiliser le gestionnaire de la base de données phpMyAdmin (accessible depuis le manager de Wamp 5). Dès l'ouverture du gestionnaire, sélectionner la base `machineasous` dans le menu de gauche puis cliquez sur le bouton Afficher sur la ligne de la table `joueurs` (premier bouton de la ligne). Un tableau contenant les enregistrements actuels doit alors apparaître. Pour ajouter de nouveaux joueurs, cliquez sur l'onglet Insérer en haut de la page. Un formulaire vous permet alors de saisir deux nouveaux joueurs. Renouvelez cette opération pour obtenir la même table que celle de nos tests (voir figure 13-12). Une fois ces ajouts effectués, vous pouvez fermer le gestionnaire pour passer aux modifications du code.

Figure 13-12

Liste des noms et prénoms de la table joueurs utilisée pour les tests de cet atelier

Ouvrez maintenant la page HTML de l'atelier 13-3 (`index.html`) et sauvegardez-la sous le nom `index.php` dans un nouveau répertoire nommé `/chap13/atelier13-4/` (attention de ne pas oublier l'extension `.php` pour ce nouveau fichier).

Dans cet atelier, la page principale de l'application doit être une page `.php` car nous allons y intégrer un script PHP pour générer le premier menu déroulant en récupérant dans la base les différents noms de famille des joueurs. Ainsi si, par la suite, un autre joueur est

ajouté dans la base de données, il apparaîtra automatiquement dans ce premier menu lors de la prochaine ouverture de l'application.

Affichez la page en mode Création et remplacez le champ de saisie du nom par deux menus déroulants en utilisant le bouton de l'onglet Formulaire de la barre Insertion correspondante (bouton Liste/Menu). Nommez ensuite le premier menu `nom` et le second `prenom` en utilisant le panneau Propriétés (sélectionnez au préalable le menu à configurer dans la fenêtre Document).

En guise d'exemple, nous allons commencer par créer le premier menu en statique (le nombre d'options du menu ne sera pas actualisé automatiquement dans le cas de l'ajout d'une nouvelle famille). Pour configurer manuellement les 3 options du premier menu (Sélectionner, Defrance et Dupond) sélectionnez le menu puis cliquez sur le bouton Valeurs de la liste dans le panneau Propriétés. Ajoutez ensuite une option par liste selon les étiquettes et les valeurs du menu désirées.

Si nous passons maintenant en mode Code, les balises correspondantes au premier menu doivent être semblables à celles du code 13-19 ci-dessous.

Code 13-19 : code d'un premier menu statique :

```
<select name='nom' id='nom' >
     <option value="">Sélectionner votre nom</option>
     <option value='Defrance'> Defrance </option>
     <option value='Dupond'> Dupond </option>
</select>
```

Passons maintenant à la transformation de ce premier menu pour le rendre dynamique. Pour cela, nous allons utiliser un script PHP qui va interroger la base de données pour en extraire la liste des différents noms de famille des joueurs. Cette liste est ensuite utilisée pour créer dynamiquement autant de balises `<option>` que nous avons d'enregistrements retournés par le serveur de base de données. Les instructions utilisées pour réaliser ce menu dynamique sont semblables à celles du code 13-20. L'ensemble de ces instructions doit se substituer aux balises du code 13-19 qui permettaient de créer le même menu mais en statique dans la page `index.php`.

Code 13-20 : script PHP de création dynamique du premier menu (celui des noms) :

```
<?php
require_once('connexionMysql.php');
mysql_select_db($base);
$requeteSQL="SELECT DISTINCT nom FROM joueurs ORDER BY nom";
$reponseSQL = mysql_query($requeteSQL);
echo "<select name='nom' id='nom' >";
echo "<option value=''>Sélectionner votre nom</option>";
while ($donnees = mysql_fetch_array($reponseSQL)) {
  echo "<option value='".$donnees['nom']."'> ".$donnees['nom']." </option>";
}
echo "</select>";
?>
```

Les deux premières instructions du code 13-20 permettent d'établir une connexion à la base de données en incluant dans la page le fichier `connexionMysql.php` (dans lequel se

trouvent les paramètres et la fonction de connexion) et d'en sélectionner la base de données `machineasous`.

```
require_once('connexionMysql.php');
mysql_select_db($base);
```

La troisième instruction contient la requête SQL destinée à récupérer les différents noms de famille de la table `joueurs` classés par ordre alphabétique. La clause `DISTINCT` permet d'éliminer les noms identiques dans la liste. Une fois définie, la requête est ensuite soumise au serveur de base de données à l'aide de la fonction `mysql_query()`.

```
$requeteSQL="SELECT DISTINCT nom FROM joueurs ORDER BY nom";
$reponseSQL = mysql_query($requeteSQL);
```

Les deux lignes suivantes (voir ci-dessous) permettent de générer en PHP les balises du début du menu qui ne sont pas intégrées dans la boucle `while()`.

```
echo "<select name='nom' id='nom' >";
echo "<option value=''>Sélectionner votre nom</option>";
```

La boucle `while()` qui génère les différentes balises `<option>` est un peu particulière car la condition de boucle contient une instruction `mysql_fetch_array()` qui transfère à chaque tour de boucle un enregistrement du résultat dans un tableau nommé `$donnees`. Ainsi nous pouvons ensuite facilement récupérer le nom du joueur à l'intérieur du corps de boucle en précisant le nom de sa colonne (soit `nom`) dans la clé du tableau (exemple : `$donnees['nom']`). Lorsqu'il n'y a plus d'enregistrements dans le résultat pointé par `$reponseSQL`, l'instruction incluse dans la condition de boucle renvoie alors la valeur `false` et entraîne la sortie de la boucle.

```
while ($donnees = mysql_fetch_array($reponseSQL)) {
  echo "<option value='".$donnees['nom']."'> ".$donnees['nom']." </option>";
}
```

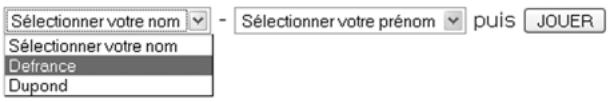
La dernière instruction de ce script permet de générer la balise fermante du menu déroulant afin que celui-ci soit conforme aux spécifications du W3C.

```
echo "</select>";
```

Les modifications de cette page sont terminées, vous pouvez à présent l'enregistrer. Le script du premier menu dynamique étant déjà opérationnel, nous vous suggérons de tester son fonctionnement dès maintenant (en utilisant la touche F12 dans Dreamweaver) afin de vous assurer que les différentes options du menu correspondent bien aux noms de famille des joueurs (Defrance et Dupond, voir figure 13-13).

Figure 13-13

Test préalable du premier menu dynamique



Ouvrez maintenant le fichier `fonctionsMachine.js` afin de créer le nouveau moteur Ajax pour cette nouvelle fonctionnalité. Pour commencer, nous allons ajouter les deux gestionnaires d'événements liés aux menus déroulants `nom` et `prenom` à la fin de la fonction `testerNavigateur()` (pour mémoire, cette fonction est appelée dès que la page est complètement chargée) afin d'être sûr que les gestionnaires ne soient pas appliqués aux éléments avant leur chargement dans la page.

Le premier gestionnaire doit permettre d'appeler la fonction `recupPrenom()` lorsque le joueur a sélectionné une entrée du menu `nom` (gestionnaire `onchange`). L'appel de cette fonction devra être accompagné en paramètre de la valeur sélectionnée dans le menu. Pour cette raison, nous ne nous contentons pas d'indiquer le nom de la fonction concernée mais nous allons utiliser la structure d'une fonction anonyme dans laquelle est inclus le nom de la fonction visée (`recupPrenoms()`) et son argument (`this.value`, voir code ci-dessous).

```
document.getElementById("nom").onchange=function() {recupPrenoms(this.value);}
```

Ancienne déclaration de gestionnaire d'événement

À titre de comparaison, le gestionnaire ci-dessus, déclaré directement dans le code JavaScript, est équivalent à celui que nous pourrions réaliser si nous l'avions inséré dans la page HTML avec la syntaxe ci-dessous. Cette ancienne méthode est désormais déconseillée car elle ne permet pas la séparation complète des scripts et de la structure de la page HTML.

```
<select name='nom' id='nom' onchange="recupPrenoms(this.value)" >
```

Le second gestionnaire permet de contrôler le menu `prenom` lorsqu'il est sélectionné à son tour (gestionnaire `onchange`). Cette fois, comme aucun paramètre n'est nécessaire, nous lui attribuons simplement le nom de la fonction qui doit être déclenchée lorsque la sélection du menu est effectuée (soit `demandeGains`).

```
document.getElementById("prenom").onchange=demandeGains;
```

Une fois modifiée, la fin de la fonction `testerNavigateur()` doit être semblable au code 13-21.

Code 13-21 : ajout des gestionnaires d'événements à la fin de la fonction `testerNavigateur()` :

```
function testerNavigateur() {
  objetXHR = creationXHR();
  if(objetXHR==null) {
  …
  }
  document.getElementById("button").onclick=jouer;
  document.getElementById("nom").onchange=function() {recupPrenoms(this.value);}
  document.getElementById("prenom").onchange=demandeGains;
}
```

Nous allons maintenant passer à la création du nouveau moteur Ajax qui a pour fonction d'envoyer une requête avec comme paramètre le nom du joueur sélectionné dans le premier menu et de récupérer la liste des prénoms correspondants pour les associer au second menu déroulant.

Pour créer ce quatrième moteur, nous allons procéder de la même manière que pour les trois derniers c'est-à-dire copier la structure des deux fonctions d'un précédent moteur pour ensuite les modifier. Une fois les deux fonctions copiées, commencez la personnalisation de la première en la renommant `recupPrenoms()`. Comme cette fonction doit récupérer la valeur du nom du joueur sélectionné dans le menu, nous allons ajouter un argument nommé `nom` entre les parenthèses de la fonction.

```
function recupPrenoms(nom) {
```

La première instruction permet de réinitialiser le second menu (celui des prénoms) au début de chaque appel de la fonction.

```
document.getElementById("prenom").options.length = 1;
```

La seconde instruction teste si l'utilisateur a bien sélectionné un nom de joueur et non la première option du menu invitant à sélectionner une option (voir ci-dessous).

```
<option value="">Sélectionner votre nom</option>
```

Si l'utilisateur a sélectionné cette première option, la valeur du nom récupéré étant vide, le test exécute alors un `return` qui interrompt le traitement en sortant de la fonction.

```
if (nom == "") return null ;
```

Dans le cas d'une sélection normale d'un nom de joueur du menu, le traitement se poursuit et l'objet XHR est alors créé.

```
objetXHR4 = creationXHR();
```

Il faut ensuite préparer le paramètre qui va être envoyé au format d'URL à la suite du nom du fichier serveur visé. Dans cette application, nous devons envoyer le nom du joueur sélectionné afin d'effectuer une recherche sélective des enregistrements qui ont cette valeur dans leur champ nom.

```
var parametres = "nom="+ nom + "&anticache="+temps ;
```

Une fois configurée, la variable `parametres` est ensuite ajoutée au nom du fichier `prenomListe.php` dans le second argument de la méthode `open()`.

```
objetXHR4.open("get","prenomsListe.php?"+parametres, true);
```

La propriété `onreadystatechange` de l'objet XHR doit ensuite être configurée avec le nom de la fonction de rappel qui doit assurer la récupération de la réponse asynchrone du serveur (soit `creationMenu2`).

```
objetXHR4.onreadystatechange = creationMenu2;
```

Enfin, la dernière instruction envoie la requête au serveur selon les paramétrages précédents.

```
objetXHR4.send(null);
```

Une fois modifiée, la fonction `recupPrenoms()` doit être semblable au code 13-22 :

Code 13-22 : fonction `recupPrenoms()` :

```
function recupPrenoms(nom) {
   document.getElementById("prenom").options.length = 1;
   if (nom == "") return null ;
   objetXHR4 = creationXHR();
   var temps = new Date().getTime();
   var parametres = "nom="+ nom +
                    "&anticache="+temps ;
   objetXHR4.open("get","prenomsListe.php?"+parametres, true);
   objetXHR4.onreadystatechange = creationMenu2;
   objetXHR4.send(null);
}
```

Nous allons maintenant passer à la fonction PHP qui va réceptionner et traiter cette requête. Nous reviendrons ensuite sur le fichier `fonctionsMachine.js` pour créer la fonction de rappel `creationMenu2()`.

Ouvrez le fichier `gainsListe.php` que nous allons utiliser comme base pour élaborer notre nouveau fichier PHP. Enregistrez-le sous le nom `prenomsListe.php` dans le même répertoire.

L'objectif de ce fichier serveur est de réceptionner le nom du joueur sélectionné dans le premier menu, puis de configurer et envoyer une requête SQL au serveur de base de données pour récupérer la liste des prénoms des enregistrements dont le nom est identique à celui du joueur.

Une fois la liste des prénoms disponibles dans le fichier PHP, deux alternatives sont envisageables pour créer le second menu déroulant avec les prénoms des personnes ayant pour nom celui sélectionné dans le premier menu.

Créez le code HTML du menu déroulant en PHP en y intégrant dynamiquement les valeurs et étiquettes des prénoms pour chaque option puis renvoyez ce fragment de code HTML au navigateur en utilisant la technique que nous avons déjà utilisée dans un atelier précédent (atelier 12-1 : transfert d'une réponse en HTML). Côté client, le fragment HTML est récupéré et intégré en lieu et place du second menu déroulant précédent.

Construisez un document JSON (nous pourrions le faire aussi de la même manière avec un document XML) contenant la liste des différents prénoms et envoyez-le au navigateur en utilisant la technique vue dans l'atelier précédent (atelier 13-3 : récupération d'une liste de données dans la base). Côté client, le document JSON est récupéré et converti en objet JavaScript avant d'être traité pour créer un arbre DOM correspondant qui remplacera le second menu existant.

Dans le cadre de cet atelier, nous allons utiliser la seconde technique. Cependant, nous devons avant cela commencer par créer une requête SQL afin de récupérer la liste des prénoms dans la table `joueurs` en fonction du nom envoyé en paramètre.

```
$commandeSQL="SELECT prenom FROM joueurs WHERE nom='".$nom."'";
$reponseSQL = mysql_query($commandeSQL);
```

Une fois le jeu d'enregistrements contenant cette liste des prénoms (`$reponseSQL`) disponible dans le fichier PHP, nous pouvons appliquer le même programme de création d'un document JSON que dans l'atelier précédent (revoir si besoin les commentaires sur ses différentes instructions) à la différence près que le nom de l'objet racine est cette fois `listePrenoms` (et non `gains` comme dans le fichier précédent).

Le fichier serveur une fois modifié doit être semblable à celui du code 13-23.

Code 13-23 : fichier `prenomsListe.php` :

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
if(isset($_REQUEST['nom'])) $nom=$_REQUEST['nom'];
else $nom="inconnu";
require_once('connexionMysql.php');
mysql_select_db($base);
$commandeSQL="SELECT prenom FROM joueurs WHERE nom='".$nom."'";
$reponseSQL = mysql_query($commandeSQL);
$debut = true;
$nbColonnes=mysql_num_fields($reponseSQL);
```

```
echo "{\"listePrenoms\":[";
if (mysql_num_rows($reponseSQL)){
  while ($ligne = mysql_fetch_array($reponseSQL)) {
    if ($debut){
     echo "{";
     $debut = false;
  } else {
     echo ",{";
  }
  for($j=0;$j<$nbColonnes;$j++){
     $colonne=mysql_field_name($reponseSQL,$j);
     echo "\"".$colonne."\":\"". utf8_encode($ligne[$colonne])."\"";
     if ($j != $nbColonnes-1) echo ",";
  } // Fin de la boucle for
  echo "}";
  } // Fin de la boucle while
} // Fin de la boucle if
echo "]}";
```

Enregistrez le fichier PHP et revenez maintenant au fichier `fonctionsMachine.js` pour créer la fonction de rappel qui va réceptionner et traiter le document JSON.

La réception du document JSON et la conversion en un objet JavaScript est identique à l'atelier précédent.

```
var nouveauResultat = objetXHR4.responseText;
var objetJSON=nouveauResultat.parseJSON();
```

Une fois que les données sont disponibles dans l'objet JSON, nous allons les récupérer en utilisant une boucle `for()` qui va parcourir les différents éléments placés dans la liste `listePrenom` (chacun de ces éléments correspond à un enregistrement de la requête SQL).

```
for (i=0; i<objetJSON.listePrenoms.length; i++)
{
```

À chaque tour de boucle (il y a autant d'itérations qu'il y a de prénoms à intégrer dans le menu déroulant), nous allons construire un élément `option` avec son contenu et son attribut `value` à l'aide des méthodes DOM.

```
var elementOption = document.createElement('option');
var prenom= objetJSON.listePrenoms[i].prenom;
var texteOption = document.createTextNode(prenom);
elementOption.setAttribute('value', prenom);
elementOption.appendChild(texteOption);
```

Pour illustrer ce que font ces instructions, si la variable `prenom` (récupérée dans l'objet JSON pour le tour de boucle considéré) est égale à `Claire`, la balise suivante devrait alors être créée au terme de ce tour de boucle.

```
<option value='Claire' > Claire </option>
```

Une fois construit, l'élément `option` est ensuite rattaché à son élément père (soit la balise `<select id="prenom">`) afin de construire un menu déroulant avec autant d'options que de prénoms correspondant au nom du joueur.

```
document.getElementById("prenom").appendChild(elementOption);
```

Une fois modifiée, la fonction `creationMenu2()` devrait être semblable au code 13-24.

Code 13-24 : fonction `creationMenu2()` :

```
function creationMenu2(){
if (objetXHR4.readyState == 4) {
 if (objetXHR4.status == 200) {
   var nouveauResultat = objetXHR4.responseText;
   var objetJSON=nouveauResultat.parseJSON();
   for (i=0; i<objetJSON.listePrenoms.length; i++)
   {
     var elementOption = document.createElement('option');
     var prenom= objetJSON.listePrenoms[i].prenom;
     var texteOption = document.createTextNode(prenom);
     elementOption.setAttribute('value', prenom);
     elementOption.appendChild(texteOption);
     document.getElementById("prenom").appendChild(elementOption);
   } // Fin de la boucle for
 } // Fin de la boucle if status
} // Fin de la boucle if readyState
} // Fin de la fonction
```

La nouvelle application de cet atelier devrait théoriquement être opérationnelle. Néanmoins, si le système d'identification du joueur est désormais basé sur ce double menu déroulant dynamique, le moteur Ajax vérifiant l'identité du joueur que nous avions développé lorsqu'il s'agissait d'une identification par la saisie du nom dans un simple champ, n'a plus de raison d'être et doit être supprimé du fichier `fonctionsMachine.js`.

Cependant, ce moteur (Moteur 2 composé des fonctions `verifNoms()` et `afficheReponse()`) récupérait en retour du serveur l'identifiant de l'utilisateur dans la table `joueurs` qui attestait l'existence du nom de l'utilisateur saisi (sinon le serveur renvoyait la valeur 0). Or, si nous ne disposons plus de la valeur de cet identifiant côté client, le système d'insertion du gain dans la base et d'affichage de l'historique ne peuvent donc plus fonctionner (ces deux systèmes utilisent actuellement la clé primaire du joueur pour ajouter son nouveau gain ou sélectionner l'historique de ses gains dans la base de données). Il est donc nécessaire d'apporter des modifications à ces deux systèmes si l'on désire conserver leurs fonctionnalités.

Il faut donc compenser l'absence de cet identifiant par l'ajout du couple nom et prénom du joueur dans les paramètres des requêtes envoyées au serveur de ces deux moteurs Ajax.

Ainsi, la fonction `jouer()` du premier moteur doit être semblable à celle du code 13-25 après modification.

Code 13-25 : fonction `jouer()` :

```
function jouer() {
  objetXHR = creationXHR();
  var temps = new Date().getTime();
  var parametres = "nom="+ codeContenu("nom") +
                   "&prenom="+ codeContenu("prenom") +
                   "&anticache="+temps ;
  objetXHR.open("get","gainAleatoire.php?"+parametres, true);
  …
}
```

De même, la fonction `demandeGains()` du troisième moteur doit être semblable au code 13-26 après modification.

Code 13-26 : fonction `demandeGains()` :

```
function demandeGains() {
  objetXHR3 = creationXHR();
  var temps = new Date().getTime();
  var parametres3 = "nom="+ codeContenu("nom") +
                    "&prenom="+ codeContenu("prenom") +
                    "&anticache="+temps ;
  objetXHR3.open("get","gainListe.php?"+parametres3, true);
…
}
```

Les fichiers serveur concernés par ces deux requêtes doivent, eux aussi, être modifiés de sorte à exploiter désormais le couple nom et prénom du joueur et non plus son identifiant comme dans le chapitre précédent.

Les requêtes SQL du fichier `gainAleatoire.php` doivent ainsi être modifiées en conséquence comme l'illustre le code 13-27.

Code 13-27 : extrait des instructions modifiées dans le fichier `gainAleatoire.php` :

```
$requeteSQL="SELECT ID FROM joueurs WHERE prenom='".$prenomJoueur.
➥"' AND nom='".$nomJoueur."' ";
$reponseSQL = mysql_query($requeteSQL);
$tableau=mysql_fetch_array($reponseSQL);
$commandeSQL="INSERT INTO gains SET montant='".$gain."',
➥joueursID='".$tableau["ID"]."', date=NOW() ";
mysql_query($commandeSQL);
```

De même, le fichier `gainListe.php` devra désormais exploiter une requête avec jointure sur les deux tables `gains` et `joueurs` pour récupérer l'historique des gains du joueur concerné comme l'illustre le code 13-28 ci-dessous.

Code 13-28 : extrait des instructions modifiées dans le fichier `gainListe.php` :

```
$commandeSQL="SELECT gains.montant, gains.date FROM gains, joueurs WHERE
➥gains.joueursID=joueurs.ID AND joueurs.prenom='".$prenom."' AND joueurs.nom='".$nom."' ";
$reponseSQL = mysql_query($commandeSQL);
```

Les modifications sont maintenant terminées, il ne vous reste plus qu'à enregistrer les différents fichiers et tester le nouveau système dans le navigateur.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Commencez par sélectionner le nom du joueur désiré dans le premier menu. Le second menu doit alors proposer la liste de tous les prénoms associés au nom sélectionné dans le premier menu. Choisissez le prénom de votre choix dans le second menu. Une fois la sélection effectuée, l'historique des gains du joueur doit apparaître en bas de l'écran. Cliquez ensuite sur le bouton JOUER, comme dans les ateliers précédents, le bouton doit être bloqué et le chargeur doit apparaître pendant la période

du transfert. Dès le nouveau résultat réceptionné, le montant du gain doit s'afficher précédé du nom du joueur et une nouvelle ligne doit être ajoutée à l'historique (voir figure 13-14).

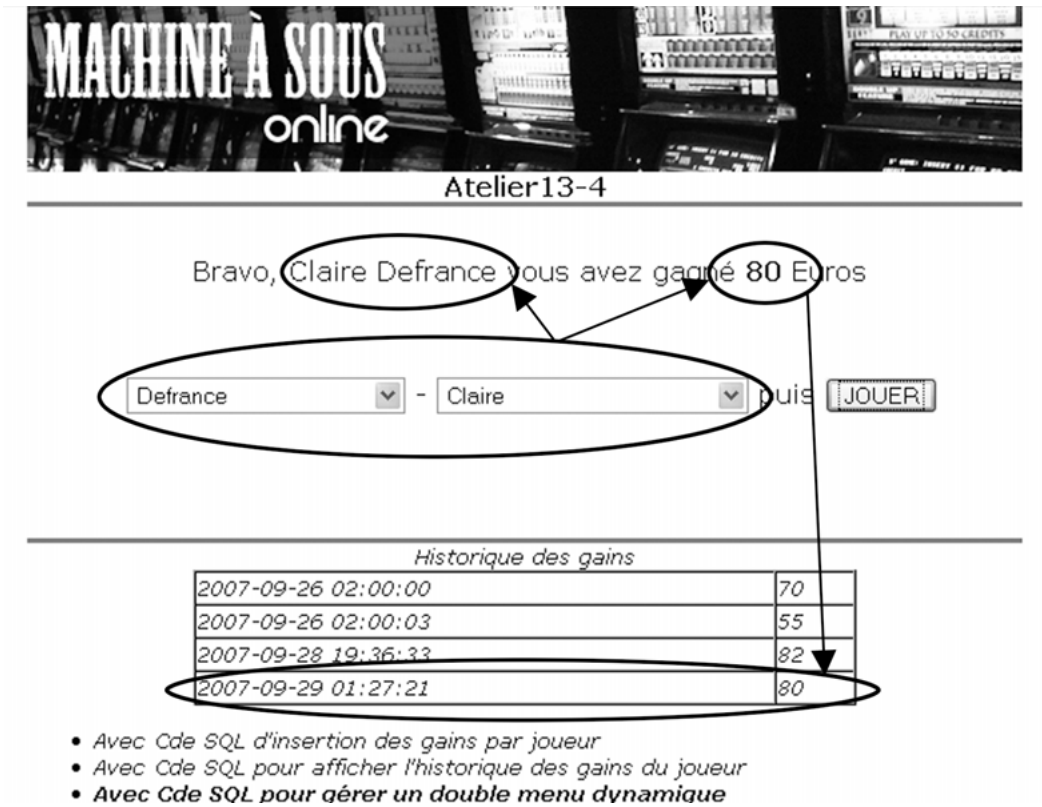


Figure 13-14

Test final du système avec un double menu déroulant dynamique

Atelier 13-5 : mise à jour de données dans la base de données

Composition du système

Nous avons vu dans l'atelier 13-3 comment insérer des données dans la base à partir d'une application Ajax-PHP. Nous vous proposons maintenant une technique pour que l'utilisateur puisse mettre à jour les informations de la base de données à partir de la même page.

Comme d'habitude, nous allons illustrer la mise en œuvre de cette technique en l'appliquant à notre machine à sous en ligne. Pour cela, nous allons permettre au joueur de corriger les résultats de ses propres gains dans le tableau de l'historique. Évidemment, l'objectif de cet atelier est purement pédagogique car vous pouvez imaginer qu'en pratique, un certain nombre de joueurs risquent d'abuser de cette nouvelle fonctionnalité pour modifier leur gain sans que cela soit justifié...

Cette structure est composée :

- d'une page PHP (`index.php`) dont la structure de base avant modification est semblable à celle de l'atelier 13-4 ;
- d'un fichier JS (`fonctionsAjax.js`) qui contient les fonctions communes à tous les moteurs Ajax ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est identique à celle de l'atelier 13-4 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier 13-4 ;
- d'un fichier serveur PHP (`gainListe.php`) identique à celui de l'atelier 13-4 ;
- d'un fichier serveur PHP (`prenomsListe.php`) identique à celui de l'atelier 13-4 ;
- d'un nouveau fichier serveur PHP (`gainModif.php`) ayant comme base de départ le fichier `gainListe.php` de l'atelier 13-4 ;
- d'un fichier PHP de connexion à la base de données (`connexionMysql.php`) identique à celui de l'atelier 13-4 ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 13-4 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le système de modification des gains va être intégré au tableau de l'historique que nous avons mis en place dans un atelier précédent. Son utilisation est très simple ; pour modifier la valeur d'un des gains de l'historique il suffit de cliquer sur le montant à actualiser dans le tableau. Celui-ci se transforme automatiquement en champ de saisie en conservant le montant actuel comme valeur initiale. Vous pouvez alors corriger le montant du gain comme dans un champ de formulaire traditionnel. Pour valider votre nouvelle saisie, il faut appuyer sur la touche Entrée de votre clavier. Une requête Ajax est alors envoyée au serveur PHP qui la transmet au serveur de base de données pour que la modification soit immédiatement effective. Si l'opération réussit, le serveur renvoie une information de validation au moteur Ajax et le tableau de l'historique est alors actualisé pour que la nouvelle valeur apparaisse, cette fois en affichage simple, dans la cellule du gain concerné.

Conception du système

Ouvrez la page d'index de l'atelier 13-4 (`index.php`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap13/atelier13-5/`. Copiez ensuite les autres fichiers de l'atelier 13-4 dans ce nouveau dossier.

Pour effectuer des modifications de données dans la table `gains` de la base, nous allons devoir disposer de la clé primaire de l'enregistrement à modifier. Il faut donc commencer par transformer la requête SQL qui récupère les données utilisées dans l'historique, soit actuellement `montant` et `date`, pour ajouter celle du `ID` de l'enregistrement concerné.

Pour cela, nous allons ouvrir le fichier `gainListe.php` et localiser la ligne dans laquelle est créée la requête SQL. Pour récupérer l'identifiant de l'enregistrement, il suffit d'ajouter le nom de la colonne que l'on désire récupérer (soit `ID` précédé du nom de la table dans laquelle elle se trouve : `gains.ID`).

```
$commandeSQL="SELECT gains.ID, gains.montant, gains.date FROM gains, joueurs WHERE gains
➥.joueursID=joueurs.ID AND joueurs.prenom='".$prenom."' AND joueurs.nom='".$nom."' ";
```

Si nous testons le système une fois la modification effectuée, le document JSON retourné par le serveur devrait alors avoir une structure semblable à celle de l'exemple du code 13-29.

Code 13-29 : exemple de document JSON renvoyé par le serveur après l'ajout de la colonne `ID` dans la requête SQL :

```
{"gains":[
  {"ID":"7","montant":"27","date":"2007-09-26 01:59:43"}
 ,{"ID":"8","montant":"14","date":"2007-09-26 01:59:48"}
 ,{"ID":"9","montant":"70","date":"2007-09-26 01:59:52"}
]}
```

Enregistrez le fichier `gainListe.php` et passons à présent côté client pour gérer ce nouveau paramètre dans la fonction de rappel du fichier `fonctionsMachine.js`.

Dans la fonction de rappel `afficheGains()`, ajoutez une instruction supplémentaire d'extraction de l'objet `objetJSON` dans la boucle `for()` afin de récupérer la valeur de l'identifiant `ID` comme l'illustre le code 13-30. Une fois la valeur de l'identifiant mémorisé dans la variable `ID`, il faut la passer en argument dans l'appel de la fonction `nouvelleLigne()` afin de pouvoir ensuite l'utiliser à l'intérieur de cette fonction.

Code 13-30 : modification de la fonction `afficheGains()` :

```
for(i=0;i<objetJSON3.gains.length;i++){
   var montant=objetJSON3.gains[i].montant;
   var date=objetJSON3.gains[i].date;
   var ID=objetJSON3.gains[i].ID;
   nouvelleLigne(tableListe,date,montant,ID);
}
```

Une fois l'identifiant `ID` disponible dans la fonction `nouvelleLigne()` nous devons l'intégrer dans chaque ligne afin de pouvoir le récupérer par la suite dans la procédure de modification d'un gain pour identifier l'enregistrement correspondant. Pour mémoriser cette information dans la structure de la ligne, nous allons ajouter un nouvel attribut `id` à l'élément `tr` (voir code 13-31).

Code 13-31 : ajout d'un nouvel attribut `id` à l'élément `tr` de la nouvelle ligne :

```
function nouvelleLigne(tab,text1,text2,ID) {
  var nouveauTR=document.createElement('tr');
  nouveauTR.setAttribute('id',ID);
```

Dans cette même fonction `nouvelleLigne()`, nous devons aussi lier la cellule `td` (soit l'élément `nouveauTD2`) qui contient le gain avec un gestionnaire d'événement `onclick` qui permet ensuite de déclencher le passage en mode modification (remplacement de la valeur du gain de la cellule par un champ de saisie). Pour cela, nous allons ajouter

l'instruction ci-dessous qui permet d'appeler la fonction `modifGain()` lorsque le joueur clique sur le montant du gain à modifier, en passant comme paramètres les éléments `tr` (`nouveauTR`), `td` (`nouveauTD2`) et le montant actuel du gain (`text2`) concernés.

```
nouveauTD2.onclick=function() { modeModif(nouveauTR,nouveauTD2,text2);}
```

Une fois modifiée, la nouvelle fonction `nouvelleLigne()` doit être semblable au code 13-32.

Code 13-32 : fonction `nouvelleLigne()` après modification :

```
function nouvelleLigne(tab,text1,text2,ID) {
  var nouveauTR=document.createElement('tr');
  nouveauTR.setAttribute('id',ID);
  //----------------------------------------
  var nouveauTD1=document.createElement('td');
  var nouveauTXT1=document.createTextNode(text1);
  nouveauTD1.appendChild(nouveauTXT1);
  var nouveauTD2=document.createElement('td');
  var nouveauTXT2=document.createTextNode(text2);
  nouveauTD2.onclick=function() { modeModif(nouveauTR,nouveauTD2,text2);}
  nouveauTD2.appendChild(nouveauTXT2);
  //----------------------------------------
  nouveauTR.appendChild(nouveauTD1);
  nouveauTR.appendChild(nouveauTD2);
  //----------------------------------------
  var nouveauTBODY=document.createElement('tbody');
  nouveauTBODY.appendChild(nouveauTR);
  tab.appendChild(nouveauTBODY);
}
```

Il faut maintenant développer la fonction `modeModif()` qui est appelée lorsque le joueur clique sur la valeur du gain à modifier. La première instruction de la fonction permet de récupérer la valeur de l'identifiant de la ligne contenant le gain à modifier (`elementTR.getAttribute('ID')`) puis de l'affecter à une variable globale nommée `idGain` (de sorte à pouvoir l'utiliser par la suite en dehors de cette fonction). La seconde instruction crée un élément `input` et l'affecte à la variable `elementInput`, elle aussi déclarée en global pour pouvoir y faire référence en dehors de la fonction.

```
function modeModif(elementTR,elementTD,valeurGain) {
  idGain=elementTR.getAttribute('ID');
  elementInput = document.createElement('input');
```

Une fois l'élément `input` créé, il faut ensuite le configurer en lui attribuant ses divers attributs et un gestionnaire d'événement qui permet de valider la saisie par une simple pression sur la touche Entrée (le programme de validation est développé dans une autre fonction nommée `testEntree()`).

Code 13-33 : instructions de configuration de l'élément `elementInput` :

```
elementInput.setAttribute('type','text');
elementInput.setAttribute('name','gain');
elementInput.setAttribute('size','20');
elementInput.setAttribute('velue', valeurGain);
elementInput.onkeypress=testEntree;
```

Une autre alternative (plus compacte) pour configurer l'élément `input` consiste à utiliser une structure `with` comme l'illustre le code 13-34 (équivalent aux instructions de la première solution du code 13-33). Comme vous pouvez le constater, l'utilisation de `with` permet de déclarer initialement l'élément concerné par les différentes configurations (soit `elementInput` dans notre cas) évitant ainsi d'avoir à le déclarer au début de chaque instruction et cela dans tout le bloc de la structure `with`.

Code 13-34 : seconde technique de configuration de l'élément `elementInput` :

```
with (elementInput) {
  setAttribute('type','text');
  setAttribute('name','gain');
  setAttribute('size','20');
  setAttribute('value',valeurGain);
  onkeypress=testEntree;
}
```

Maintenant que l'élément `input` est créé et configuré, nous devons l'intégrer à une nouvelle cellule `td` avant de pouvoir procéder au remplacement de la cellule `td` actuelle par cette dernière.

```
var nouveauTD = document.createElement("td");
nouveauTD.appendChild(elementInput);
elementTR.replaceChild(nouveauTD,elementTD);
```

La fonction `modeModif()` complète doit ensuite être semblable à celle du code 13-35.

Code 13-35 : fonction `modeModif()` complète :

```
function modeModif(elementTR,elementTD,valeurGain) {
  idGain=elementTR.getAttribute('ID');
  elementInput = document.createElement('input');
  with (elementInput) {
     setAttribute('type','text');
     setAttribute('name','gain');
     setAttribute('size','20');
     setAttribute('value',valeurGain);
     onkeypress=testEntree;
   }
   var nouveauTD = document.createElement("td");
   nouveauTD.appendChild(elementInput);
   elementTR.replaceChild(nouveauTD,elementTD);
}
```

Passons maintenant à la fonction `testEntree()` qui est appelée pour chaque touche saisie par l'utilisateur lorsque nous sommes en mode Modification. Le premier objectif de cette fonction est de détecter si l'utilisateur a appuyé sur la touche Entrée afin de déclencher le traitement d'actualisation de la nouvelle valeur du gain. Pour cela, nous allons utiliser une structure déjà présentée dans un atelier précédent permettant d'exploiter l'objet `event` quel que soit le type de navigateur utilisé (IE ou autre navigateur compatible W3C comme Firefox).

```
function testEntree(event) {
 event = window.event||event;
```

Une fois l'objet `event` disponible au sein de la fonction, nous pouvons utiliser sa propriété `keyCode` qui nous permet de récupérer le code de la touche saisie et de le tester pour déclencher le traitement uniquement s'il s'agit de la touche Entrée (soit le code 13 pour la touche Entrée).

```
var codeTouche= event.keyCode;
  if(codeTouche==13) // Test si touche Entrée
  {
  // Traitement de la valeur saisie
  }
```

Pour traiter la nouvelle valeur du gain, nous devons l'envoyer à un programme PHP qui permet de créer une requête SQL en rapport pour effectuer la mise à jour dans la base. Pour cela, nous allons faire appel à un nouveau moteur Ajax pour que ce traitement se fasse sans réactualiser la page comme nous en avons l'habitude maintenant.

Nous allons donc appeler la fonction `modifGain()` d'un nouveau moteur, mais avant cela nous devons préparer les paramètres que nous allons communiquer à cette fonction. Pour réaliser la mise à jour, le moteur Ajax a besoin de deux valeurs, la première est l'identifiant de l'enregistrement qui doit être modifié `idGain`. Ce dernier ayant été déclaré en variable globale nous pouvons donc en disposer au sein de cette fonction et l'indiquer en premier argument de la fonction `modifGain()` sans action préalable. Le second est la valeur du nouveau montant du gain que l'utilisateur vient de saisir. Pour cette seconde information, nous devons lire la valeur de l'élément `Input` (soit `elementInput`, disponible dans cette fonction car il a été, lui aussi, déclaré préalablement en global) avant de l'insérer en second argument de la fonction `modifGain()`.

```
var nouveauGain=elementInput.value;
modifGain(idGain,nouveauGain);
```

Une fois créée, la nouvelle fonction `testEntree()` doit être semblable au code 13-36.

Code 13-36 : fonction `testEntree()` complète :

```
function testEntree(event) {
  event = window.event||event;
  var codeTouche= event.keyCode;
  if(codeTouche==13) // Test si touche Entrée
  {
    var nouveauGain=elementInput.value;
    modifGain(idGain,nouveauGain);
  }
}
```

Pour la création du nouveau moteur, nous allons commencer par copier les deux fonctions d'un précédent moteur afin de bénéficier de la structure commune à tous les moteurs Ajax. Renommons la première fonction `modifGain()` et saisissez `id` et `montant` comme arguments dans les parenthèses de la fonction.

```
function modifGain(id,montant) {
```

Dans cette nouvelle fonction, nous allons commencer par préparer les paramètres que vous allez communiquer dans la requête asynchrone en y intégrant les deux arguments de la fonction.

```
var parametres = "id="+ id +
                 "&montant="+ montant +
                 "&anticache="+temps ;
```

Modifiez ensuite le second argument de la méthode `open()` en indiquant le nom du fichier serveur `gainModif.php` qui est chargé de faire la mise à jour de la base de données.

```
objetXHR5.open("get","gainModif.php?"+parametres, true);
```

Déclarez ensuite `actualiserGain` comme étant le nom de la fonction de rappel du moteur Ajax.

```
objetXHR5.onreadystatechange = actualiseGain;
```

Une fois les modifications effectuées, la fonction `modifGain()` doit être semblable au code 13-37.

Code 13-37 : fonction `modifGain()`après modification :

```
function modifGain(id,montant) {
 objetXHR5 = creationXHR();
 var temps = new Date().getTime();
 var parametres = "id="+ id +
                  "&montant="+ montant +
                  "&anticache="+temps ;
 objetXHR5.open("get","gainModif.php?"+parametres, true);
 objetXHR5.onreadystatechange = actualiseGain;
 objetXHR5.send(null);
}
```

Passons maintenant à la fonction PHP `gainModif.php` qui va traiter cette requête.

Pour créer ce fichier, nous partons sur la base du fichier `gainListe.php` que nous enregistrons sous le nom `gainModif.php`. À l'intérieur du fichier, configurez deux structures `if()` afin de réceptionner les deux variables HTTP `id` et `montant` qui ont été passées en paramètres lors de l'envoi de la requête Ajax.

```
if(isset($_REQUEST['id'])) $id=$_REQUEST['id'];
else $id=0;
if(isset($_REQUEST['montant'])) $montant=$_REQUEST['montant'];
else $montant=0;
```

Configurez ensuite la commande SQL qui permet de faire la mise à jour du gain. Cette commande identifie l'enregistrement à actualiser en se référant à sa clé primaire `ID` par une clause `WHERE`.

```
$commandeSQL="UPDATE gains SET montant='".$montant."' WHERE ID='".$id."'";
```

Lors de la soumission de la commande SQL au serveur de base de données avec la fonction `mysl_query()` nous allons récupérer la valeur retournée dans une variable `$reponse` (soit `true` si la soumission a été effectuée correctement ou `false` dans le cas contraire). Cette même variable est ensuite renvoyée au navigateur afin d'attester que la mise à jour a bien été effectuée.

```
reponseSQL = mysql_query($commandeSQL);
echo $reponseSQL ;
```

Le fichier gainModif.php complet doit ensuite être semblable au code 13-38.

Code 13-38 : fichier gainModif.php :

```
header("Content-Type: text/plain ; charset=utf-8");
header("Cache-Control: no-cache , private");
header("Pragma: no-cache");
if(isset($_REQUEST['id'])) $id=$_REQUEST['id'];
else $id=0;
if(isset($_REQUEST['montant'])) $montant=$_REQUEST['montant'];
else $montant=0;
require_once('connexionMysql.php');
mysql_select_db("machineasous");
$commandeSQL="UPDATE gains SET montant='".$montant."' WHERE ID='".$id."'";
$reponseSQL = mysql_query($commandeSQL);
echo $reponseSQL ;
```

Retournons maintenant côté client pour terminer la configuration du moteur Ajax dans le fichier fonctionMachine.js. Après avoir renommé la fonction de rappel du moteur Ajax avec actualiseGain(), placez-vous après l'instruction de récupération de l'information retournée par le fichier serveur (nouveauResultat) et ajoutez un test if() afin de conditionner l'appel de la fonction demandeGain() par l'état de l'information retournée. Ajoutez ensuite une structure alternative else afin d'afficher une fenêtre d'alerte signalant un problème technique dans le cas contraire.

```
if(nouveauResultat) demandeGains();
else alert("probleme technique");
```

Ainsi, lorsque le navigateur réceptionne la confirmation de la mise à jour du gain (dans ce cas nouveauResultat est égal à true) la fonction demandeGain() est appelée et le tableau de l'historique est réinitialisé avec les nouvelles valeurs de la base de données.

Une fois les modifications effectuées, la fonction de rappel actualiseGain() complète est semblable au code 13-39.

Code 13-39 : fonction de rappel actualiseGain() :

```
function actualiseGain(){
if (objetXHR5.readyState == 4) {
 if (objetXHR5.status == 200) {
  var nouveauResultat = objetXHR5.responseText;
  if(nouveauResultat) demandeGains();
  else alert("probleme technique");
 }
}
}
```

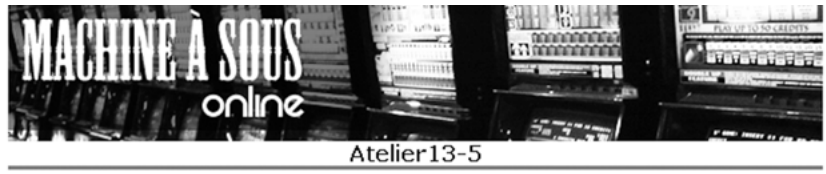
Test du système

Ouvrez la page index.html dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Commencez par sélectionner le nom d'un joueur dans le premier menu puis son prénom dans le second menu. Dès la sélection du prénom, le tableau de l'historique correspondant avec le joueur doit s'afficher en bas de l'écran. Choisissez un gain à modifier et cliquez sur sa cellule. Celle-ci doit alors se transformer en un champ de saisie avec comme valeur initiale celle du gain actuel. Modifiez cette valeur puis appuyez sur la touche Entrée du clavier pour valider votre saisie. La requête doit alors être envoyée au serveur et si l'actualisation de la base s'est bien effectuée, celui-ci doit renvoyer une

information de confirmation en retour qui déclenche une seconde requête pour actualiser le tableau de l'historique avec les nouvelles valeurs de la base de données.

Figure 13-15

Test du système de mise à jour des gains de l'historique



Atelier13-5

Defrance - Jean-Marie puis JOUER

Historique des gains

2007-09-29 19:38:10	88
2007-09-29 20:05:07	66
2007-10-01 21:35:42	1
2007-10-01 21:38:40	37

- *Avec Cde SQL d'insertion des gains par joueur*
- *Avec Cde SQL pour afficher l'historique des gains du joueur*
- *Avec Cde SQL pour gérer un double menu dynamique*
- ***Avec Cde SQL pour mettre à jour les gains des joueurs***

14

Bibliothèque jQuery

Jusqu'à présent, pour réaliser nos moteurs Ajax, nous avons utilisé l'objet XMLHttpRequest et certaines propriétés et méthodes JavaScript pour manipuler le DOM. Comme vous avez pu le constater, leurs syntaxes et surtout leurs compatibilités avec les navigateurs ne sont pas toujours sans poser de problèmes. Pour solutionner cela, nous allons maintenant faire usage d'une bibliothèque d'objets JavaScript qui va se substituer aux scripts habituels.

Il existe plusieurs bibliothèques JavaScript de ce genre actuellement, comme Prototype par exemple, mais dans le cadre de cet ouvrage nous avons choisi de vous présenter jQuery car cette bibliothèque remporte actuellement un succès croissant chez les développeurs. À tel point que certains CMS de renom, comme Drupal ou Spip, l'ont déjà adoptée, ce qui laisse présager d'un avenir prometteur pour elle.

Introduction à jQuery

jQuery est une bibliothèque JavaScript qui permet de manipuler le DOM très facilement, de gérer les événements, de créer des effets graphiques et d'implémenter des moteurs Ajax avec une syntaxe très simple et concise. La plupart des instructions jQuery commencent par la création d'un objet jQuery auquel on applique ensuite une ou plusieurs méthodes.

La classe jQuery

L'instanciation d'un objet jQuery peut être réalisée avec la syntaxe suivante :

```
jQuery()
```

Cependant, en pratique nous utilisons toujours le raccourci suivant qui est un alias de l'appel à la classe jQuery :

```
$()
```

Les sélecteurs

L'objet jQuery va nous permettre de sélectionner les éléments du DOM que nous désirons manipuler. Pour cela, nous allons insérer entre les parenthèses un sélecteur semblable à ceux des CSS pour définir l'élément (ou les éléments) désiré(s).

Par exemple, pour sélectionner tous les éléments `<p>` de la page, nous allons créer un objet jQuery de la manière suivante :

```
$("p")
```

De même, si nous voulons sélectionner l'ensemble des éléments portant sur la classe `menu`, nous utiliserons le code suivant :

```
$(".menu")
```

Enfin, pour sélectionner l'élément dont l'identifiant est `info`, il faut alors utiliser le code ci-dessous :

```
$("#info")
```

Les autres sélecteurs jQuery

L'utilisation de la syntaxe des CSS n'est pas la seule manière de sélectionner un élément du DOM avec jQuery. En effet, il est aussi possible d'exploiter certaines syntaxes XPATH ou encore des méthodes propres à l'objet jQuery, seules ou couplées avec la syntaxe CSS comme l'illustre l'exemple ci-dessous.

```
$("#info > li")
```

Cette instruction sélectionne tous les éléments `<li>` enfants de l'élément dont l'identifiant est `info`.

Pour découvrir les autres sélecteurs utilisables avec jQuery, rendez-vous sur le site suivant :

http://docs.jquery.com/DOM/Traversing/Selectors

Les méthodes

Une fois l'élément (ou le groupe d'éléments) sélectionné, nous pouvons lui appliquer des méthodes de la classe jQuery avec la syntaxe pointée :

```
$(selecteur).methode();
```

Il est aussi possible d'appliquer plusieurs méthodes les unes à la suite des autres. Dans ce cas les méthodes s'exécuteront comme elles se lisent, de la gauche vers la droite :

```
$(selecteur).methode1().methode2().methode3();
```

Avant de passer à la création d'un moteur Ajax avec jQuery, nous vous proposons de vous initier rapidement à l'usage de cette bibliothèque dans les parties suivantes avec quelques exemples d'instructions jQuery classées par thèmes.

Tester le chargement du DOM

Pour tester le chargement complet d'une page Web, nous avons jusqu'à maintenant utilisé le code ci-dessous :

```
window.onload = function() {
  //mettre ici les instructions qui s'exécutent si la page complète est chargée.
 };
```

Si l'on applique la méthode `load()` à l'élément `window`, l'équivalent en jQuery serait alors le code ci-dessous :

```
$(window).load(function() {
  //mettre ici les instructions qui s'exécutent si la page complète est chargée.
});
```

Cependant, en général, le chargement des éléments du DOM est assez rapide. Mais, si la page contient des images, il faut attendre leurs chargement complet pour que la méthode `load()` réagisse, alors que cela n'est pas nécessaire pour pouvoir appliquer des méthodes jQuery ou des gestionnaires d'événements aux éléments du DOM.

Aussi, pour optimiser la déclaration des méthodes ou des gestionnaires d'événements, jQuery met aussi à votre disposition une méthode spécifique `ready()` qui permet de détecter la fin du chargement des éléments du DOM sans attendre le chargement des images ou autres composants qui pourraient freiner leur configuration. Cette méthode s'appliquera par contre à l'élément `document` comme l'illustre le code ci-dessous.

```
$(document).ready(function() {
  // mettre ici les instructions qui s'exécutent si le DOM est chargé
});
```

Instructions de manipulation du DOM avec ou sans jQuery

Pour bien comprendre l'action des instructions jQuery pour chacun des exemples ci-dessous, nous allons rappeler l'instruction équivalente qui ne fait pas appel à la classe jQuery.

Installation de la bibliothèque jQuery

Pour que ces instructions jQuery puissent être utilisées en guise d'alternatives aux méthodes de manipulation du DOM courantes, il convient au préalable de disposer de la bibliothèque jQuery. Pour cela, vous devez la télécharger(*) sur le site `jquery.com` (voir figure 14-1) puis copier le fichier `jquery.js` dans un répertoire de votre site. Pour disposer de la classe jQuery et de ses nombreuses méthodes, il faut évidemment faire référence à ce fichier en ajoutant une balise `<script>` dans la page HTML concernée.

```
<script src="jquery.js" type="text/javascript"></script>
```

(*) La bibliothèque jQuery existe en version compressée (dans ce cas le code des classes et méthodes de la bibliothèque n'est pas lisible) ou non compressée (fichier plus volumineux mais avec un code lisible), la première version étant évidemment beaucoup plus légère et donc plus rapide à charger que la seconde. Pour un usage en production, nous vous conseillons de prendre la version compressée (Pack : environ 30ko) sachant que si vous désirez parcourir les instructions de la bibliothèque, vous avez toujours la possibilité de consulter la version non compressée.

À noter qu'il est aussi possible d'utiliser une bibliothèque en ligne pour vos développements si vous désirez faire rapidement des tests sans avoir à télécharger le fichier `jquery.js` sur votre ordinateur (vous pouvez trouver ci-dessous un exemple de balise permettant d'accéder à une bibliothèque en ligne). Cependant, en production il est fortement recommandé d'installer sa propre bibliothèque jQuery directement sur son serveur si vous désirez que le chargement de votre application Ajax ne soit pas tributaire d'un serveur externe.

```
<script type="text/javascript" src="http://code.jquery.com/jquery-latest.pack.js"></script>
```

☞

Installation de la bibliothèque jQuery *(suite)*

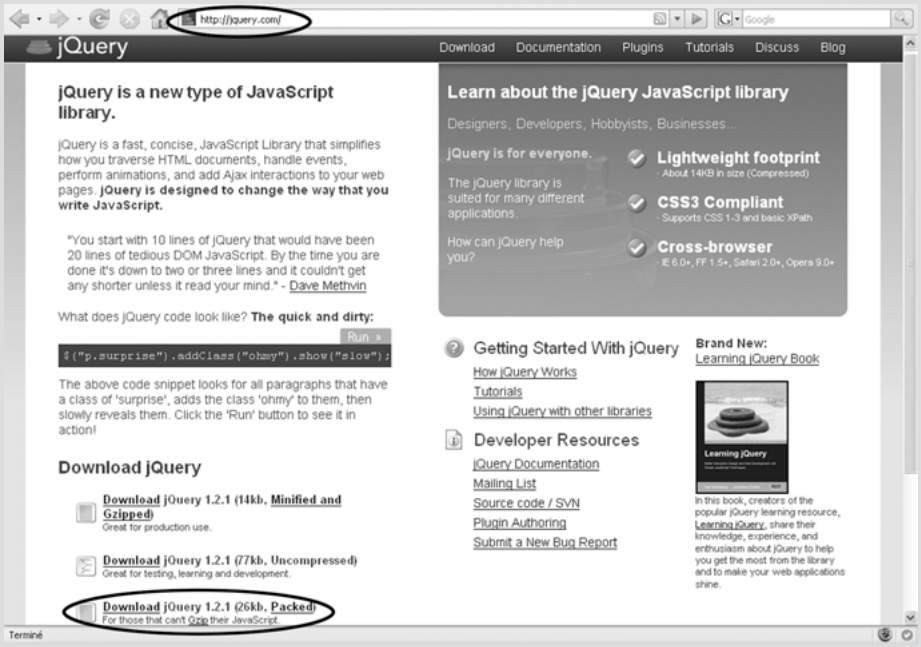


Figure 14-1

Page de téléchargement de la bibliothèque jQuery sur le site www.jquery.com

Vérifier que le DOM est bien chargé

jQuery permet de manipuler le DOM ou encore de configurer des gestionnaires d'événements. Cependant, avant d'appliquer les méthodes jQuery aux éléments du document, il convient de s'assurer que le DOM est complètement chargé. Pour cela, jQuery propose la méthode `ready()` qui permet de vérifier que le DOM est intégralement chargé, ce qui évite d'utiliser `window.onload()`. Aussi, si vous désirez tester individuellement les différents exemples de cette partie, nous vous recommandons d'utiliser cette méthode et d'y inclure le code à tester comme indiqué ci-dessous.

```
$(document).ready(function(){
  //mettre ici le code jQuery à tester
});
```

Récupérer l'élément dont l'identifiant est `charge` :

Code 14-1 **:** sans jQuery :

```
document.getElementById("charge");
```

Code 14-2 **:** avec jQuery :

```
$("#charge");
```

Récupérer la liste des éléments `<p>` d'une page Web :

Code 14-3 **:** sans jQuery :

```
document.getElementsByTagName("p");
```

Code 14-4 : avec jQuery :

```
$("p");
```

Changer le style `color` d'un élément dont l'identifiant est `info` :

Code 14-5 : sans jQuery :

```
document.getElementById("info").style.color="red";
```

Code 14-6 : avec jQuery :

```
$("#info").css("color","red");
```

Récupérer le texte d'un élément `<div>` dont l'identifiant est `info` :

Code 14-7 : sans jQuery :

```
document.getElementById("info").firstChild.nodeValue;
```

Code 14-8 : avec jQuery :

```
$("#info").text();
```

Récupérer la valeur d'un élément `<input>` dont l'identifiant est `nom` :

Code 14-9 : sans jQuery :

```
document.getElementById("nom").value;
```

Code 14-10 : avec jQuery :

```
$("#nom").val();
```

Remplacer le texte d'un élément `<div>` dont l'identifiant est `info` par un autre texte :

Code 14-11 : sans jQuery :

```
document.getElementById("info").firstChild.nodeValue="Defrance";
```

Code 14-12 : avec jQuery :

```
$("#info").html("Defrance");
```

Remplacer le texte d'un élément `<div>` dont l'identifiant est `info` par un fragment HTML :

Code 14-13 : sans jQuery :

```
document.getElementById("info").innerHTML="<b>Defrance</b>";
```

Code 14-14 : avec jQuery :

```
$("#info").html("<b>Defrance</b>");
```

Ajouter à la balise `<body>` un nouvel élément `<div>` contenant le texte `Defrance` en lui attribuant un style `background` :

Code 14-15 : sans jQuery :

```
var element = document.getElementsByTagName("body");
var nouveauElement = document.createElement("div");
var nouveauContenu = document.createTextNode("Defrance");
```

```
nouveauElement.style.background='yellow';
nouveauElement.appendChild(nouveauContenu);
element[0].appendChild(nouveauElement);
```

Code 14-16 : avec jQuery :

```
$('<div></div>')
  .html('Bonjour')
  .css('background', 'yellow')
  .appendTo("body");
```

Je pense qu'avec ces quelques exemples, vous êtes maintenant convaincu que l'utilisation de jQuery permet de rendre le code beaucoup plus compact et qu'il améliore ainsi sa lisibilité et sa maintenance.

Les autres méthodes de manipulation d'élément du DOM de jQuery

Pour découvrir les autres manipulation DOM utilisables avec jQuery, utilisez le lien ci-dessous :

http://docs.jquery.com/Manipulation

Configuration de gestionnaires d'événements avec ou sans jQuery

jQuery permet aussi de configurer des gestionnaires d'événements comme l'illustre l'exemple ci-dessous.

Configuration du gestionnaire d'événement `click()` pour la balise dont l'identifiant est `monLien` :

Code 14-17 : sans jQuery :

```
window.onload=function() {
 document.getElementById("monLien").onclick=function()
 {alert("Merci d'avoir clique sur ce lien !");};
};
```

Code 14-18 : avec jQuery :

```
$(document).ready(function() {
 $("#monLien").click(function()
 {alert("Merci d'avoir clique sur ce lien !");}
 );
});
```

Les autres gestionnaires d'événements jQuery

Pour découvrir les autres gestionnaires d'événements utilisables avec jQuery, utilisez le lien ci-dessous :

http://docs.jquery.com/Events

Création d'effets graphiques avec jQuery

Même s'il existe de nombreux plug-ins qui vous permettent ensuite d'enrichir votre interface avec des animations de nouvelle génération, jQuery propose déjà en standard quelques méthodes pour réaliser rapidement des effets graphiques simples comme l'illustrent les exemples ci-dessous.

Faire apparaître progressivement le texte des éléments `<div>` au chargement de la page :

Code 14-19 : avec jQuery :

```
$(document).ready(function() {
$("div").fadeIn("slow");
);
);
```

Rendre visible le texte de l'élément dont l'identifiant est `info` :

Si l'élément concerné est lié à un style qui le rend invisible au chargement de la page, il convient d'utiliser dans ce cas le style `display` et non `visibility`.

Code 14-20 **:** avec jQuery :

```
$("#info").show();
```

Les autres effets jQuery

Pour découvrir les autres effets utilisables avec jQuery, utilisez le lien ci-dessous :

http://docs.jquery.com/Effects

Création de moteurs Ajax avec jQuery

jQuery ne sert pas seulement à manipuler les éléments du DOM, à créer des effets graphiques ou à configurer des gestionnaires d'événements, il possède aussi une série de méthodes spécifiques à la mise en œuvre d'applications Ajax.

Création et configuration d'une requête Ajax POST asynchrone :

Code 14-21 **:** sans jQuery :

```
objetXHR = creationXHR();
var parametres = "nom=Defrance&prenom=Jean-Marie";
objetXHR.open("post"," serveur.php ", true);
objetXHR.onreadystatechange = fonctionRappel;
objetXHR.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
objetXHR.send(parametres);
function fonctionRappel() {
 if (objetXHR.readyState == 4) {
  if (objetXHR.status == 200) {
   var reponse = objetXHR.responseText;
   alert( "La réponse est : " + reponse );
  }
 }
}
function creationXHR() {
 var resultat=null;
 try {
  resultat= new XMLHttpRequest();
 }
 catch (Error) {
  try {
   resultat= new ActiveXObject("Msxml2.XMLHTTP");
  }
  catch (Error) {
```

```
    try {
     resultat= new ActiveXObject("Microsoft.XMLHTTP");
    }
    catch (Error) {
     resultat= null;
    }
   }
  }
 return resultat;
}
```

Code 14-22 : avec jQuery :

```
$.ajax({
    type: 'POST',
    url: 'serveur.php',
    data: "nom=Defrance&prenom=Jean-Marie",
    success: fonctionRappel
      });
function fonctionRappel(reponse) {
   alert( "La réponse est : " + reponse );
}
```

Avec les moteurs Ajax, plus encore qu'avec d'autres actions, jQuery permet d'obtenir un code très concis, lisible et simple à maintenir. Si on ajoute à cela le fait que ces moteurs peuvent être utilisés sans problème sur la plupart des navigateurs sans avoir à faire des scripts de test et prévoir des instructions alternatives pour que votre moteur puisse fonctionner, vous conviendrez qu'il devient très intéressant d'utiliser la bibliothèque jQuery.

Dans le code 14-22, vous remarquerez que cette fois l'objet jQuery (instancié par `$`) n'a pas de parenthèses car il n'est pas utilisé comme sélecteur (la méthode `ajax()` ne s'appliquant pas à un élément spécifique du DOM).

De même, la méthode `ajax()` comporte plusieurs options (`type`, `url`, `data`, `success`) qui permettent de configurer la requête Ajax ainsi créée. En réalité, il existe de nombreuses autres options possibles pour cette méthode `ajax()` mais comme leur valeur par défaut correspond à la valeur courante des requêtes Ajax, leur configuration n'est alors pas nécessaire. Par exemple, l'option `async` permet d'indiquer si la requête doit être effectuée en mode asynchrone (qui est la valeur par défaut, soit `true`) ou synchrone (si l'on attribue `false` à cette option). Ainsi, si vous désirez faire une requête Ajax synchrone, il suffit simplement d'ajouter l'option `async: false` à la liste des options de la méthode (pour connaître les différentes options de la méthode `ajax()`, utilisez le lien de l'encadré ci-dessous, cliquez alors sur `jQuery.ajax(option)`, puis sur l'onglet `Options`).

De même, il est intéressant de remarquer que la valeur de la réponse du serveur est envoyée automatiquement dans l'argument de la fonction de rappel (`fonctionRappel(reponse)`). Il devient donc inutile de faire appel à la propriété `responseText` de l'objet XHR pour récupérer cette valeur comme nous le faisions avant d'utiliser un moteur avec jQuery.

Les autres méthodes Ajax de jQuery

Pour découvrir les autres méthodes Ajax utilisables avec jQuery, utilisez le lien ci-dessous :

http://docs.jquery.com/Ajax

Visual jQuery

Pour chaque famille d'instructions jQuery abordée dans les parties précédentes, nous avons indiqué les liens sur la documentation officielle correspondante. Cependant, il existe une interface graphique qui permet d'accéder à ces différentes informations en sélectionnant successivement des options dans un menu. Cette interface se trouve à l'adresse ci-dessous et nous vous invitons à l'ajouter à vos favoris pour faciliter vos futurs développements.

http://visualjquery.com

Figure 14-2

Interface de Visual jQuery

Atelier 14-1 : requête asynchrone POST et réponse au format Texte avec jQuery

Composition du système

Afin que vous puissiez bien identifier les scripts à intégrer pour développer un moteur Ajax à l'aide de jQuery, nous allons reprendre certaines des applications déjà présentées et les modifier pour exploiter la bibliothèque jQuery. Le fonctionnement de ces applications est identique à celui des ateliers que nous allons utiliser comme base de départ et nous vous invitons à vous reporter à la partie qui présente l'atelier d'origine pour vous remémorer son fonctionnement.

Nous allons commencer par reprendre le système de l'atelier 11-1 qui exploitait une requête Ajax POST pour transmettre les valeurs du champ `nom` et `prenom` au serveur.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure est identique à celle de l'atelier 11-1 ;
- du fichier de la bibliothèque jQuery (`jquery.js`) qui est téléchargé puis ajouté dans un répertoire de cet atelier ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est semblable à celle de l'atelier 11-1 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 11-1 ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 10-4 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Vous avez certainement remarqué que la structure de ce premier atelier jQuery comporte maintenant le fichier de la bibliothèque `jquery.js` et que le fichier `fonctionsAjax.js` regroupant les différentes fonctions communes aux moteurs Ajax et à la manipulation d'éléments du DOM, désormais inutiles, a été supprimé.

Fonctionnement du système

Le fonctionnement du jeu est identique à celui de l'atelier 11-4 que nous allons prendre comme base de départ pour effectuer nos modifications.

Conception du système

Ouvrez la page HTML de l'atelier 11-1 précédent (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap14/atelier14-1/`. Copiez ensuite les autres fichiers de l'atelier précédent dans ce nouveau dossier.

Commencez par télécharger la dernière version de la bibliothèque jQuery sur le site `jquery.com`. Une fois sur la page d'accueil, cliquez sur le lien `download` dans le menu du haut puis sélectionnez la version courante `Current Release - Packed`, (pour nos essais nous avons utilisé la version 1-2). Après avoir téléchargé le fichier sur votre ordinateur, renommez-le avec le nom `jquery.js` et copiez-le dans le même répertoire que celui de l'atelier actuel.

Ouvrez ensuite la page `index.html` et ajoutez une balise de script faisant référence à la bibliothèque `jquery.js` que nous venons de télécharger et supprimez la balise faisant référence au fichier `fonctionsAjax.js`.

```
<script src="jquery.js" type="text/javascript"></script>
```

Ouvrez ensuite le fichier `fonctionsMachine.js` et commencez par ajouter l'équivalent d'un gestionnaire d'événement `onclick` afin d'appeler la fonction `jouer()` lorsque l'utilisateur clique sur le bouton JOUER.

```
$('#button').click(function () {jouer();});
```

Afin de s'assurer que cette méthode `.click()` est appliquée au bouton uniquement après le chargement des éléments du DOM, nous allons conditionner son initialisation par la méthode `.ready()` (revoir si besoin l'encadré concernant cette méthode en début de ce chapitre).

```
$(document).ready(function() {
  $('#button').click(function () {jouer();});
});
```

Pour information, le gestionnaire d'événement équivalent sans jQuery que nous aurions dû utiliser en JavaScript aurait été alors le suivant.

```
document.getElementById("button").onclick=jouer;
```

Localisez ensuite la fonction `jouer()` qui va être appelée lors de l'envoi de la requête Ajax au serveur. Supprimez tout le contenu de cette fonction puis insérez l'appel de la méthode Ajax de jQuery.

```
$.ajax({ … });
```

Ajoutez ensuite à l'intérieur des accolades de cette méthode les options nécessaires à la configuration de la requête.

Il s'agit de l'option `type` pour commencer avec la valeur `POST` (par défaut la valeur de l'option `type` étant `GET`) afin d'indiquer que la requête utilise la méthode HTTP `POST`.

```
type: 'POST',
```

Suivi de l'option `url` qui permet de nommer le fichier qui assure le traitement de la requête coté serveur. Dans le cadre de cet atelier, nous conserverons le même fichier serveur que dans l'atelier 11-1, la valeur de l'option `url` doit donc être `gainAleatoire.php`.

```
url: 'gainAleatoire.php',
```

Puis l'option `data` qui contient les données au format d'URL à envoyer avec la requête. À noter que pour récupérer les valeurs des champs de saisie `nom` et `prenom`, nous allons utiliser aussi deux autres objets jQuery qui permettent de sélectionner les éléments selon leur `id`, puis la méthode `val()` qui permet de récupérer la valeur des éléments ainsi sélectionnés.

```
data: "nom="+ $('#nom').val()+"&"+"prenom="+ $('#prenom') .val(),
```

Les données du serveur étant renvoyées au format Texte nous indiquons explicitement le format utilisé à l'aide de l'option `dataType`.

```
dataType: 'text',
```

Et enfin l'option `success`, qui désigne la fonction de rappel du moteur, comme nous le faisions avec la propriété `onreadystatechange` dans les moteurs avant d'utiliser jQuery.

```
success: actualiserPage,
```

À noter qu'il est aussi possible d'ajouter l'option `error` afin de traiter les éventuels problèmes de serveur comme nous le faisions déjà en testant la valeur de `status` dans la fonction de rappel du moteur de l'atelier 11-1.

```
error: function() {alert('Erreur serveur');}
```

Une fois toutes les options paramétrées, la fonction `jouer()` doit être semblable au code 14-23 ci-dessous :

Code 14-23 : fonction `jouer()` avec jQuery :

```
function jouer() {
  $.ajax({
    type: 'POST',
    url: 'gainAleatoire.php',
```

```
    data: "nom="+ $('#nom').val()+"&"+"prenom="+ $('#prenom') .val(),
    dataType: 'text',
    success: actualiserPage,
    error: function() {alert('Erreur serveur');}
  });
}
```

Pour information, nous indiquons ci-dessous (code 14-24) la fonction `jouer()` équivalente sans jQuery. Nous vous invitons à comparer ces instructions avec celles du code 14-23 dans lequel nous avons utilisé les méthodes de la bibliothèque jQuery.

Code 14-24 : fonction `jouer()` sans jQuery :

```
function jouer() {
 objetXHR = creationXHR();
 var parametres = "nom="+ codeContenu("nom")+"&"+"prenom="+ codeContenu("prenom");
 objetXHR.open("post","gainAleatoire.php", true);
 objetXHR.onreadystatechange = actualiserPage;
 objetXHR.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
 objetXHR.send(parametres);
}
```

La fonction de création et de configuration de la requête Ajax est maintenant terminée, mais il nous reste encore à créer la fonction de rappel `actualiserPage()` pour que le système soit opérationnel. Pour cela, nous allons aussi commencer par supprimer le contenu de la fonction de rappel que nous avons récupéré de l'atelier 11-1.

```
function actualiserPage() {
 …
}
```

Comme nous l'avons indiqué dans la présentation du moteur Ajax avec jQuery, la valeur de la réponse du serveur est envoyée automatiquement dans l'argument de la fonction de rappel, il convient donc de l'ajouter entre les parenthèses.

```
function actualiserPage(reponse) {
 …
}
```

Le programme serveur étant identique à celui de l'atelier 11-1, les données sont donc envoyées au format texte et séparées par le symbole « : ». Il faut donc filtrer la réponse du serveur à l'aide de la fonction `split(":")` pour récupérer les deux informations (le résultat et le nom du gagnant) dans un tableau de variables `nouveauResultat` comme nous l'avions déjà fait dans l'atelier 11-1.

```
function actualiserPage(reponse) {
 var nouveauResultat = reponse.split(":");
 …
}
```

Une fois les données disponibles dans le tableau `nouveauResultat`, il suffit d'utiliser la méthode jQuery `.html()` pour intégrer le résultat du jeu et le nom du gagnant dans la chaîne de caractères `info` (pour mémoire, cette chaîne de caractères est composée de deux balises dont des identifiants sont `resultat` et `gagnant`).

```
$('#resultat').html(nouveauResultat[1]);
$('#gagnant').html(nouveauResultat[0]);
```

La dernière action à effectuer consiste maintenant à rendre visible la chaîne de caractères d'identifiant `info`. Pour cela, nous allons utiliser la méthode jQuery `.css` qui permet de modifier la valeur d'un style. Dans notre cas, nous allons remplacer la valeur `hidden` du style `visibility` de l'élément `info` par la valeur `visible`.

```
$('#info').css("visibility", "visible");
```

À noter que si nous avions utilisé initialement le style `display` à la place de `visibility` pour gérer l'affichage de l'élément `info` dans la feuille de styles (la directive équivalente de la règle de styles `#info` aurait été alors `display: none`) nous aurions pu utiliser avantageusement les méthodes jQuery `.show()` et `.hide()` comme l'illustre le code ci-dessous (pour plus d'informations sur ces deux méthodes vous pouvez consulter l'API référence Effects de la documentation jQuery en ligne : *http://docs.jquery.com/Effects*).

```
$('#info').show();
```

Une fois terminée, la fonction `actualiserPage()` doit être semblable au code 14-25 ci-dessous.

Code 14-25 : fonction `actualiserPage()` avec jQuery :

```
function actualiserPage(reponse) {
   var nouveauResultat = reponse.split(":");
   $('#resultat').html(nouveauResultat[1]);
   $('#gagnant').html(nouveauResultat[0]);
   $('#info').css("visibility", "visible");
}
```

Pour information, nous indiquons ci-dessous (code 14-26) la fonction `actualiserPage()` équivalente n'utilisant pas jQuery. Nous vous invitons à comparer ces instructions avec celles du code 14-25 dans lequel nous avons utilisé les méthodes de la bibliothèque jQuery.

Code 14-26 : fonction `actualiserPage()` sans jQuery :

```
function actualiserPage() {
  if (objetXHR.readyState == 4) {
    if (objetXHR.status == 200) {
      var nouveauResultat = objetXHR.responseText.split(":");
      remplacerContenu("resultat", decodeURI(nouveauResultat[1]));
      remplacerContenu("gagnant", decodeURI(nouveauResultat[0]));
      document.getElementById("info").style.visibility="visible";
    }
  }
}
```

Le système est maintenant opérationnel. Cependant, dans l'atelier 11-1, nous avions ajouté des instructions de gestion de l'affichage de l'animation et du blocage du bouton JOUER pendant le temps de traitement serveur de la requête Ajax. Avec jQuery cela est d'autant plus facile de gérer ce genre de fonctionnalité qu'il existe deux méthodes dédiées à détecter le début et la fin du traitement serveur (`ajaxStart()` et `ajaxStop()`).

Pour la gestion de l'affichage de l'animation, nous devons coupler ces deux méthodes avec la méthode jQuery `.css()` que nous avons utilisée précédemment comme l'illustre le code ci-dessous :

```
$('#charge').ajaxStart(function(request, settings) { $(this).css("visibility", "visible") });
$('#charge').ajaxStop(function(request, settings){ $(this).css("visibility", "hidden") });
```

En ce qui concerne le blocage du bouton JOUER, nous allons coupler ces deux mêmes méthodes avec la méthode jQuery `.attr()`, ce qui permet de changer la valeur de l'attribut `disabled` de l'élément dont l'identifiant est `#button`.

```
$('#button').ajaxStart(function(request, settings) { $(this).attr("disabled",true) });
$('#button').ajaxStop(function(request, settings){ $(this).attr("disabled",false) });
```

Pour que ces méthodes soient appliquées aux éléments concernés après leur chargement, nous allons insérer ces 4 instructions dans le gestionnaire `ready()` (qui détecte la fin du chargement des éléments DOM de la page) déjà créé pour le gestionnaire `onclick` du bouton JOUER.

Code 14-27 : gestion de l'animation et du bouton avec jQuery :

```
$(document).ready(function() {
 $('#button').click(function () {jouer();});
 //gestion du bouton et de l'animation
 $('#button').ajaxStart(function(request, settings) { $(this).attr("disabled",true) });
 $('#button').ajaxStop(function(request, settings){ $(this).attr("disabled",false) });
 $('#charge').ajaxStart(function(request, settings) { $(this).css("visibility", "visible") });
 $('#charge').ajaxStop(function(request, settings){ $(this).css("visibility", "hidden") });
});
```

Pour information, nous indiquons ci-dessous (code 14-28) les instructions placées dans les fonctions `jouer()` et `actualiserPage()` qui permettaient de gérer l'animation et le bouton avant d'utiliser jQuery. Nous vous invitons à comparer ces instructions avec celles du code 14-27 dans lequel nous avons utilisé les méthodes de la bibliothèque jQuery.

Code 14-28 : gestion de l'animation et du bouton avant d'utiliser jQuery :

```
function jouer() {
  objetXHR = creationXHR();
  …
  document.getElementById("button").disabled= false;
  document.getElementById("charge").style.visibility="hidden";
}
function actualiserPage() {
  …
  document.getElementById("button").disabled= false;
  document.getElementById("charge").style.visibility="hidden";
}
```

Les modifications du fichier `fonctionsMachine.js` sont maintenant terminées, vous pouvez l'enregistrer et passer à la phase de test du système.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 de Dreamweaver. Le système doit se comporter comme celui de l'atelier 11-1 hormis que cette fois le moteur Ajax fonctionne avec l'objet et les méthodes jQuery que nous venons de mettre en place.

Pour mémoire, pour tester le système vous devez saisir votre nom et prénom dans les champs appropriés puis cliquer sur le bouton JOUER. Dès l'envoi de la requête, le bouton JOUER est bloqué (il est alors grisé) et l'animation apparaît afin de vous signaler que le traitement est en cours. À la fin du traitement, le bouton JOUER doit retrouver son

apparence initiale, l'animation doit disparaître et le message d'information rappelant votre nom et prénom et indiquant le montant de votre gain doit s'afficher.

Atelier 14-2 : requête asynchrone POST et réponse au format JSON avec jQuery

Composition du système

Pour continuer notre comparatif entre les moteurs Ajax en JavaScript et en jQuery, nous allons maintenant reprendre le système de l'atelier 12-3 dans lequel la réponse du serveur était renvoyée au format JSON.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure est identique à celle de l'atelier 12-3 ;
- du fichier de la bibliothèque jQuery (`jquery.js`) qui est identique à celui que nous avions téléchargé dans l'atelier 14-1 ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est semblable à celle de l'atelier 12-3 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 12-3 ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 12-3 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du jeu est identique à celui de l'atelier 12-3 que nous allons prendre comme base de départ pour effectuer nos modifications.

Conception du système

Ouvrez la page HTML de l'atelier 12-3 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap14/atelier14-2/`. Copiez ensuite les autres fichiers de l'atelier dans ce nouveau dossier ainsi que le fichier de la bibliothèque jQuery (`jquery.js`) qui est copié depuis le répertoire de l'atelier précédent.

Ouvrez ensuite la page `index.html` et ajoutez une balise de script faisant référence à la bibliothèque `jquery.js` et supprimez la balise faisant référence au fichier `fonctions-Ajax.js`.

```
<script src="jquery.js" type="text/javascript"></script>
```

Ouvrez ensuite le fichier `fonctionsMachine.js` et ajoutez la méthode jQuery `.click` du bouton JOUER afin d'appeler la fonction `jouer()` lorsque le joueur clique sur ce bouton comme nous l'avons fait dans le précédent atelier.

```
$(document).ready(function() {
 $('#button').click(function () {jouer();});
});
```

Localisez ensuite la fonction `jouer()` qui est appelée lors de l'envoi de la requête Ajax au serveur. Supprimez tout le contenu de cette fonction puis insérez l'appel de la méthode Ajax de jQuery.

```
$.ajax({ … });
```

Ajoutez ensuite à l'intérieur des accolades de cette méthode les options nécessaires à la configuration de le requête.

Les deux premières options sont identiques à celles du moteur Ajax jQuery de l'atelier précédent.

```
type: 'POST',
url: 'gainAleatoire.php',
```

L'option suivante est `data`, elle contient les données au format d'URL à envoyer avec la requête. Dans cet atelier, comme nous envoyons uniquement au serveur le nom du joueur, nous n'avons besoin que de la valeur du champ `nom`.

```
data: "nom="+ $('#nom').val(),
```

Contrairement à l'atelier précédent dans lequel nous avions réceptionné les données du serveur au format Texte, cette fois-ci nous allons devoir indiquer explicitement le format JSON à l'aide de l'option `dataType`.

```
dataType: 'json',
```

Et enfin, les options `success` et `error` sont elles aussi, identiques à celles de l'atelier précédent.

```
success: actualiserPage,
error: function() {alert('Erreur serveur');}
```

Une fois toutes les options paramétrées, la fonction `jouer()` doit être semblable au code 14-29 ci-dessous.

Code 14-29 : Fonction `jouer()` :

```
function jouer() {
  $.ajax({
   type: 'POST',
   url: 'gainAleatoire.php',
   data: "nom="+ $('#nom').val(),
   dataType: 'json',
   success: actualiserPage,
   error: function() {alert('Erreur serveur');}
   });
}
```

La fonction `jouer()` étant maintenant terminée, passons à la création de la fonction de rappel `actualiserPage()`. Commencez par supprimer le contenu de la fonction de rappel comme dans l'atelier précédent et ajoutez la réponse du serveur dans son argument.

```
function actualiserPage(reponse) {
 …
}
```

Gestion des objets et tableaux JavaScript avec la méthode .each()

La méthode jQuery `.each()` permet de récupérer les valeurs des attributs d'un objet JavaScript. Pour cela, il suffit d'indiquer le nom de l'objet dans le premier argument et la fonction qui permet de parcourir tous les attributs de l'objet dans le second argument de la méthode.

Par exemple, si nous désirons récupérer les deux attributs de l'objet ci-dessous :

```
var objetJS={nom:'Defrance',prenom:'Jean-Marie'}
```

Il suffit de créer un objet jQuery et de configurer les arguments de sa méthode `.each()` de la manière suivante :

```
$.each(objetJS, function(attribut,valeur) {
 console.info(attribut+' est égal à '+valeur);
});
```

Cette commande affiche dans la console les lignes ci-dessous :

```
nom est égal à Defrance
prenom est égal à Jean-Marie
```

À noter que cette même méthode `.each()` peut aussi être utilisée pour récupérer les données d'un tableau indicé en JavaScript. La procédure est la même que pour un objet hormis le fait que la fonction insérée dans le second argument n'a pas de paramètre, les valeurs du tableau étant récupérées à l'aide de la clé `this` comme l'illustre l'exemple ci-dessous.

```
var tableauJS = [ '1', '2'];
$.each(tableauJS, function() {
 console.info('valeur='+this);
});
```

Ce programme afficherait alors dans la console :

```
valeur=1
valeur=2
```

L'utilisation conjointe de ces deux exploitations de la méthode `.each()` permet ainsi d'extraire les données d'un objet JSON quelle que soit sa structure.

Pour mémoire, l'objet JSON renvoyé par le serveur a une structure semblable à celle du code 14-30 ci-dessous.

Code 14-30 : exemple d'objet JSON renvoyé par le serveur :

```
{"resultats":{"nom":"Defrance","gain":90}}
```

Comme vous pouvez le constater, l'objet JSON est composé de deux objets JavaScript. Nous allons donc utiliser deux méthodes jQuery `.each()` imbriquées (voir l'encadré précédent) afin d'en extraire les données.

Pour récupérer ces données, nous allons commencer par initialiser un tableau `resultats` qui contient les valeurs du gain et le nom du joueur.

```
function actualiserPage(reponse) {
 var resultats=new Array();
```

La première méthode permet d'accéder au premier objet `resultats`.

```
$.each(reponse, function(nomRes,res) {
…
});
```

Avec la seconde boucle, nous pouvons ensuite mémoriser les deux données `nom` et `gain` dans le tableau précédemment initialisé.

```
$.each(res, function(attribut,valeur) {
 resultats[attribut]=valeur;
});
```

Une fois les deux méthodes `.each()` configurées, il faut ensuite affecter les résultats mémorisés dans le tableau `resultats[]` aux deux zones dont les identifiants sont respectivement `nom` et `gain`.

```
$('#resultat').html(resultats["gain"]);
$('#gagnant').html(resultats["nom"]);
```

Pour mémoire, la structure du message d'information dans lequel nous venons d'intégrer les nouveaux résultats est la suivante :

```
<div id="info">Bravo, M <span id="gagnant"></span> vous avez gagné
➥<span id="resultat"></span> Euros</div>
```

Ce nouveau message d'information étant désormais prêt, il ne nous reste plus qu'à l'afficher en changeant le style `visibility` avec la valeur `visible` comme nous l'avions déjà fait dans l'atelier précédent.

Une fois terminée, la fonction `actualiserPage()` doit être semblable au code 14-31.

Code 14-31 : fonction `actualiserPage()` :

```
function actualiserPage(reponse) {
  var resultats=new Array();
  $.each(reponse, function(nomRes,res) {
   $.each(res, function(attribut,valeur) {
    resultats[attribut]=valeur;
   });
  });
  $('#resultat').html(resultats["gain"]);
  $('#gagnant').html(resultats["nom"]);
  //affiche la zone info
  $('#info').css("visibility", "visible");
}
```

Comme dans l'atelier précédent nous allons, ici aussi, gérer avec les méthodes jQuery `ajaxStart()` et `ajaxStop()` l'affichage de l'animation et le blocage du bouton JOUER pendant le temps de traitement de la requête Ajax.

Code 14-32 : gestion de l'animation et du bouton avec jQuery :

```
$(document).ready(function() {
 $('#button').click(function () {jouer();});
 //gestion de l'animation et du bouton
 $('#button').ajaxStart(function(request, settings) { $(this).attr("disabled",true) });
 $('#button').ajaxStop(function(request, settings){ $(this).attr("disabled",false) });
 $('#charge').ajaxStart(function(request, settings) { $(this).css("visibility", "visible") });
 $('#charge').ajaxStop(function(request, settings){ $(this).css("visibility", "hidden") });
});
```

Les modifications du fichier `fonctionsMachine.js` sont maintenant terminées, vous pouvez l'enregistrer et passer à la phase de test du système.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Le système doit se comporter comme celui de l'atelier 12-3 hormis que cette fois le moteur Ajax fonctionnera avec l'objet et les méthodes jQuery que nous venons de mettre en place.

Atelier 14-3 : requête asynchrone POST et réponse au format XML avec jQuery

Composition du système

Pour continuer notre comparatif entre les moteurs Ajax en JavaScript et en jQuery, nous vous proposons maintenant de reprendre le système de l'atelier 12-2 dans lequel la réponse du serveur était renvoyée au format XML.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure est identique à celle de l'atelier 12-2 ;
- du fichier de la bibliothèque jQuery (`jquery.js`) qui est identique à celui que nous avions téléchargé dans l'atelier 14-1 ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est semblable à celle de l'atelier 12-2 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) dont la base est identique à celle de l'atelier 12-2 ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 12-2 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du jeu est identique à celui de l'atelier 12-2 que nous allons prendre comme base de départ pour effectuer nos modifications.

Conception du système

Ouvrez la page HTML de l'atelier 12-2 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap14/atelier14-3/`. Copiez ensuite les autres fichiers de l'atelier dans ce nouveau dossier ainsi que le fichier de la bibliothèque jQuery (`jquery.js`) qui est copié depuis le répertoire de l'atelier précédent.

Ouvrez ensuite la page `index.html` et ajoutez une balise de script faisant référence à la bibliothèque `jquery.js` et supprimez la balise faisant référence au fichier `fonctionsAjax.js`.

```
<script src="jquery.js" type="text/javascript"></script>
```

Ouvrez ensuite le fichier `fonctionsMachine.js` et ajoutez la méthode jQuery `.click` du bouton JOUER afin d'appeler la fonction `jouer()` lorsque le joueur clique sur ce bouton, comme nous l'avons fait dans le précédent atelier.

```
$(document).ready(function() {
 $('#button').click(function () {jouer();});
});
```

Localisez ensuite la fonction `jouer()` qui est appelée lors de l'envoi de la requête Ajax au serveur. Supprimez tout le contenu de cette fonction puis insérez l'appel de la méthode Ajax de jQuery.

```
$.ajax({ … });
```

Ajoutez ensuite à l'intérieur des accolades de cette méthode les options nécessaires à la configuration de le requête. Les trois premières options seront identiques à celles du moteur Ajax jQuery des ateliers précédents.

```
type: 'POST',
url: 'gainAleatoire.php',
data: "nom="+ $('#nom').val(),
```

Contrairement à l'atelier précédent dans lequel nous avions réceptionné les données du serveur au format JSON, cette fois-ci nous allons devoir indiquer explicitement le format XML à l'aide de l'option `dataType`.

```
dataType: 'xml',
```

Et enfin, les options `success` et `error` sont elles aussi identiques à celles des ateliers précédents.

```
success: actualiserPage,
error: function() {alert('Erreur serveur');}
```

Une fois toutes les options paramétrées, la fonction `jouer()` doit être semblable au code 14-33 ci-dessous :

Code 14-33 : fonction `jouer()` :

```
function jouer() {
 $.ajax({
 type: 'POST',
 url: 'gainAleatoire.php',
 data: "nom="+ $('#nom').val(),
 dataType: 'xml',
 success: actualiserPage,
 error: function() {alert('Erreur serveur');}
 });
}
```

La fonction `jouer()` étant maintenant terminée, passons à la création de la fonction de rappel `actualiserPage()`. Commencez par supprimer le contenu de la fonction de rappel comme dans l'atelier précédent et ajoutez la réponse du serveur dans son argument.

```
function actualiserPage(reponse) {
 …
}
```

Pour mémoire, l'objet XML renvoyé par le serveur a une structure semblable à celle du code 14-34 ci-dessous.

Code 14-34 : exemple d'objet XML renvoyé par le serveur :

```
<?xml version="1.0" encoding="utf-8"?>
<resultats>
    <nom>Defrance</nom>
    <gain>74</gain>
</resultats>
```

Comme vous pouvez le constater, l'objet XML est composé d'un nœud racine `<resultats>` et de deux nœuds enfants `<nom>` et `<gain>`. Pour récupérer les données des nœuds enfants nous allons utiliser une méthode jQuery `.find()` couplée à la méthode `.text()` pour récupérer le contenu de la balise trouvée.

Si nous appliquons la méthode `.find()` à l'élément du résultat renvoyé par le serveur (`reponse`) en indiquant en paramètre le nom du nœud enfant recherché, nos pouvons récupérer le contenu du nœud ainsi ciblé et l'enregistrer dans une variable comme l'illustre la première ligne de code de la fonction ci-dessous.

```
function actualiserPage(reponse) {
 var gain=$(reponse).find('gain').text();
```

Une fois la valeur mémorisée dans la variable `gain` il suffit ensuite de l'affecter à la zone dont l'identifiant est `resultat` à l'aide de la méthode `html()`.

```
$('#resultat').html(gain);
```

Pour récupérer le nom du joueur, la procédure est semblable hormis que dans ce cas, le nom du nœud recherché est `nom` et que sa valeur est ensuite affectée à la zone dont l'identifiant est `gagnant`.

```
var gagnant=$(reponse).find('nom').text();
 $('#gagnant').html(gagnant);
```

Le nouveau message d'information étant désormais prêt, il ne nous reste plus qu'à l'afficher en donnant au style `visibility` la valeur `visible` comme nous l'avions déjà fait dans l'atelier précédent.

```
$('#info').css("visibility", "visible");
```

Une fois terminée, la fonction `actualiserPage()` doit être semblable au code 14-35.

Code 14-35 : fonction `actualiserPage()` :

```
function actualiserPage(reponse) {
 var gain=$(reponse).find('gain').text();
 $('#resultat').html(gain);
 var gagnant=$(reponse).find('nom').text();
 $('#gagnant').html(gagnant);
 //affiche la zone info
 $('#info').css("visibility", "visible");
}
```

Comme dans l'atelier précédent nous allons, ici aussi, gérer avec les méthodes jQuery `ajaxStart()` et `ajaxStop()` l'affichage de l'animation et le blocage du bouton JOUER pendant le temps de traitement de la requête Ajax.

Gestion de document XML avec de multiples items

Dans les documents XML, il est très fréquent d'avoir une structure composée de multiples items comme l'illustre l'exemple ci-dessous.

```
<resultats>
  <gain>74</gain>
  <gain>23</gain>
  <gain>59</gain>
</resultats>
```

Dans ce cas, vous devez exploiter une méthode `each()` afin de parcourir tous les items du noeud racine. En guise d'exemple, voici un code qui permet de récupérer dans un tableau les différents gains du document ci-dessus.

```
var resultat=new Array()
$(reponse).find('gain').each(function() {
 resultat[]= $(this).text();
});
```

Code 14-36 : gestion de l'animation et du bouton avec jQuery :

```
$(document).ready(function() {
 $('#button').click(function () {jouer();});
 //gestion de l'animation et du bouton
 $('#button').ajaxStart(function(request, settings) { $(this).attr("disabled",true) });
 $('#button').ajaxStop(function(request, settings){ $(this).attr("disabled",false) });
 $('#charge').ajaxStart(function(request, settings) { $(this).css("visibility", "visible") });
 $('#charge').ajaxStop(function(request, settings){ $(this).css("visibility", "hidden") });
});
```

Les modifications du fichier `fonctionsMachine.js` sont maintenant terminées, vous pouvez l'enregistrer et passer à la phase de test du système.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Le système doit se comporter comme celui de l'atelier 12-2 hormis que cette fois le moteur Ajax fonctionne avec l'objet et les méthodes jQuery que nous venons de mettre en place.

Atelier 14-4 : vérification instantanée de la saisie dans une base de données avec jQuery

Composition du système

Pour clôturer notre comparatif entre les moteurs Ajax en JavaScript et en jQuery, nous vous proposons maintenant de réaliser un système de vérification de la saisie semblable à celui de l'atelier 13-1. Cela nous permet ainsi d'illustrer la gestion d'un événement (en l'occurrence il s'agit de la saisie d'un nouveau caractère) avec l'objet et les méthodes jQuery.

Cette structure est composée :

- d'une page HTML (`index.html`) dont la structure est identique à celle de l'atelier 13-1 ;
- du fichier de la bibliothèque jQuery (`jquery.js`) qui est identique à celui que nous avions téléchargé dans l'atelier 14-1 ;
- d'un fichier JS (`fonctionsMachine.js`) qui contient les fonctions spécifiques à l'application Ajax de la machine à sous dont la structure de base avant modification est semblable à celle de l'atelier 13-1 ;
- d'un fichier serveur PHP (`gainAleatoire.php`) identique à celui de l'atelier 13-1 ;
- d'un fichier serveur PHP (`nomVerification.php`) identique à celui de l'atelier 13-1 ;
- d'un fichier de connexion à la base de données (`connexionMysql.php`) identique à celui de l'atelier 13-1 ;
- d'une feuille de styles (`style.css`) identique à celle de l'atelier 13-1 ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du jeu est identique à celui de l'atelier 13-1 que nous allons prendre comme base de départ pour effectuer nos modifications.

Conception du système

Ouvrez la page HTML de l'atelier 13-1 (`index.html`) et sauvegardez-la sous le même nom dans un nouveau répertoire nommé `/chap14/atelier14-4/`. Copiez ensuite les autres fichiers de l'atelier dans ce nouveau dossier ainsi que le fichier de la bibliothèque jQuery (`jquery.js`) qui est copié depuis le répertoire de l'atelier précédent.

Ouvrez ensuite la page `index.html` et ajoutez une balise de script faisant référence à la bibliothèque `jquery.js` et supprimez la balise faisant référence au fichier `fonctions-Ajax.js`.

```
<script src="jquery.js" type="text/javascript"></script>
```

Ouvrez ensuite le fichier `fonctionsMachine.js`. Pour mémoire, ce fichier comporte deux moteurs Ajax. Le premier permet d'envoyer une requête accompagnée du nom du joueur au fichier serveur `gainAleatoire.php` afin de récupérer le montant du gain et le nom du gagnant. Le second, quant à lui, permet d'interroger la base de données via PHP et une requête Ajax déclenchée à chaque nouvelle saisie d'un caractère dans le champ `nom` au fichier serveur `nomVerification.php` afin de récupérer un état indiquant si le nom est présent ou pas dans la base.

Pour le premier moteur, son adaptation en jQuery est semblable à celle que nous avons déjà détaillée dans l'atelier 14-1 (hormis que dans notre cas, seule la valeur du champ `nom` est envoyée en paramètre de la requête Ajax). Nous nous contentons donc de remplacer les fonctions `jouer()` et `actualiserPage()` par celles du fichier `fonctionsMachine.js` de l'atelier 14-1 et d'ajouter les gestionnaires du bouton et de l'animation (voir code 14-37). À noter que pour préparer le blocage du bouton par le second moteur nous avons placé son instruction d'activation dans la fonction `actualiserPage()` (alors que nous avions utilisé initialement une méthode `ajaxStop()` dans le gestionnaire `ready()`).

Code 14-37 : fonctions `jouer()` et `actualiserPage()` avec jQuery :

```
function jouer() {
 $.ajax({
    type: 'POST',
    url: 'gainAleatoire.php',
    data: "nom="+ $('#nom').val(),
    dataType:'text',
    success: actualiserPage,
    error: function() {alert('Erreur serveur');}
 });
}
function actualiserPage(reponse) {
   var nouveauResultat = reponse.split(":");
   $('#resultat').html(nouveauResultat[1]);
   $('#gagnant').html(nouveauResultat[0]);
   $('#button').attr("disabled",false);
 $('#info').css("visibility", "visible");
}
$(document).ready(function() {
  $('#button').click(function () {jouer();});
  //gestion du bouton
  $('#button').ajaxStart(function(request, settings) { $(this).attr("disabled",true) });
  //gestion de l'animation
  $('#charge').ajaxStart(function(request, settings) { $(this).css("visibility", "visible") });
  $('#charge').ajaxStop(function(request, settings){ $(this).css("visibility", "hidden") });
});
```

Si vous le désirez, vous pouvez dès maintenant tester le bon fonctionnement de ce premier moteur dans le navigateur avant de passer à la création du second moteur (attention dans ce cas à neutraliser les instructions du second moteur en les commentant afin d'éviter l'affichage d'erreurs lors de vos tests).

Passons maintenant à la création du second moteur. Commençons par mettre en place le gestionnaire `.bind()` qui assure la surveillance de la saisie d'un caractère dans le champ `nom`.

```
$(document).ready(function() {
  $('#nom').bind('keyup', null, verifierNom);
  …
}
```

Ce gestionnaire appelle ainsi la fonction `verifierNom()` à chaque nouveau caractère saisi dans le champ `nom`. La fonction `verifierNom()` réceptionne donc en paramètre (`event`) l'événement correspondant.

```
function verifierNom(event)
{
```

Dans l'atelier 13-1, la requête Ajax utilisée pour assurer la vérification de la saisie était une requête GET. Aussi, pour ce second moteur jQuery, nous allons utiliser le même type de requête (nous devons donc transmettre cette fois les paramètres directement dans l'URL du serveur).

Pour transmettre les paramètres, nous allons commencer par les préparer dans une variable `cible`. Les deux premières lignes de la fonction de ce moteur permettent de construire cette URL en y insérant la valeur du champ `nom` (extraite de l'événement `event` en utilisant les attributs `.target.value`).

```
function verifierNom(event)
{
 var cible ="nomVerification.php?";
 cible +="nom="+ event.target.value;
```

La création et la configuration de l'objet jQuery Ajax est ensuite semblable à celles que nous avons déjà réalisées précédemment hormis le fait que le type est GET et que les paramètres sont intégrés dans l'URL du serveur (grâce à la variable `cible`).

```
$.ajax({
    type: 'GET',
    url: cible,
    dataType:'text',
    success: afficherReponse,
    error: function() {alert('Erreur serveur');}
   }); }
```

Une fois terminée, la fonction `verifierNom()` doit être semblable aux instructions du code 14-38.

Code 14-38 : fonction `verifierNom()` :

```
function verifierNom(event) {
  var cible ="nomVerification.php?";
  cible +="nom="+ event.target.value;
    $.ajax({
       type: 'GET',
       url: cible,
       dataType:'text',
       success: afficherReponse,
       error: function() {alert('Erreur serveur');}
      }); }
```

Il nous reste maintenant la création de la fonction de rappel `afficherReponse()`. Cette fonction doit réceptionner la réponse du serveur et, selon sa valeur, afficher un message informant l'utilisateur qu'il a été reconnu ou non. Si l'utilisateur est reconnu, nous devons aussi réactiver le bouton de sorte à ce qu'il puisse jouer.

```
function afficherReponse(reponse) {
  if(reponse!=0) {
    … //utilisateur reconnu
  }else{
   … //utilisateur inconnu
  }
}
```

Pour afficher le message nous allons utiliser une instruction jQuery qui affecte à la zone dont l'identifiant est `message` le texte correspondant au résultat du serveur à l'aide de la méthode `html()`. Nous allons utiliser ensuite le principe des méthodes en chaîne de jQuery pour ajouter à cette même instruction une autre méthode `css()` qui permet de rendre le message visible et de l'afficher dans la couleur désirée (rouge si l'utilisateur est inconnu, vert sinon).

```
$('#message').html("Joueur identifié").css({"visibility"
➥:"visible","color":"green"});
```

Une seconde instruction permettant d'activer ou de désactiver le bouton JOUER accompagne chacune des deux alternatives de cette structure de choix.

```
$('#button').attr("disabled",false);
```

Une fois terminée, la fonction `afficherReponse()` doit être semblable au code 14-39 :

Code 14-39 : fonction `afficheReponse()` :

```
function afficherReponse(reponse) {
  if(reponse!=0) {
    $('#message').html("Joueur identifié").css({"visibility"
    ➥:"visible","color":"green"});
    $('#button').attr("disabled",false);
   }else{
    $('#message').html("Joueur inconnu").css({"visibility":"visible","color":"red"});
    $('#button').attr("disabled",true);
  }
}
```

Les modifications du fichier `fonctionsMachine.js` sont maintenant terminées, vous pouvez l'enregistrer et passer à la phase de test du système.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Le système doit se comporter comme celui de l'atelier 13-1 hormis que cette fois le moteur Ajax fonctionne avec les objets et méthodes jQuery que nous venons de mettre en place.

15

Plug-ins jQuery

Ce dernier chapitre de la troisième partie vient compléter le chapitre 14 sur jQuery en vous présentant quelques exemples de plug-ins jQuery qui trouveront certainement une place dans vos futures applications.

Les plug-ins jQuery sont des bibliothèques complémentaires à la bibliothèque de base que nous avons utilisée dans le chapitre précédent. D'installation très facile, ils vous permettront, en quelques lignes de code, de disposer d'applications avancées que vous pourrez personnaliser à votre guise.

Mise en œuvre d'un plug-in jQuery

Localisation des plug-ins jQuery

La sélection des 3 plug-ins jQuery présentés dans ce chapitre est évidemment très loin d'être exhaustive. Le but de ce chapitre est principalement de vous initier à la mise en œuvre de quelques plug-ins afin que vous puissiez ensuite appliquer la même procédure pour en installer d'autres selon les besoins de vos futures applications.

Vous trouverez une liste conséquente de plug-ins jQuery à l'adresse ci-dessous. Vous pourrez les classer selon leur famille (Ajax, Effets graphiques, Menus de navigation, Widget…), leur nom, leur date ou encore selon leur popularité.

http://jquery.com/plugins/

Comment installer un plug-in jQuery ?

L'installation d'un plug-in jQuery est très simple. Vous devez commencer par télécharger les ressources du plug-in jQuery sur votre ordinateur puis vous devez placer un ou plusieurs fichiers JavaScript et CSS dans un répertoire du site en cours de développement.

Il faut ensuite ajouter des balises `<script>` ou `<link>` pour référencer les ressources nécessaires dans la page sur laquelle vous désirez mettre en place l'application (voir exemple du code 15-1). Evidemment, le fichier de la bibliothèque de base `jquery.js` doit être préalablement référencé de la même manière dans cette même page.

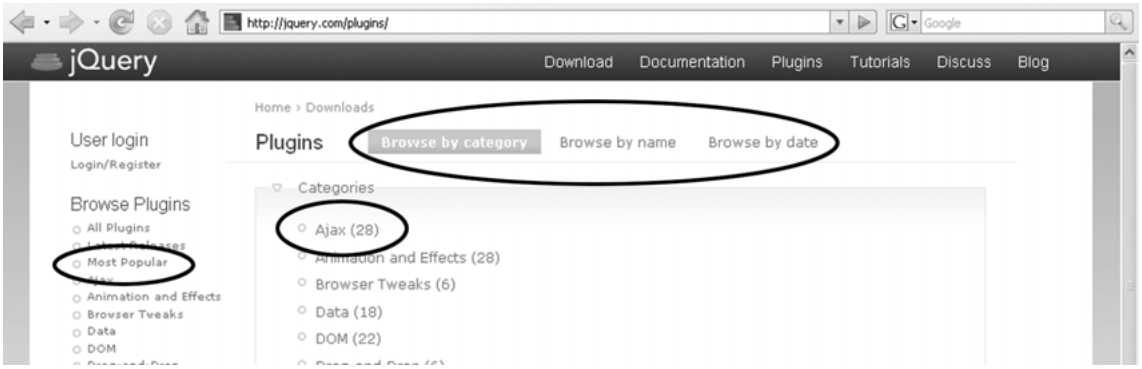


Figure 15-1

Annuaire des plug-ins jQuery

Code 15-1 : exemple de balises de référence à des fichiers JavaScript et CSS d'un plug-in jQuery :

```
<link rel="stylesheet" href="ui.tabs.css" type="text/css" />
<script src="ui.tabs.js" type="text/javascript"></script>
```

Une fois les ressources disponibles dans la page, il ne vous reste plus qu'à créer un objet jQuery, en référençant l'élément qui accueille l'application à mettre en place, pour ensuite lui appliquer une des méthodes du plug-in jQuery (voir exemple du code 15-2).

Code 15-2 : exemple d'appel d'une méthode de plug-in jQuery :

```
$(document).ready(function() {
 $('#menuOnglet ul').tabs({ remote: true });
});
```

Dans l'exemple du code ci-dessus, le sélecteur utilisé (`#menuOnglet ul`) fait référence aux balises `<ul>` enfants de l'élément portant l'identifiant `menuOnglet`. L'application concernée (en l'occurrence ici, un menu à onglets) prend donc place sur ces balises ainsi ciblées (voir exemple du code 15-3).

Code 15-3 : exemple de balises qui supportent l'application du plug-in jQuery :

```
<div id="menuOnglet">
  <ul>
   <li><a href="infoCss.html"><span>CSS</span></a></li>
   <li><a href="infoXml.html"><span>XML</span></a></li>
  </ul>
</div>
```

Atelier 15-1 : plug-in UI Tabs : menu à onglets

Composition du système

Ce plug-in permet de mettre en œuvre un système de navigation dynamique par onglets. Dans notre exemple, le contenu de chaque onglet est stocké dans une page HTML spécifique mais il pourrait tout aussi bien être généré dynamiquement par un fichier PHP, si besoin.

Cette structure est composée :

- d'une page HTML (`index.html`) qui contient l'application ;
- du fichier de la bibliothèque de base jQuery (`jquery.js`) ;
- du fichier de la bibliothèque du plug-in UI Tabs (`ui.tabs.js`) ;
- du fichier de la feuille de styles du plug-in UI Tabs (`ui.tabs.css`) ;
- du fichier d'une feuille de styles complémentaire pour le navigateur IE (`ui.tabs-ie.css`) ;
- du fichier d'une feuille de style de personnalisation du contenu de chaque onglet qui est à définir selon la mise en forme désirée (`styleContenu.css`) ;
- d'un fichier élément graphique pour la construction des onglets du menu (`tab.png`) ;
- d'un ensemble de 4 pages contenant le contenu (fragments de codes HTML) de chaque onglet (`infoCss.html`, `infoXml.html`, `infoJavaScript.html`, `infoDom.html`) ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système de navigation est très simple, dès que l'utilisateur clique sur un des onglets du menu, la page HTML correspondante est chargée et les fragments de codes HTML qu'elle contient s'affichent dans la zone située en dessous du menu.

Chaque chargement de la page est signalé par l'affichage d'une animation dans l'onglet. Dès que le chargement d'une page est terminé, la représentation des onglets est modifiée afin d'indiquer quel est l'onglet actuellement actif.

Conception du système

Ouvrez une nouvelle page HTML et sauvegardez-la sous le nom `index.html` dans un nouveau répertoire nommé `/chap15/atelier15-1/`.

Commencez par télécharger le fichier zip regroupant les différentes ressources sur le site de l'auteur (voir adresse ci-dessous).

http://stilbuero.de/jquery/tabs/

Copiez ensuite les différents fichiers nommés dans la composition du système indiquée précédemment dans ce nouveau dossier. À noter que le kit de l'auteur contient un fichier de la bibliothèque de base, vous pouvez donc utiliser ce fichier (dans ce cas renommez le `jquery.js`) ou récupérer celui que nous avons déjà utilisé dans les ateliers du chapitre 14.

Ouvrez ensuite la page `index.html` et ajoutez une balise de `script` faisant référence à la bibliothèque `jquery.js`.

```
<script src="jquery.js" type="text/javascript"></script>
```

Dans ce même fichier, ajoutez ensuite une autre balise de `script` afin de référencer le plug-in `ui.tabs.js`.

```
<script src="ui.tabs.js" type="text/javascript"></script>
```

Puis ajoutez trois liens `link` pour pouvoir disposer des feuilles de styles CSS du plug-in (`ui.tabs.css`) et son complément pour le navigateur Internet Explorer (`ui.tabs-ie.css`)

ainsi que de la feuille de styles personnalisée (`styleContenu.css`) qu'il convient de créer spécialement pour mettre en forme le contenu de chaque onglet.

Ouvrez ensuite une balise `<script>` et insérez un gestionnaire `.ready()` afin de détecter le chargement complet des éléments du DOM.

Code 15-4 : gestionnaire de test du chargement du DOM :

```
<script type="text/javascript">
$(document).ready(function() {
 …
});
</script>
```

Ajoutez à l'intérieur une instruction qui permet de créer un objet jQuery sélectionnant les balises avec lesquelles l'application va être reliée. Dans notre cas, le sélecteur de l'objet référence la (ou les) balise(s) `<ul>` enfant(s) de l'élément dont l'identifiant est `memoOnglet`.

Code 15-5 : ajout de l'instanciation de l'objet jQuery :

```
<script type="text/javascript">
$(document).ready(function() {
 $('#menuOnglet ul');
});
</script>
```

Appliquez ensuite la méthode `tabs()` du plug-in jQuery à l'objet sélecteur précédemment configuré. Cette méthode possédant plusieurs options de configuration, nous allons utiliser dans notre exemple l'option `cache` qui permet de bloquer la mise en mémoire des données.

Code 15-6 : ajout de l'appel de la méthode `tabs()` :

```
<script type="text/javascript">
$(document).ready(function() {
 $('#menuOnglet ul').tabs({ cache: true });
});
</script>
```

Pour terminer la configuration de la page `index.html`, il faut maintenant créer les balises sur lesquelles le plug-in jQuery va s'appliquer. Pour cela, placez-vous dans la balise `body` de la page et ajoutez une balise `div`, son `id` étant configuré avec la valeur `menuOnglet`.

Code 15-7 : création de la zone d'affichage du menu :

```
<div id="menuOnglet">
 …
</div>;
```

Insérez ensuite à l'intérieur de cette balise une structure de listes non ordonnées (balise `<ul>`). Le menu comportera autant d'onglets qu'il y a de balises `<li>` dans cette structure. Dans notre exemple, nous allons ajouter 4 balises `<li>` afin de créer le même nombre d'onglets.

Code 15-8 : ajout des balises `<li>` :

```
<div id="menuOnglet">
 <ul>
   <li>…</li>
   <li>…</li>
   <li>…</li>
```

```
    <li>…</li>
  </ul>
</div>;
```

Ajoutez à l'intérieur de chaque balise `<li>` le nom qui apparaît sur l'onglet. Encadrez ensuite ce nom par un lien hypertexte qui pointe sur le fichier contenant les fragments HTML du contenu de l'onglet concerné.

Code 15-9 : intégration des informations des 4 onglets :

```
<div id="menuOnglet">
 <ul>
   <li><a href="infoCss.html" ><span>CSS</span></a></li>
   <li><a href="infoJavascript.html" ><span>JavaScript</span></a></li>
   <li><a href="infoDom.html" ><span>DOM</span></a></li>
   <li><a href="infoXml.html" ><span>XML</span></a></li>
 </ul>
</div>;
```

Les modifications du fichier `index.html` sont maintenant terminées, vous pouvez l'enregistrer et passer à la création de la feuille de styles dédiée à la mise en forme du contenu. Celle-ci peut être personnalisée selon les contenus mis dans chaque onglet. Nous vous proposons ci-dessous quelques styles très simples pour créer ce fichier en guise d'exemple.

Code 15-10 : création de la feuille de styles `styleContenu.css` :

```
body {
 font-size: 16px;
 font-family: Verdana, Helvetica, Arial, sans-serif;
}
h1 {
 margin: 1em 0 1.5em;
 font-size: 18px;
}
p {
 margin: 0;
 font-size: 12px;
}
```

Enregistrez ce fichier sous le nom `styleContenu.css` avant de passer aux tests du système dans le navigateur.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Le système doit afficher le menu avec ses 4 onglets. Si la structure de la liste s'affiche sans mise en forme, vérifier la présence des différents fichiers JavaScript et CSS indiqués précédemment ainsi que la configuration des différents liens qui les référencent dans la page `index.html`.

Le premier onglet nommé CSS doit être ouvert par défaut. Cliquez ensuite sur l'onglet de votre choix. L'animation `chargeur.gif` doit apparaître dans l'onglet pendant le temps du chargement de la page de l'onglet sélectionné. Dès que le chargement est terminé, le

contenu doit s'afficher dans la zone d'affichage et le menu doit changer d'apparence pour indiquer l'onglet actuellement sélectionné (voir figure 15-2).

Figure 15-2

Test du plug-in UI Tabs : menu à onglets dynamiques

Atelier 15-2 : plug-in jQuery.Suggest : Autosuggestion

Composition du système

Ce plug-in permet de mettre en œuvre un système d'autosuggestion semblable à celui de Google Suggest (revoir la présentation de cette application dans le chapitre 2). Pour la démonstration nous allons utiliser un simple champ de saisie en dessous duquel vont s'afficher les différentes suggestions du serveur selon les caractères saisis (affichage de la liste à partir de 2 caractères). Côté serveur, nous allons utiliser un tableau associatif pour stocker les données qui font l'objet d'une recherche, mais il vous sera très facile par la suite d'adapter ce système à des informations issues d'une base de données.

Cette structure est composée :

- d'une page HTML (`index.html`) qui contient l'application ;
- du fichier de la bibliothèque de base jQuery (`jquery.js`) ;
- du fichier de la bibliothèque du plug-in jQuery.suggest (`jquery.suggest.js`) ;
- du fichier de la feuille de styles du plug-in jQuery.suggest (`jquery.suggest.css`) ;
- du fichier serveur PHP proposé dans la démonstration de l'auteur (`search.php`) ;
- du fichier d'une feuille de styles de personnalisation du contenu de la page de votre application qui est à définir selon la mise en forme désirée (`styleContenu.css`).

Fonctionnement du système

Si vous saisissez au moins deux caractères dans le champ de saisie de l'application, une requête Ajax est envoyée afin de récupérer toutes les données du serveur contenant la chaîne de caractères saisie. Cette liste s'affiche en dessous du champ et il vous est facile de sélectionner la valeur désirée dans les suggestions proposées.

Pour simuler la sélection, une boîte d'alerte affiche la donnée choisie dès que vous avez cliqué sur une des suggestions de la liste.

Conception du système

Ouvrez une nouvelle page HTML et sauvegardez-la sous le nom `index.html` dans un nouveau répertoire nommé `/chap15/atelier15-2/`.

Téléchargez ensuite le fichier zip regroupant les différentes ressources sur le site de l'auteur (cliquez sur le lien Download de la dernière version de la bibliothèque à l'adresse ci-dessous).

http://jquery.com/plugins/project/suggest/

Copiez ensuite dans ce nouveau dossier les différents fichiers composant le système. À noter que le kit de l'auteur contient un fichier de la bibliothèque de base, vous pouvez donc utiliser ce fichier (dans ce cas renommez-le `jquery.js`) ou récupérer celui que nous avons déjà utilisé dans les ateliers du chapitre 14.

Ouvrez la page `index.html` pour y ajouter 2 liens `link` afin de pouvoir disposer des feuilles de styles CSS du plug-in (`jquery.suggest.css`) ainsi que de la feuille de styles personnalisée (`styleContenu.css`) qu'il convient de créer spécialement pour positionner et mettre en forme le contenu de la page de l'application.

Ajoutez ensuite une balise de `script` faisant référence à la bibliothèque de base `jquery.js`.

```
<script src="jquery.js" type="text/javascript"></script>
```

En dessous de cette balise de `script`, ajoutez-en une seconde afin de référencer le plug-in `jquery.suggest.js`.

```
<script src=" jquery.suggest.js" type="text/javascript"></script>
```

Avant de développer le code qui appelle la bibliothèque que nous venons de mettre en place, cliquez sur le lien de la documentation proposée sur la page d'accueil de l'auteur (voir adresse indiquée précédemment). Depuis cette page, vous pouvez trouver un exemple de script pour exploiter rapidement la bibliothèque dont nous allons nous inspirer. À noter qu'en dessous de ce script, vous pouvez aussi télécharger, dans le répertoire de votre atelier, le fichier serveur `search.php` utilisé dans la démonstration en ligne du système (lien « source »).

Dans la page `index.html`, créez une balise `<script>` pour y insérer un gestionnaire `.ready()` afin de détecter le chargement complet des éléments du DOM.

Code 15-11 : ajout du gestionnaire de test du chargement du DOM :

```
<script type="text/javascript">
$(document).ready(function() {
 …
});
</script>
```

Depuis la page de la documentation, copiez les 2 lignes de code de l'objet jQuery Suggest puis collez-les à l'intérieur du gestionnaire `.ready()` (voir code 15-12).

Code 15-12 : ajout du script de création de l'objet jQuery :

```
<script type="text/javascript">
$(document).ready(function() {
  jQuery("#suggest").suggest("search.php",{
    onSelect: function()
     {alert("Votre sélection est : " + this.value)}
   });
});
</script>
```

Ces lignes de code permettent de créer un objet jQuery sélectionnant le champ de saisie que nous allons utiliser (champ dont le `id` est `suggest`) et de lui appliquer la méthode `.suggest()` de la bibliothèque `jquery.suggest.js`.

Dans les arguments de cette méthode, nous devons désigner le fichier serveur qui est destinataire de la requête Ajax, soit `search.php` (attention de ne pas écrire son chemin si le fichier se trouve dans le même répertoire) suivi d'un gestionnaire `onSelect` qui déclenche pour nos tests une boîte d'alerte lors de la sélection d'une des suggestions.

Placez-vous dans la balise `<body>` et ajoutez une structure de page afin de contenir le champ de recherche du formulaire.

Code 15-13 : création d'une structure de page :

```
<div id="page">
 <h2>Autosuggestion jQuery</h2>
  <!--zone du formulaire-->
  <div id="formulaire">
…
  </div>
</div>
```

Placez-vous ensuite à l'intérieur de la balise `<div>` dont l'identifiant est `formulaire` et ajoutez les balises du formulaire et du champ de saisie.

Code 15-14 : Ajout des balises du formulaire et de son champ de saisie :

```
<form>
 Saisissez votre clé de recherche :
 <input id="suggest" size="30" />
</form>
```

À noter que l'attribut `id` du champ de saisie doit être configuré avec le nom `suggest` de sorte à ce qu'il puisse être sélectionné par l'objet jQuery du code 15-12.

Les modifications du fichier `index.html` sont maintenant terminées, vous pouvez l'enregistrer et passer à la création de la feuille de styles dédiée au positionnement des éléments et à la mise en forme de la page de l'application. Nous vous proposons ci-dessous quelques styles très simples pour créer ce fichier en guise d'exemple.

Code 15-15 : styles de la page de l'application :

```
body {
  font-size: 16px;
  font-family: Verdana, Helvetica, Arial, sans-serif;
}
h1 {
  margin: 1em 0 1.5em;
  font-size: 18px;
}
p {
  margin: 0;
  font-size: 12px;
}
#page {
  position: relative;
  margin: 0 auto;
  width:600px;
  height:200px;
```

```
  border-top: medium solid #ff0000;
  border-bottom: medium solid #ff0000;
}
#formulaire {
  position: absolute;
  left: 38px;
  top: 103px;
  width: 532px;
}
```

Enregistrez ce fichier sous le nom `styleContenu.css` et assurez-vous que les styles sont bien appliqués à la page `index.html` en l'ouvrant dans un navigateur.

Côté serveur, nous allons utiliser le fichier proposé par l'auteur du plug-in (vous pouvez le télécharger sur son site ou le ressaisir dans votre éditeur en vous inspirant du code 15-16). Nous vous proposons cependant ci-dessous de détailler son fonctionnement afin que vous puissiez éventuellement le modifier pour l'adapter à votre application.

Ce fichier est très simple, il récupère le mot de recherche envoyé par le moteur Ajax (grâce à la variable HTTP `$_GET['q']`), lui applique une fonction `strtolower()` pour convertir tous ses caractères en minuscules puis effectue une recherche de ce résultat dans toutes les clés du tableau associatif `$items`. À noter qu'une structure de test `if(!$q)` est ajoutée avant le traitement afin d'éviter la génération d'un message d'erreur au cas ou le fichier `search.php` serait appelé sans paramètre d'URL.

Code 15-16 : programme du fichier serveur `search.php` :

```
$q = strtolower($_GET["q"]);
if (!$q) return;
//déclaration et initialisation du tableau des données
$items = array(
"Great Bittern"=>"Botaurus stellaris",
"…"=>"…",
"Heuglin's Gull"=>"Larus heuglini"
);
//recherche du mot dans les clés dans le tableau
foreach ($items as $key=>$value) {
  if (strpos(strtolower($key), $q) !== false) {
    echo "$key|$value\n";
  }
}
```

La structure du système de recherche s'appuie sur une boucle `foreach()` qui parcourt toutes les clés des données du tableau (`$key`). Dès que le mot recherché (`$q`) est détecté (à l'aide de la fonction `strpos()`) dans la clé en cours, la condition du test `if()` renvoie `true`, ce qui déclenche l'affichage de la clé et de la valeur correspondante en retour (ces deux valeurs sont séparées par le symbole « | », soit `$key|$value`).

Les données utilisées pour nos tests sont les mêmes que celles du site de démonstration de l'auteur mais vous pouvez évidemment les remplacer facilement par celles de votre application ou, mieux, coupler ce système avec des résultats issus d'une requête SQL.

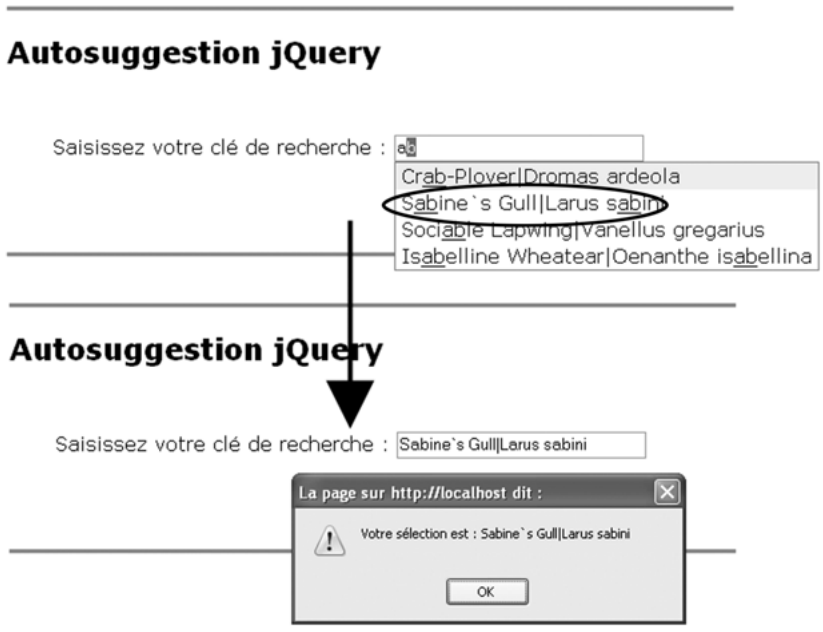
Enregistrez maintenant le fichier `search.php` et passez aux tests du système dans le navigateur.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Le navigateur doit afficher le champ de saisie vide. Saisissez une première lettre (« a » par exemple) puis une seconde (« b » par exemple). Une requête Ajax est alors envoyée au serveur et les données retournées dans sa réponse s'affichent dans une liste placée en dessous du champ de saisie (voir le haut de la figure 15-3). En vous inspirant des suggestions, continuez votre saisie pour affiner votre recherche. Dès que le nombre de suggestions vous permet de faire votre choix sans ambiguïté, cliquez sur l'une des options pour la sélectionner. Une boîte d'alerte doit alors s'afficher vous rappelant le choix effectué (voir le bas de la figure 15-3).

Figure 15-3

Test du plug-in jQuery.suggest : champ de saisie avec autosuggestion



Atelier 15-3 : plug-in jQuery.calendar : widget calendrier

Composition du système

Contrairement aux plug-ins des deux premiers ateliers, qui intégraient un moteur Ajax jQuery, celui-ci est autonome et permet simplement de renseigner le champ « date » d'un formulaire à partir d'un petit widget calendrier.

Cette structure est composée :

- d'une page HTML (`index.html`) qui contient l'application ;
- du fichier de la bibliothèque de base jQuery (`jquery.js`) ;
- du fichier de la bibliothèque du plug-in jQuery.calendar (`jquery-calendar.js`) ;
- du fichier de la feuille de styles du plug-in jQuery.calendar (`jquery-calendar.css`) ;
- du fichier d'une feuille de styles de personnalisation du contenu de la page de votre application qui est à définir selon la mise en forme désirée (`styleContenu.css`).

Fonctionnement du système

Ce plug-in peut avantageusement être intégré dans un formulaire de réservation ou tout autre formulaire pour lequel l'utilisateur devra être assisté dans le choix d'une date valide dans un format normalisé.

Dès que l'utilisateur place son pointeur à l'intérieur du champ, l'événement est détecté et le widget calendrier apparaît en dessous du champ. L'utilisateur peut alors passer d'un mois à l'autre, voire d'une année à l'autre, afin de sélectionner la date désirée dans un mini calendrier. Une fois la date sélectionnée, le calendrier disparaît et la date choisie est mémorisée dans un format normalisé (par exemple : 20/10/2007) dans le champ de saisie.

Conception du système

Ouvrez une nouvelle page HTML et sauvegardez-la sous le nom `index.html` dans un nouveau répertoire nommé `/chap15/atelier15-3/`.

Rendez-vous ensuite sur le site de l'auteur à adresse ci-dessous.

http://jquery.com/plugins/project/jquery-calendar/

Depuis cette page, cliquez sur le lien « View the jQuery Calendar Project Page ». Dans le nouvel écran, deux liens différents vous permettent de télécharger la bibliothèque (`jquery-calendar.js`) et la feuille de styles du plug-in (`jquery-calendar.css`) sur votre ordinateur. Dans le bas de l'écran, dans la rubrique Location packages, cliquez sur le lien French. Le code source d'un script de localisation en français doit alors apparaître à l'écran. Copiez ce code puis collez-le dans une balise `<script>` de la page `index.html`.

Code 15-17 : script à intégrer dans la page `index.html` :

```
<script type="text/javascript">
/* French initialisation for the jQuery calendar extension. */
/* Written by Keith Wood (kbwood@iprimus.com.au). */
$(document).ready(function(){
  popUpCal.regional['fr'] = {
  clearText: 'Effacer',
  closeText: 'Fermer',
  prevText: '&lt;Préc',
  nextText: 'Proch&gt;',
  currentText: 'En cours',
  dayNames: ['Di','Lu','Ma','Me','Je','Ve','Sa'],
  monthNames: ['Janvier','Février','Mars','Avril','Mai','Juin','Juillet','Août',
➥'Septembre','Octobre','Novembre','Décembre']};
  popUpCal.setDefaults(popUpCal.regional['fr']);
});
</script>
```

Comme vous pouvez le remarquer, ce code est déjà conditionné par un gestionnaire `.ready()` et contient les différentes traductions des liens texte et autres informations du widget qui vous permettent de personnaliser votre calendrier en français.

Placez-vous avant l'accolade de fermeture du gestionnaire et ajoutez maintenant l'instanciation d'un objet jQuery avec sélection du champ `date` auquel nous allons appliquer la méthode `calendar()` de la bibliothèque.

Code 15-18 : affectation de la méthode `calendar()` à la balise `date` :

```
<script type="text/javascript">
…
$('#date').calendar();
});
</script>
```

Copiez ensuite, dans le répertoire de votre atelier, les fichiers de la bibliothèque de base `jquery.js` et `styleContenu.css` que nous avons déjà utilisés dans les ateliers précédents. Puis, en haut de la page `index.html`, ajouter 2 liens `link` afin de pouvoir disposer des feuilles de styles CSS du plug-in (`jquery-calendar.css`) ainsi que de la feuille de styles personnalisée de la page (`styleContenu.css`).

Ajoutez ensuite une balise de `script` faisant référence à la bibliothèque de base `jquery.js`.

```
<script src="jquery.js" type="text/javascript"></script>
```

En dessous de cette balise de `script`, ajoutez-en une seconde afin de référencer le plug-in `jquery-calendar.js`.

```
<script src="jquery-calendar.js" type="text/javascript"></script>
```

Placez-vous maintenant dans la balise `<body>` et ajoutez une structure de page afin de contenir le champ de recherche du formulaire.

Code 15-19 : création d'une structure de page :

```
<div id="page">
  <h2>jQuery.Calendar</h2>
    <!--zone du formulaire-->
    <div id="formulaire">
…
    </div>
</div>
```

Placez-vous ensuite à l'intérieur de la balise `<div>` dont l'identifiant est `formulaire` et ajoutez les balises du formulaire et du champ de saisie.

Code 15-20 : ajout des balises du formulaire et de son champ de saisie :

```
<form>
 Saisissez votre clé de recherche :
 <input type="text" name="date" id="date" />
</form>
```

À noter que l'attribut `id` du champ de saisie devra être configuré avec le nom `date` de sorte à ce qu'il puisse être sélectionné par l'objet jQuery du code 15-18.

Les modifications du fichier `index.html` sont maintenant terminées, vous pouvez l'enregistrer et passer aux tests du système dans le navigateur.

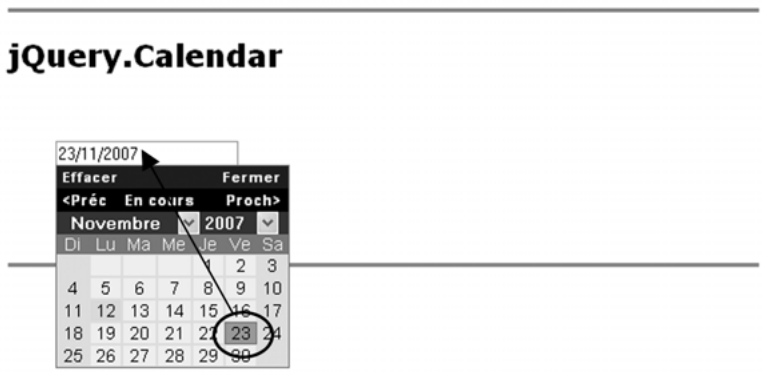
Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Le navigateur doit afficher le champ de saisie vide. Placez votre pointeur à l'intérieur du champ. Le calendrier doit alors apparaître en dessous du champ. Parcourez si besoin les différents mois ou années disponibles à l'aide des flèches Précédent et

Suivant ou, plus directement, à l'aide des menus déroulants. Arrêtez votre choix sur une date et cliquez dans la case correspondante dans le calendrier. Le calendrier doit alors disparaître et la date doit être mémorisée au format standard dans le champ de saisie.

Figure 15-4

Test du plug-in jQuery.calendar : widget calendrier



Partie IV

Ressources sur les technologies associées

Cette partie comporte plusieurs chapitres dédiés aux différentes technologies utilisées dans les applications Ajax-PHP. Vous pourrez ainsi vous y reporter ponctuellement pour trouver des compléments d'information lors de la réalisation des ateliers de la partie III de cet ouvrage, mais vous pouvez aussi les parcourir indépendamment du reste de l'ouvrage pour vous initier ou parfaire vos connaissances sur l'une de ces technologies.

16

XHTML

HTML permettait initialement de structurer et de mettre en forme une page Web. Avec le XHTML, sa présentation est maintenant gérée par les feuilles de style CSS, cette évolution permet désormais de scinder parfaitement la structure de la forme des pages Web afin qu'elles soient plus facilement traitées par des applications ou des périphériques différents sans pour autant nécessiter de créer autant de versions de la page que l'on nécessite de représentations.

Du HTML au XHTML

La tendance du HTML étant d'évoluer progressivement vers le XML, une évolution du HTML conforme aux contraintes du XML a ainsi été créée, il s'agit du XHTML. Même si le XHTML est considéré comme le successeur du HTML, il n'est pas issu du HTML mais du XML dont il hérite de ses spécificités.

Les versions du XHTML

Les premières spécifications du XHTML ont vus le jour en 2000 sous l'appellation XHTML 1.0. Un an plus tard, elle fut remplacée par la version 1.1. La version 2.0 est actuellement à l'étude par le W3C mais n'a pas encore été publiée.

Les contraintes du XHTML

En effet le HTML n'étant pas compatible avec le XML, il a donc fallu créer un nouveau langage dérivé du XML, mais conservant les principes fondamentaux du HTML. Le XHTML n'a donc pas la même tolérance que le HTML et il faudra veiller à respecter les contraintes de balisage énumérées ci-après :

- Toujours clôturer une balise ouvrante par une balise fermante (le nom de la balise fermante étant le même que celui de la balise ouvrante mais précédée du symbole "/").

```
<p>Bonjour tout le monde</p>
```

- Si une balise n'a pas de contenu, il n'y aura pas de balise fermante mais le symbole ">" de la balise ouvrante devra être précédé du symbole "/".

```
<br />
```

- Toujours mettre le nom des balises et des attributs en minuscules.

```
<a href="monFichier.php" >Cliquez-moi</a>
```

- Les attributs doivent toujours avoir une valeur et cette valeur doit toujours être encadrée par des guillemets simples ou doubles.

```
<option value="fr" selected="selected">France</option>
```

Composition d'un élément HTML

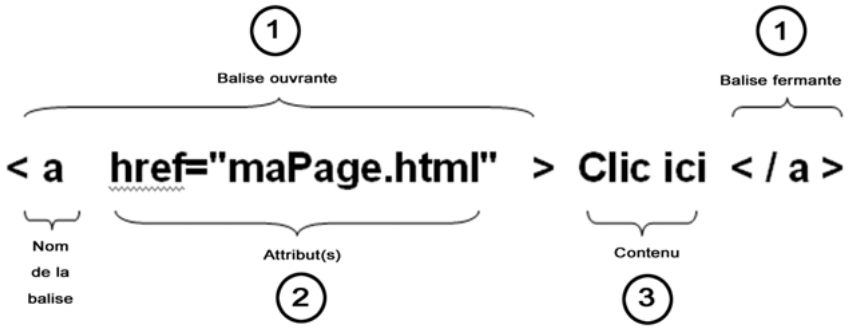
Il est important de bien identifier les composants d'un élément XHTML car nous les utiliserons à maintes reprises par la suite lors de l'utilisation des attributs et des méthodes du DOM.

Un élément HTML est délimité par la balise ouvrante et fermante (voir repère 1 de la figure 16-1). Il est composé de :

- son (ou ses) attribut(s) placé(s) dans la balise ouvrante (voir repère 2 de la figure 16-1).
- son contenu placé entre la balise ouvrante et la balise fermante (voir repère 3 de la figure 16-1).

Figure 16-1

Composition d'un élément.



Les balises neutres `<div>` et `<span>`

Le HTML met à notre disposition un grand nombre de balises et il serait trop long de toutes les détailler dans cet ouvrage. Cependant, dans les applications Ajax, nous aurons très fréquemment l'occasion d'interagir avec les balises de contenu usuelles du XHTML (`<h1>`, `<p>`, ...) et plus particulièrement avec les balises neutres `<div>` et `<span>`.

Les balises `<div>` et `<span>` ont été créées pour faciliter l'usage des feuilles de styles au sein des pages HTML afin de rendre plus souple l'application d'un style à une partie précise de la page. Contrairement aux autres balises HTML de contenu (`<h1>`, `<p>`, ...) qui impliquent une mise en forme par défaut, les balises `<div>` et `<span>` sont des balises génériques (pas de formatage par défaut) et seront donc souvent

☞

liées à un style par le biais d'une `class` ou d'un `id`. Le fait d'associer un identifiant `id` à ces balises permettra aussi de les manipuler avec du code JavaScript ce qui explique leur usage très fréquent dans les applications Ajax.

La balise `<div>` est de type `block` tout comme certaines balises traditionnelles du HTML (`<p>`, `<table>`, `<h1>`, `<li>`, ...) et peut contenir un simple texte mais aussi d'autres éléments conteneurs permettant ainsi de leur appliquer son style par héritage. Elle permet ainsi de diviser (d'ou le nom de la balise : `<div>`) une page HTML en différents blocs structurels. Il est possible de mettre tout type d'éléments HTML à l'intérieur d'une balise `<div>` : images, tableaux, paragraphes, autres balises `<div>`. À noter enfin, que l'utilisation d'une balise `<div>` entraîne un retour à la ligne obligatoire avant et après le bloc qu'elle délimite.

Dans l'exemple ci-dessous la balise `<div>` dont le `id` est `page` contient deux autres éléments `<h1>` et `<p>` qui hériteront de son style (voir code 16-2).

Code 16-1 :

```
<div id="page">
  <h1>Bonjour</h1>
  <p>Bravo, vous avez gagné 65 euros</p>
 </div>
```

La balise `<span>`, quant à elle, est de type `inline`, comme les balises `<a>`, `<input>`, `<i>`, `<b>`, etc du HTML traditionnel. Elle ne délimite pas un bloc structurel mais une zone à l'intérieur d'une balise de contenu et permet ainsi d'appliquer un style spécifique à une chaîne de caractères. Évidemment, aucun retour à la ligne ne sera généré avec une balise `<span>` puisqu'elle sera insérée directement dans la ligne d'un texte (type `inline`).

Dans l'exemple ci-dessous, la balise `<span>` délimite une zone dans laquelle sera placé le résultat du gain afin de lui appliquer un style particulier (grâce à l'`id resultat`) pour mettre sa valeur en évidence.

Code 16-2 :

```
Bravo, vous avez gagné <span id="resultat">0</span> euros
```

Structure d'un document XHTML

La déclaration XML

La première ligne du code permet d'indiquer que le document appartient à la famille XML 1.0 et qu'il doit être bien formé tout comme un document XML dont il est issu (la déclaration XML reste cependant facultative et peut même engendrer des erreurs sur certains navigateurs anciens).

Code 16-3 :

```
<?xml version="1.0"?>
```

Le DOCTYPE

Pour qu'un navigateur interprète correctement le code d'une page XHTML 1.0, il faut préciser le DOCTYPE qu'il doit utiliser pour sa validation. En effet, le fichier DTD du DOCTYPE auquel il se réfère contient toutes les contraintes auxquelles doivent se conformer les éléments et attributs de la page Web. Ainsi, l'utilisation d'une balise non décrite dans le fichier DTD est interdite.

Si le DOCTYPE est bien renseigné et les codes XHTML et feuilles de style CSS sont élaborés selon les normes de la spécification du DOCTYPE choisi, la page sera interprétée correctement par la majorité des navigateurs actuels. Dans le cas contraire, si le

DOCTYPE n'est pas renseigné, le navigateur tentera de se comporter comme un ancien navigateur et parcourra la page en mode Quirks (en considérant qu'il s'agit d'une ancienne page HTML 4) avec les risques de résultats imprévisibles que cela comporte.

Les deux spécifications DOCTYPE les plus employées actuellement sont les DTD strict et transitional.

Si vous utilisez la balise de déclaration d'un DOCTYPE strict, le navigateur effectuera une interprétation très rigoureuse des éléments HTML et de leurs règles d'imbrication, alors qu'avec le DOCTYPE transitional la validation sera plus permissive et acceptera plus facilement certaines entraves aux spécifications XHTML.

Par exemple, avec le DOCTYPE strict vous ne pouvez pas utiliser les attributs de présentation utilisés avec le HTML 4, devenus obsolètes avec le XHTML en raison de la séparation entre la structure et la forme d'une page (comme `align` ou `font` par exemple). Il faut donc obligatoirement utiliser des feuilles de styles CSS pour appliquer une mise en forme à une page Web. De même, tous les contenus textuels devront être insérés dans des éléments conteneurs (comme H1, DIV, …) rendant ainsi impossible l'écriture d'un texte directement dans la balise BODY.

Le DOCTYPE transitional, quant à lui, a été élaboré pour favoriser la transition du HTML vers le XHTML strict. Il est donc plus souple et permet par exemple d'utiliser certains attributs de mise en forme devenus obsolètes avec le XHTML strict.

Code 16-4 : balise de déclaration d'un DOCTYPE strict :

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

Code 16-5 : balise de déclaration d'un DOCTYPE transitional :

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

Utilisez un validateur pour vérifier votre page XHTML

Si votre page est accessible depuis Internet, vous pouvez alors utiliser le validateur du W3C pour vérifier que votre code est bien conforme à la spécification du DOCTYPE déclaré dans la page. Pour cela, il suffit d'utiliser l'adresse indiquée ci-dessous et de saisir l'URL de votre page. Si votre page est correctement codée, vous obtiendrez un message de félicitations, sinon le validateur vous indiquera les erreurs qu'elle comporte, ce qui vous permettra de remédier aux problèmes.

http://validator.w3.org

L'élément racine du document

Dans une page XHTML, la balise de l'élément racine `<html>` contient deux attributs. Le premier attribut `xmlns` (XML-Name-Space) précise l'espace de nommage XML qui prend toujours la même valeur : *http://www.w3.org/1999/xhtml* alors que le second `xml:lang` indique la langue utilisée dans la page et prend la valeur `fr` si votre document est en français.

Code 16-6 :

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
…
</html>
```

La balise meta Content-Type

La balise meta `Content-Type` indique que votre page contient du HTML ou un langage héritant de ses spécificités comme le XHTML (`text/html`) et précise l'encodage utilisé dans le document (soit l'encodage UTF-8 dans notre exemple : `charset=utf-8`).

Code 16-7 :

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

La balise de titre

Comme pour les pages HTML 4, le titre est toujours très important et doit être choisi judicieusement. En effet, le titre ne se contente pas d'apparaître dans la barre de titre (barre située en haut de la fenêtre des navigateurs), il est aussi utilisé par défaut lors de l'enregistrement de votre page dans les favoris et son contenu est surtout exploité par les moteurs de recherche pour le référencement de votre page.

Code 16-8 :

```
<title>Titre de la page</title>
```

Une page XHTML complète

La configuration minimale d'une page doit contenir les balises que nous venons de présenter ci-dessus et être structurée en deux parties à l'aide des balises `<head>` et `<body>` (voir code 16-9). La première balise `<head>` est la balise d'en-tête de la page et regroupe les informations complémentaires à la page (invisibles à l'écran) alors que la seconde balise `<body>` est la balise du corps de la page et contient tous les éléments visibles du document XHTML.

Code 16-9 :

```
<?xml version="1.0"?>
<!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">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Titre de la page</title>
</head>
<body>
Ici le contenu visible de la page
</body>
</html>
```

17

CSS

Les feuilles de styles en cascade, également appelées CSS (*Cascading Style Sheets*) ont été ajoutées au W3C dès 1996. Il a cependant fallu attendre 2001 et l'apparition de Internet Explorer 6 pour qu'elles soient largement utilisées sur la plupart des sites.

Parmi les freins à leur diffusion, citons l'incidence des éditeurs Wysiwyg (comme Dreamweaver ou FrontPage) pour lesquels il a été très difficile d'implémenter la séparation du contenu et de la forme dans les fonctionnalités de leur interface. Mais le principal frein est, sans aucun doute, le problème de compatibilité entre les navigateurs : en effet, ceux-ci ont longtemps interprété les styles à leur manière, sans se préoccuper des spécifications publiées.

Désormais, et grâce au XHTML qui sépare parfaitement le contenu de la forme, l'usage des styles est devenu obligatoire (du moins en mode XHTML strict) pour appliquer une présentation à une page Web.

Cependant, hormis le fait d'apporter une solution pour la mise en forme du contenu d'une page XHTML, les CSS présentent bien d'autres avantages. Elles permettent d'alléger le code source de votre page et en facilitent la lecture ; elles contribuent à rendre un site homogène et à en améliorer la maintenance en centralisant dans un même fichier les codes de mise en forme du site entier ; elles permettent de dépasser (et de loin) les limites de la mise en forme bridées par le HTML. De même, si vous vous préoccupez de l'accessibilité de votre site, l'usage des CSS sera la solution idéale pour autoriser la consultation de vos pages par un large éventail d'utilisateurs, enfin, les styles pouvant être changés dynamiquement par le DOM, l'apparence d'une page pourra ainsi être modifiée à la volée, ce qui se révélera très utile pour mettre en œuvre les fonctions de traitement d'une réponse d'une requête Ajax.

Les différentes versions des CSS

La première spécification du W3C sur les CSS date de 1996 et a été publiée sous le nom *Feuilles de styles en cascade niveau 1*, également appelée CSS1. Une nouvelle version nommée CSS2 lui succéda en 1998. Enfin, une autre publication CSS3 est à l'étude depuis 2001 mais dans le cadre de cet ouvrage, nous nous limiterons à la version CSS2 qui a l'avantage de pouvoir être interprétée actuellement par la plupart des navigateurs récents.

Syntaxe des CSS

Le sélecteur de style

Pour qu'un style soit appliqué à un élément particulier de la page HTML, il faut utiliser une référence à cet élément. Pour matérialiser cette référence, nous utiliserons différents types de sélecteurs que nous allons présenter dans cette partie.

Le sélecteur est suivi d'accolades dans lesquelles viendra se placer la déclaration du style que nous présenterons ensuite.

Syntaxe générale des sélecteurs :

```
selecteur {déclaration du style}
```

Le sélecteur de balise

Le sélecteur le plus simple se nomme *sélecteur de balise*. Il permet de redéfinir la mise en forme par défaut d'une balise standard du HTML. Ce sélecteur est très intéressant si l'on désire appliquer un ensemble de styles à une page HTML complète pour en modifier son apparence mais sans modifier les balises qui la structurent. Il suffira alors d'ajouter la définition des styles en interne ou dans une feuille de styles externe pour que la page soit modifiée automatiquement.

Par exemple, pour appliquer un style particulier à toutes les balises `h1` de la page, il suffit de saisir le nom de la balise `h1` (sans les symboles < et >) suivi des accolades contenant la déclaration du style à appliquer à ce type de balise (voir code 17-1).

Syntaxe :

```
nomBalise {déclaration du style}
```

Code 17-1 : exemple de sélection de toutes les balises `h1` :

```
h1 {déclaration du style}
```

Il est aussi possible d'affecter un même style à plusieurs types de balise. Dans ce cas, il suffit de saisir leurs noms séparés par des virgules. Dans l'exemple ci-dessous, le même style sera appliqué aux balises `h1` ainsi qu'aux balises `p` de la page.

Code 17-2 : exemple de sélection de plusieurs types de balises :

```
h1,p {déclaration du style}
```

Si vous désirez appliquer un style à un élément situé dans plusieurs balises imbriquées, il faudra alors indiquer les différentes balises dans leur ordre d'imbrication à la suite l'une de l'autre et sans virgule en guise de sélecteur. Dans l'exemple ci-dessous, le style sera appliqué au contenu situé dans une balise `p`, elle-même située dans une balise `div`.

Code 17-3 : exemple de sélection de balises imbriquées :

```
div p {déclaration du style}
```

La classe class

Avec le sélecteur de balise que nous avons présenté précédemment, l'application d'un style est limitée à une balise standard du HTML (ou à la composition de plusieurs balises). Il est cependant souvent nécessaire de devoir appliquer un style à une sélection de plusieurs balises standards bien définies ou encore à des contenus qui ne sont pas inclus dans une balise standard.

Les balises div et span

Pour faciliter l'application des styles à une page Web, des balises neutres (sans mise en forme prédéfinie) ont été ajoutées au HTML. Il existe deux types de balises neutres : la balise `<div>` qui est de type `block` et peut structurer d'autres éléments `block` dans une page, et la balise `<span>` qui est de type `inline` et permet de délimiter une chaîne de caractères dans un texte (pour plus de détail sur ces balises, reportez-vous si besoin au chapitre concernant le XHTML). Ces deux balises seront très souvent utilisées avec les sélecteurs `class` et `id` présentés dans cette partie.

Le sélecteur `class` pourra alors solutionner ces problèmes car il permet d'identifier nominativement la sélection des balises auxquelles nous désirons appliquer le style. De plus, si le contenu n'est pas inclus dans une balise standard, le sélecteur `id` peut être couplé à une balise neutre (`div` ou `span`, voir encadré) pour délimiter ainsi la zone à laquelle il faut appliquer un style particulier.

Il est important de bien comprendre qu'une même classe peut être attribuée à plusieurs balises (contrairement au sélecteur `id` que nous allons présenter ci-après) et que ces balises pourront être éventuellement de type différent (`div`, `p`) si les propriétés concernées par la règle de styles sont disponibles dans chacune de ces balises.

Syntaxe :

```
.nomClass {déclaration du style}
```

Exemple d'identification de la (ou les) balise(s) concernée(s) par la classe :

```
<div class="nomClass" > … </div>
```

L'identifiant id

Le sélecteur `id` peut aussi être utilisé pour appliquer des styles et fonctionne d'une manière similaire au sélecteur `class` hormis sa syntaxe qui est différente et surtout le fait qu'un même sélecteur `id` ne pourra identifier qu'un seul élément dans le document HTML (contrairement au sélecteur `class`) et il sera plutôt utilisé pour la mise en page (positionnement de bloc conteneur, par exemple) que pour la mise en forme de caractères.

Cela est une caractéristique très importante de ce type de sélecteur car elle nous permettra aussi d'utiliser ce même sélecteur `id` pour référencer individuellement des éléments afin de leur appliquer des traitements JavaScript.

Pour déclarer un style avec un identifiant, il faut faire précéder son nom d'un dièse (*#*). La déclaration du style sera encadrée par des accolades comme avec le sélecteur `class`.

À noter qu'il est possible de faire cohabiter dans la même balise une identification par une classe et une autre par un identifiant. On peut ainsi convenir de configurer les styles des éléments à l'aide des classes alors que les identifiants seront réservés à la programmation JavaScript pour manipuler ces mêmes éléments, par exemple.

Syntaxe :

```
#nomId {déclaration du style}
```

Exemple d'identification de la (ou les) balise(s) concernée(s) par un identifiant :

```
<div id="nomId" > … </div>
```

Les pseudo-classes

Dans certains cas, il est possible qu'un élément ne puisse pas être identifié par une balise (qu'elle soit standard comme `p` ou neutre comme `div`). Pour solutionner ce problème, vous pouvez recourir aux pseudo-classes. Il en existe deux types : les pseudo-classes de lien pour l'application d'un style particulier lorsqu'un lien a été visité, par exemple (voir tableau 17-1), et les pseudo-classes de paragraphe pour l'application d'un style particulier à la première lettre d'un paragraphe, par exemple (voir tableau 17-2).

Tableau 17-1 Pseudo-classes de lien

Pseudo-classe	Désignation
`:link`	Lien pas encore sélectionné
`:visited`	Lien déjà visité
`:hover`	Lien pendant le survol de la souris
`:active`	Lien au moment ou il est cliqué

Attention à l'ordre des pseudo-classes de lien

Pour éviter des effets inattendus, il est impératif que l'ordre des déclarations ci-dessous soit respecté : `:link`, puis `:visited`, puis `:hover` et enfin `:active`.

Tableau 17-2 Pseudo-classes de paragraphe

Pseudo-classe	Désignation
`:first-letter`	Première lettre d'un paragraphe
`:first-line`	Première ligne d'un paragraphe

Pour déclarer une pseudo-classe, il faut commencer par indiquer le sélecteur choisi. Cela peut être un simple sélecteur de balise (`a` par exemple), une classe ou un identifiant `id`. Vous devez ensuite faire suivre ce sélecteur de la pseudo-classe à considérer (voir les tableaux 17-1 ou 17-2). Enfin, comme pour tous les sélecteurs, la déclaration du style encadrée par des accolades clôture la ligne.

Syntaxe :

```
sélecteur : pseudo-classe {déclaration du style}
```

Pour illustrer une utilisation d'une pseudo-classe, nous vous proposons de l'appliquer pour définir le style d'une couleur aux différents états de tous les liens hypertextes d'une page (voir code 17-4).

Code 17-4 : voir fichier `exemplePseudoClasse.html` :

```
<head>
<style type="text/css">
<!--
a:link {color: #999999;}
a:visited {color: #FF00FF;}
a:hover {color: #FF0000;}
a:active {color: #FFFF00;}
-->
</style>
```

```
</head>
<body>
<a href="#">Cliquez ici</a>
</body>
```

La déclaration d'un style

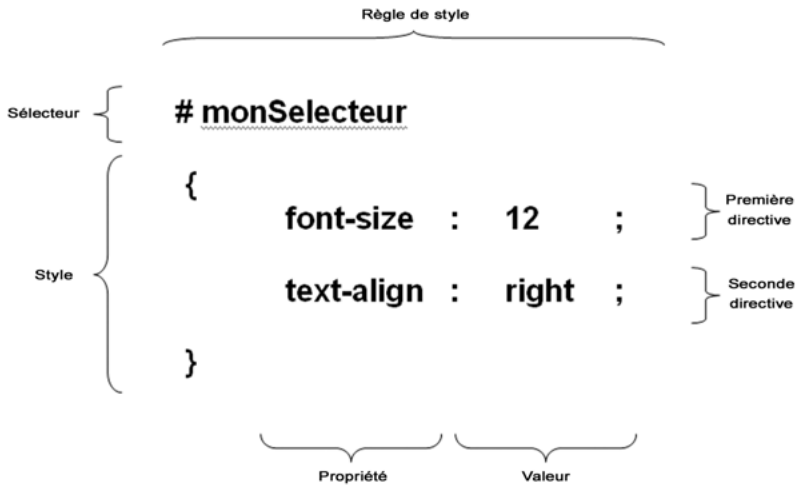
Nous avons vu comment identifier l'élément auquel sera appliqué le style avec les sélecteurs. Nous allons voir maintenant comment déclarer le style lui-même pour ensuite le placer entre les accolades qui suivent le sélecteur.

Terminologie des styles

On appelle communément un *style* l'ensemble des directives elles-mêmes composées d'une propriété et d'une valeur. Les styles sont ensuite liés à un sélecteur et l'ensemble constitue ce qu'on appelle une *règle de styles*. Pour que tous ces termes soient clairement identifiés, nous avons résumé la syntaxe d'une règle de styles en annotant les termes associés à ses différents composants dans la figure 17-1.

Figure 17-1

Terminologie des styles



La déclaration d'une directive de style est composée de deux parties. La première correspond à la propriété du style et la deuxième à la valeur qui lui est appliquée. Ces deux parties sont séparées par deux points (:) et terminées par un point virgule (;). L'ensemble constitue ce qu'on appelle une *directive* (et aussi le style dans le cas particulier où il n'y a qu'une seule directive).

Syntaxe d'une directive :

```
propriété :  valeur ;
```

Le style ainsi constitué sera ensuite placé entre les accolades, précédées du sélecteur. Cet ensemble constitue ce qu'on appelle une *règle de styles*.

Syntaxe d'une règle de styles constituée d'un sélecteur et d'un style d'une seule directive :

```
sélecteur {propriété :  valeur ;}
```

Il est possible de déclarer plusieurs directives de style pour un même sélecteur. Dans ce cas, les couples propriété/valeur de chaque directive sont placés à la suite les uns des autres en respectant la même syntaxe que pour la déclaration d'une directive isolée. Un groupe de directives constitue ce qu'on appelle le style proprement dit.

Syntaxe d'un style constitué de deux directives :

```
propriété_1 : valeur_1 ; propriété_2 : valeur_2;
```

Syntaxe d'une règle de styles constituée d'un sélecteur et d'un style de deux directives :

```
sélecteur {propriété_1 : valeur_1 ; propriété_2 : valeur_2;}
```

Les propriétés et les valeurs d'un style

Il existe une grande diversité de propriétés classées par familles selon l'élément auquel elles se rapportent mais elles sont bien trop nombreuses pour toutes les mentionner dans le cadre de cet ouvrage. Vous trouverez ci-dessous quelques exemples de ces propriétés mais nous vous invitons à consulter le site officiel du W3C, `http://www.w3.org/TR/REC-CSS2/`, pour en obtenir une liste exhaustive.

Tableau 17-3 Quelques exemples de propriétés et de leurs valeurs associées

Familles	Propriétés	Valeurs
Arrière-plan	background-color	<couleur> transparent
	background-image	<url> none
Police	font-size	<taille> small medium large …
	font-style	normal italic oblique
	font-weight	normal bold lighter …
Texte	text-align	left right center justify
	text-decoration	none underline overline …
	vertical-align	top middle bottom …
	color	<couleur>

Tableau 17-3 Quelques exemples de propriétés et de leurs valeurs associées *(suite)*

Familles	Propriétés	Valeurs
Boîte	`border-color`	`<couleur>`
	`margin`	`<dimension-absolue>` `<dimension-relative>`
	`padding`	`<dimension-absolue>` `<dimension-relative>`

Affichage	`position`	`static`
		`absolute`
		`relative`
		`fixed`

	`visibility`	`collapse`
		`visible`
		`hidden`

	`display`	`block`
		`inline`
		`none`
		…

…	…	…

Les unités d'une dimension

En ce qui concerne les valeurs correspondantes à la dimension d'un élément, différentes unités peuvent être utilisées avec les CSS : le pixel (px), le pica (pc), le point (pt), le centimètre (cm), le pouce (in) et le pourcentage (%) pour indiquer une dimension relative.

Pour les polices, il est possible d'indiquer leur taille d'une manière fixe (en point ou en pixel) ou relative (en em : 1 em correspondant à 100 % de la taille de la police en cours). Enfin, si vous désirez que votre valeur soit prise en compte par tous les navigateurs, ne mettez pas d'espace entre la valeur et l'unité (exemple : 2em).

Valeurs des couleurs

Avec les CSS, les couleurs peuvent être exprimées de différentes manières : avec la notation traditionnelle en hexadécimale au format #RRVVBB avec pour R, V et B une valeur hexadécimale fonction des 3 composantes d'une couleur Rouge, Vert et Bleu (par exemple #FF0000 pour du rouge), avec la notation hexadécimale abrégée (soit #F00 pour du rouge, une couleur n'étant représentée que par un seul symbole hexa), avec la notation RGB décimale (soit rgb(255,0,0) pour du rouge) ou encore en utilisant le nom d'une couleur normalisée en anglais (red, gree, yellow, blue…).

Application d'un style

Différentes manières de spécifier un style

Une fois les règles de styles définies, il faut ensuite les spécifier à un endroit pour qu'elles puissent agir sur les éléments de la page HTML. Il existe plusieurs endroits où spécifier un style et nous proposons de vous les présenter ci-dessous.

Style en ligne

Cette méthode de déclaration d'un style directement dans la balise de l'élément n'est pas très avantageuse en production car les styles ne sont pas séparés de la présentation HTML et la centralisation des styles n'est plus assurée, d'où un problème de maintenance. En revanche, elle peut être exploitée pour la mise au point de la maquette d'une page car chaque style ne concerne qu'un seul élément isolé et cela permet de faire des réglages sans perturber le reste de la page.

Vous trouverez ci-dessous un exemple d'utilisation (voir fichier `exempleInLine.html`) :

```
<h1 style="color:red;">Bonjour à tous</h1>
```

Style interne

Si vous désirez que les règles de styles soient appliquées uniquement aux éléments d'une page, il est possible de les spécifier dans une balise `<style>` imbriquée dans la balise `<head>` de la page concernée.

Dans l'exemple ci-dessous, nous spécifions le même style que précédemment mais il est appliqué de la même manière à tous les éléments `h1` de la page.

Code 17-5 : page HTML (voir fichier `exempleInterne.html`) :

```
<head>
<style type="text/css">
<!--
h1 {
color: red;
}
-->
</style>
</head>
<body>
<h1>Bonjour à tous</h1>
<h1>Bonsoir à tous</h1>
</body>
```

Vous remarquerez que la règle de styles est encadrée par les symboles des commentaires HTML (`<!--` et `-->`). Cette précaution permet d'éviter qu'un navigateur ne supportant pas les styles ne génère une erreur, même si cela est désormais très rare.

Feuille de styles externe

La façon la plus intéressante d'exploiter tous les avantages des règles de styles est de les spécifier dans une feuille de styles externe à la page HTML. Celle-ci pourra ensuite être appliquée facilement à toutes les pages d'un site, permettant ainsi d'obtenir un site homogène et d'en faciliter sa maintenance. À noter que cette méthode permet aussi d'améliorer le temps de chargement des pages car, une fois chargée, la feuille de styles

sera mémorisée dans le cache du navigateur permettant ainsi de ne pas la recharger à chaque appel d'une page du site.

Pour créer une feuille de styles, il suffit de saisir les règles des styles directement dans un fichier portant une extension `.css`. Une balise de lien `<link>` devra ensuite être ajoutée dans toutes les pages désirant exploiter les styles du fichier.

Dans l'exemple ci-dessous, le style précédent a été déplacé dans le fichier nommé `style.css` et un lien a été ajouté à la page HTML. Outre les avantages que nous venons de rappeler, le fonctionnement sera strictement identique à l'exemple précédent.

Contenu du fichier `style1.css` :

```
/* CSS Document */
h1 {
color: red;
}
```

Code 17-6 : page HTML (voir fichier `exempleExterne.html`) :

```
<html>
<head>
<link href="style1.css" rel="stylesheet" type="text/css" />
</head>
<body>
<h1>Bonjour à tous</h1>
</body>
</html>
```

Si vous observez le lien dans la page HTML, vous remarquerez que l'attribut `rel="stylesheet"` a été ajouté afin de préciser que le lien concerne une feuille de styles.

Style importé

Il existe aussi une autre alternative à l'utilisation du lien `<link>` pour lier une feuille de styles. En effet, il est aussi possible d'importer une feuille de styles dans la page HTML avec une propriété du CSS2 (`@import`) au lieu d'y faire référence par le biais d'un lien, simple balise HTML. L'avantage de cette méthode est de pouvoir importer une feuille de styles, non seulement dans une page HTML comme pour la méthode précédente, mais surtout dans une autre feuille de styles qui hériterait ainsi des styles de la seconde feuille.

Dans l'exemple ci-dessous, nous avons créé une seconde feuille de styles nommée `style2.css` qui contient la propriété `@import` qui importe la feuille de styles précédente (non modifiée). La page HTML, quant à elle, fait maintenant référence à cette nouvelle feuille `style2.css`. Nous avons également modifié le style de la seconde balise de titre en `h2` de sorte à pouvoir constater l'effet produit par les deux feuilles de styles dans la page HTML. Ainsi, si vous testez la page, le premier texte « Bonjour à tous » s'affichera en rouge (grâce à la feuille `style1.css`) et le second texte « Bonsoir à tous » s'affichera quant à lui en vert (grâce à la feuille `style2.css`).

Contenu du fichier `style1.css` :

```
/* CSS Document */
h1 {
color: red;
}
```

Contenu du fichier `style2.css` :

```
/* CSS Document */
@import url(style1.css);
h2 {
color: green;
}
```

Code 17-7 : page HTML (voir fichier `exempleImport.html`) :

```
<html>
<head>
<link href="style2.css" rel="stylesheet" type="text/css" />
</head>
<body>
<h1>Bonjour à tous</h1>
<h2>Bonsoir à tous</h2>
</body>
</html>
```

L'effet cascade d'un style

Dans la partie précédente, nous avons présenté plusieurs moyens de définir un style. Cependant, en cas de conflit, il faut faire appel aux ordres des priorités (voir tableau 17-4) pour arbitrer l'application du style à l'élément concerné.

En règle générale, le style le plus près de l'élément est celui qui a la plus forte priorité. Ainsi, le style en ligne (même s'il est peu utilisé) aura priorité sur le style interne de la page qui lui-même aura priorité sur le style de la feuille CSS externe.

Toutefois, les règles de priorités des styles ne sont pas toujours appliquées pour résoudre une erreur de définition. En effet, il est quelquefois intéressant de les utiliser pour modifier localement (pour une page, par exemple) le style général du site. Dans ce cas, il suffit de surcharger volontairement le style interne de la page afin qu'il écrase celui qui est défini dans la feuille de style externe.

Tableau 17-4 Règles de priorités des styles

Règle (de la plus forte à la plus faible)	Type de définition du style
1	Style en ligne
2	Style interne
3	Style externe

Forcer un style avec !important

À noter qu'il existe une solution pour modifier l'ordre établi de ces priorités. Il suffit pour cela d'ajouter la valeur `!important` après la valeur de la directive et avant le point virgule pour rendre prioritaire le style concerné sur les autres styles quelles que soit leurs priorités. Dans l'exemple ci-dessous (voir code 17-8), nous avons utilisé cet artifice pour forcer le style interne (`color: red`) sur le style en ligne (`color: green`).

Code 17-8 :

```
<head>
<style type="text/css">
```

☞

```
<!--
p {
color: red!important;
}
-->
</style>
</head>
<body>
<p style="color: green">Bonjour</p>
</body>
```

L'effet d'héritage d'un style

Lorsqu'un style est appliqué à un élément qui contient d'autres éléments enfants, ces derniers héritent alors du style de leurs parents.

Ainsi, dans l'exemple ci-dessous l'élément `<i>` hérite du style de son parent `<h1>` en plus de son propre style. Si vous testez cette page, vous remarquerez que tout le texte est de couleur rouge et que le mot « tous » est en plus souligné. La balise `<i>` hérite donc bien de la couleur de la balise `<h1>` en plus de son style de soulignement.

Code 17-9 : page HTML (voir fichier `exempleHeritage.html`) :

```
<head>
<style type="text/css">
<!--
h1 {
color: red;
}
i {
text-decoration: underline;
}
-->
</style>
</head>
<body>
<h1>Bonjour à <i>tous</i></h1>
</body>
```

18

XML

Définition du XML

XML est l'acronyme de eXtensible Markup Language. Comme le HTML, le XML est une norme SGML (Standard Generalized Markup Language) mais celle-ci a été développée bien plus tard (en 1998) alors que le HTML est défini par le consortium W3C depuis 1990.

Un document XML se caractérise par le fait qu'il contient uniquement des données structurées, sans aucune indication quant à leur présentation. Ainsi, si vous ouvrez un document XML dans un navigateur, il n'affiche que la structure des données sous forme d'arborescence. XML est donc particulièrement bien adapté pour structurer, enregistrer et transmettre des données.

Le XML est aussi un langage qui utilise des balises non prédéfinies pour structurer un document, contrairement au HTML pour lequel l'usage de balises standard est obligatoire (`<head>`, `<body>`, `<p>`…).

Avantages du XML

Les avantages du XML sont multiples. En voici quelques-uns qui devraient vous convaincre de son intérêt.

- **Simple –** Comme les documents de type HTML, le document XML est un simple document texte construit à partir de balises contenant des informations. Il est donc lisible et interprétable par tous sans outil spécifique et avec peu de connaissances préalables.
- **Souple –** L'utilisateur peut, s'il le désire, structurer les données et nommer librement chaque balise et attribut du document (contrairement au HTML pour lequel les noms des balises et des attributs sont prédéfinis).
- **Extensible –** Le nombre de balises n'est pas limité (comme c'est le cas pour le HTML) et peut donc être étendu à volonté.

- **Indépendant –** Grâce à son contenu basé sur un document texte et donc universel, il peut être utilisé sur tout type de plate-forme (PC, Mac, Unix…) mais également avec tout type de langage de programmation (PHP, ASP…).
- **Interopérabilité –** Le fait que le XML soit un langage universel favorise l'interopérabilité des applications et permet de réaliser rapidement et simplement des échanges de données.
- **Gratuit –** Le XML est développé par le consortium W3C. Son utilisation est donc libre et ne nécessite pas l'achat d'une licence commerciale.

Utilisations du XML

Pour le stockage de données

Grâce à sa structure, le document XML hiérarchise et formate les données. Il permet de stocker des données complexes et favorise ainsi leur exploitation au sein d'une application.

Pour le transfert de données

Lors du développement d'applications sur des serveurs hétérogènes, on est fréquemment confronté à des problèmes de transfert de données entre applications. Il est toujours possible de créer des programmes pour convertir ces données d'un format vers un autre, mais il est souvent plus judicieux d'exploiter un document XML pour assurer leur transfert (tout en préservant leur structure) entre ces deux applications.

Structure d'un document XML

Si vous avez déjà utilisé un document XHTML, vous ne serez pas dépaysé face à un document XML.

L'exemple ci-dessous permet de stocker d'une manière structurée l'âge des petits-enfants de plusieurs pères de famille (pour simplifier la représentation, seuls les descendants masculins ont été représentés).

Exemple de document XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE info SYSTEM "http://adressedusite.com/info.dtd">
 <ages>
  <pere prenom="Jean" nom="Dupond">
   <enfant prenom="Paul">
    <petitenfant prenom="Laurent">8</petitenfant>
    <petitenfant prenom="Julien">5</petitenfant>
   </enfant>
   <enfant prenom="Alain">
    <petitenfant prenom="Charles">12</petitenfant>
   </enfant>
  </pere>
  <pere prenom="Claude" nom="Durand">
   <enfant prenom="Fabrice">
    <petitenfant prenom="Alex">10</petitenfant>
    <petitenfant prenom="Maxime">7</petitenfant>
    <petitenfant prenom="Fabien">3</petitenfant>
```

```
  </enfant>
 </pere>
 <pere prenom="Thierry" nom="Duval">
  <enfant prenom="Nicolas">
   <petitenfant />
   <!--Nicolas n'a pas d'enfant-->
  </enfant>
 </pere>
</ages>
```

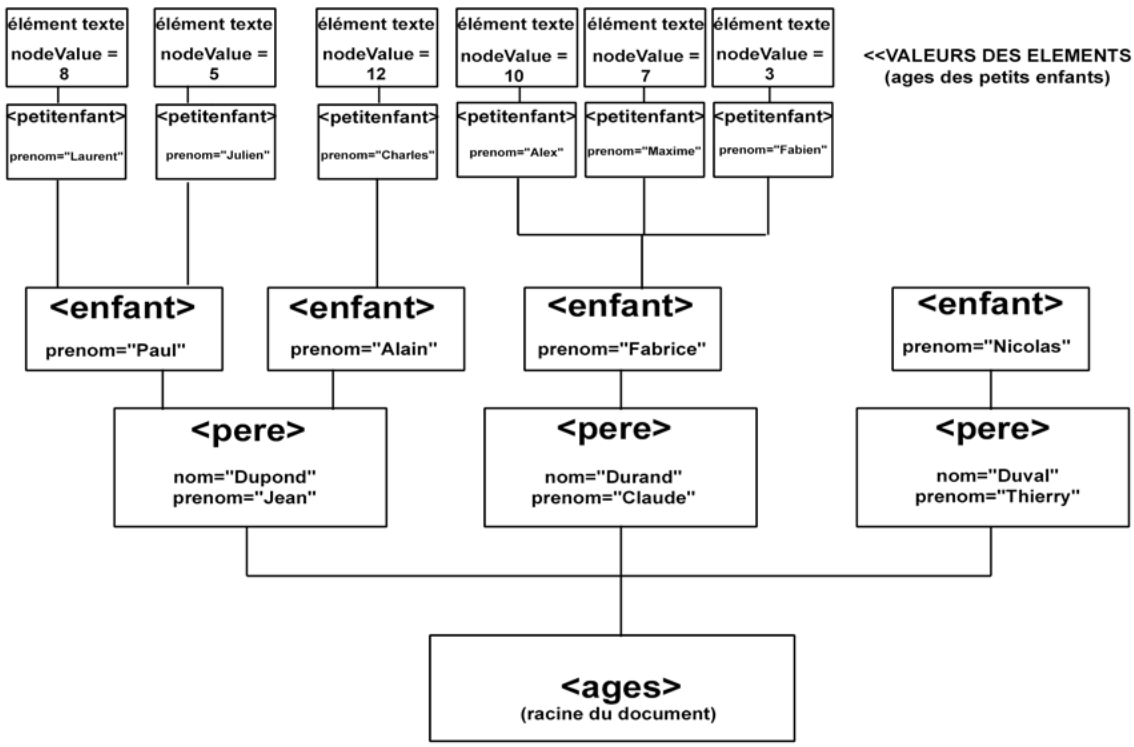


Figure 18-1

Structure d'un document XML

L'en-tête

Le document commence par un en-tête (facultatif) qui contient des informations sur la version de XML (`version="1.0"`), le jeu de caractères utilisé (`encoding="UTF-8"`) et l'autonomie du document (`standalone="no"`). Dans l'en-tête, seule la version est obligatoire. Si aucun type de codage n'est défini, l'UTF-8 est utilisé par défaut.

```
<?xml version="1.0" encoding="UTF-8"? standalone="no" >
```

L'en-tête peut aussi faire référence à une déclaration du type de document (la DTD : Document Type Definition) qui permet de valider la conformité du document en se référant à l'URL d'un document en ligne ou en local (exemple : http://adressedusite.com/info.dtd).

```
<!DOCTYPE info SYSTEM "http://adressedusite.com/info.dtd">
```

Si l'en-tête se réfère à une DTD externe (comme c'est le cas dans l'exemple ci-dessus), le document n'est pas autonome et l'attribut `standalone` doit être configuré avec la valeur `"no"`. Dans le cas contraire (s'il n'y a pas de DTD ou si elle est interne), le document est autonome et la valeur de l'attribut `standalone` doit être définie à `"yes"`. En cas d'absence de l'attribut `standalone`, la valeur par défaut est `"no"`.

Le document XML qui suit l'en-tête utilise des blocs de construction semblables à ceux des documents HTML pour structurer son contenu (éléments, attributs, valeurs et commentaires).

L'élément

Un élément (appelé aussi nœud) est l'entité de base du document XML. Il peut contenir d'autres éléments ou tout type de contenu (chaîne de caractères, etc.). Le contenu d'un élément est encadré par une balise ouvrante (par exemple `<pere>`) et une balise fermante : une balise fermante contient le même nom que la balise ouvrante précédé d'un slash (par exemple `</pere>`).

Si l'élément ne possède pas de contenu, les balises ouvrante et fermante sont remplacées par une seule balise comportant un slash après le nom de l'élément (par exemple `<petit-enfant />`).

Le nom indiqué dans ces deux balises doit décrire le contenu de l'élément, mais il n'est pas prédéfini comme en HTML (`<body>`, `<table>`, `<form>`, etc.). Bien que le nom de l'élément est libre, il doit être composé uniquement des lettres de l'alphabet, des chiffres ou des caractères « - » et « _ ». Il ne doit jamais contenir d'espace ou commencer par un chiffre.

L'attribut

Il est possible d'ajouter des attributs à la balise ouvrante d'un élément (par exemple `<enfant nom="Paul">`). Le nom des attributs contenus dans une balise est couplé avec une valeur encadrée par des guillemets (par exemple `nom="Paul"`). Un attribut doit toujours avoir une valeur. Le nombre d'attributs par élément n'est pas limité, à condition que chaque nom d'attribut soit différent ; l'exemple ci-après est donc incorrect : `<pere nom="Durand" nom="Dupond">`. Si un élément comporte plusieurs attributs, ils doivent être séparés par des espaces (par exemple `<pere prenom="Jean" nom="Dupond">`).

Les valeurs

Dans un document XML, les valeurs peuvent correspondre à des valeurs d'attribut (par exemple `nom="Paul"`) ou à des valeurs d'élément (par exemple `<petitenfant nom="Alex">10</petitenfant>`).

> **Important**
> La valeur d'un élément doit être considérée comme un élément texte enfant à part entière de cet élément dans la hiérarchie du document XML (voir figure 18-1).

Les commentaires

Comme pour le HTML, des commentaires peuvent être ajoutés dans un document XML. La syntaxe est d'ailleurs identique à celle utilisée pour intégrer des commentaires dans une page HTML (par exemple `<!--Ceci est un commentaire XML-->`). À l'intérieur d'un commentaire, vous pouvez utiliser tout type de symbole sauf les doubles tirets « -- ». Les commentaires servent à annoter les documents XML afin de se rappeler de l'utilité de certains blocs d'éléments ou pour détailler la structure du document. Ils peuvent également servir à déboguer en neutralisant une partie du document afin qu'il ne soit pas interprété par l'analyseur XML.

Règles d'écriture d'un document XML bien formé

Même si les documents XML sont simples et extensibles, ils ne sont pas pour autant dépourvus de règles. On appelle « document bien formé » un document qui respecte les règles suivantes.

- **Un seul élément racine –** Chaque document XML doit posséder un seul élément racine. L'élément racine contient tous les autres éléments du document. Cet élément particulier s'appelle « nœud racine » ou « root ».

Exemple :

```
<ages><pere>35</pere><pere>43</pere><ages>
```

Ici, la balise `ages` est le nœud racine du document XML.

- **Balises de fermeture obligatoires –** Comme nous l'avons vu précédemment, chaque élément doit être encadré par des balises ouvrante et fermante. Contrairement au HTML (dans lequel la balise `<p>` n'est pas obligatoirement fermée, de même que `<hr>`, qui est une balise inhérente sans balise de fermeture), le XML ne supporte pas l'absence de balises fermantes. Il faudra donc veiller à toujours ajouter une balise de fermeture à tous les éléments d'un document XML. Si le document possède un élément vide, utilisez une balise unique avec un slash avant le signe `>` final (par exemple `<enfant/>`).

- **Respecter l'imbrication des éléments –** Lorsque vous ouvrez un premier élément puis un second, insérez la balise de fermeture du second avant celle du premier. Ainsi le code suivant est incorrect : `<a><b>contenu</a></b>`, alors que celui-ci est correct : `<a><b>contenu</b></a>`.

- **Respecter la casse –** Le XML est sensible à la casse. Ainsi, les noms d'éléments « `pere` », « `Pere` » et « `PERE` » sont considérés comme différents. D'autre part, les noms des éléments et des attributs doivent être saisis en minuscules.

- **Mettre les valeurs des attributs entre guillemets –** Si une balise contient un couple nom d'attribut et sa valeur, la valeur doit toujours figurer entre guillemets (simples ou doubles), par exemple : `<enfant nom="paul">`. Un attribut doit toujours avoir une valeur.

- **Utiliser les entités prédéfinies pour les caractères réservés –** Comme en HTML, il existe des caractères réservés dont l'usage est interdit (`<`, `>`, `&`, `'` et `"`). Pour chacun de ces caractères, utilisez l'entité prédéfinie correspondante (`&lt;`, `&gt;`, `&amp;`, `&quot;`, `&apos;`).

- **Utiliser une section CDATA pour échapper un bloc de texte complet –** Afin d'éviter d'utiliser des entités pour des longs blocs de texte comportant des caractères réservés, vous pouvez ajouter une section CDATA en respectant la syntaxe suivante : `<![CDATA[bloc de texte]]>`.

Vérification d'un document XML

Pour savoir si un document est bien formé, une méthode simple consiste à l'ouvrir dans un navigateur Internet récent possédant un interpréteur XML intégré, comme les navigateurs ultérieurs à IE 6 ou à Firefox 2 (voir figure 18-2). Le navigateur présentera alors le document XML sous la forme d'un arbre. Chaque nœud de cet arbre pourra être développé en cliquant sur le + qui le précède : dès qu'un niveau est développé, un - vient remplacer le + afin de permettre à l'utilisateur de le replier.

Figure 18-2

Affichage d'un document XML bien formé dans un navigateur Internet

DTD et document XML valide

Nous venons d'aborder les règles d'écriture que doit respecter un document XML bien formé. Ces règles confirment que le document est conforme sur le plan syntaxique. Cependant, il existe un autre niveau d'exigence pour un document XML : celui d'être valide.

Pour qu'un document XML bien formé soit valide, il faut qu'il soit conforme à la déclaration du type de document (DTD) qui est spécifiée dans l'en-tête du fichier XML (revoir la présentation de l'en-tête ci-dessus). Cette DTD décrit toutes les contraintes auxquelles le contenu du document XML doit se conformer. Elle précise par exemple les types d'éléments autorisés ainsi que leurs attributs et valeurs possibles.

La DTD peut être spécifiée en interne, directement intégrée au document, ou en externe en précisant l'adresse où l'on peut la trouver. Une DTD peut être privée (`SYSTEM`) ou publique (`PUBLIC`). Dans le cas d'une DTD privée (réservée à un groupe spécifique d'utilisateurs), une référence à un fichier portant l'extension `.dtd` doit figurer dans l'en-tête du document comme dans l'exemple ci-dessous.

```
<!DOCTYPE info SYSTEM "http://adressedusite.com/info.dtd">
```

L'avantage de disposer d'un document valide est de pouvoir accéder à des fonctionnalités avancées comme dans les applications d'échange de données dans lesquelles la définition du type de document de part et d'autre est primordiale pour que les transferts puissent être réalisés, mais soyez rassurés, pour les applications Ajax de cet ouvrage il ne sera pas nécessaire de créer de DTD.

19

JavaScript

JavaScript est un langage de script qui permet de créer de l'interactivité dans les pages Web. Les scripts JavaScript sont intégrés au code HTML de la page Web et sont exécutés par le navigateur sans devoir faire appel aux ressources du serveur. Ces instructions sont donc traitées en direct par le navigateur (contrairement aux langages serveur comme le PHP).

Javascript est différent de Java

Bien que les deux soient utilisés pour créer des pages Web évoluées et qu'ils reprennent le terme Java (café en américain), le premier est intégré dans la page HTML et interprété par le navigateur (fichier source du code), alors que le second est accessible par un module à part (applet) sous forme compilée (fichier binaire).

Avertissement

L'objectif de ce chapitre et de vous donner les bases du langage JavaScript afin que vous puissiez comprendre les instructions utilisées dans les différents ateliers. Il n'a donc pas la prétention d'être un cours exhaustif sur ce langage car il faudrait y consacrer un livre entier pour cela.

Si vous désirez compléter votre apprentissage du JavaScript, nous vous suggérons de visiter le site ci-dessous.

http://www.w3schools.com/js/

Vous pouvez aussi consulter les ouvrages suivants :

- Thierry Templier et Arnaud Gougeon, *Javascript pour le Web 2.0,* Éditions Eyrolles ;
- Shelley Powers, *Débuter en JavaScript*, Éditions Eyrolles.

La syntaxe de JavaScript

Attention à la casse de vos codes

Le JavaScript est sensible à la casse, il convient donc de bien vérifier que la capitalisation de vos instructions est correcte. Par exemple, la fonction `alert()` s'écrit en minuscule et non pas `Alert()` ou `ALERT()`, de même si vous déclarez une variable `monResultat`, il est impératif de respecter la même casse à chaque fois que vous l'utilisez dans le programme.

Emplacements des codes JavaScript

Il est possible de placer des fonctions ou du code JavaScript directement dans la page HTML. Dans ce cas, les instructions sont insérées dans l'en-tête ou dans le corps de la page et encadrées par une balise de script comme dans le code 19-1 ci-dessous.

Code 19-1 : fonction JavaScript intégrée dans une page HTML :

```
<script language="javascript" type="text/javascript">
    <!--
    function jouer() {
          alert('vous avez gagné');
    }
    -->
</script>
```

Les gestionnaires d'événements peuvent, eux aussi, être intégrés directement dans une balise HTML comme dans le code 19-2 ci-dessous.

Code 19-2 : gestionnaire d'événement inséré dans une balise HTML :

```
<body onload= "alert('bonjour à tous');" >
<form name="form">
  <input type="button" id="go" name="go" onclick="jouer();" value="jouer" />
</form>
```

Cependant, de même qu'il faut désormais intégrer dans une feuille de styles CSS externe tous les paramètres de mise en forme d'une page XHTML afin de séparer la structure de la présentation d'une page Web, nous vous recommandons pour vos futurs développements JavaScript de regrouper tous les codes, fonctions et gestionnaires d'événements dans un fichier JavaScript externe afin de bien séparer la structure HTML de la programmation.

Ainsi, si nous reprenons les deux exemples précédents de codes intégrés dans la page, nous pourrions obtenir le même fonctionnement en les regroupant dans un fichier externe comme l'illustre le code 19-3 ci-dessous.

Code 19-3 : fichier JavaScript externe (exemple : `fonctionsMachine.js`) :

```
window.onload= chargement;
function chargement() {
  document.getElementById("go").onclick=jouer;
  alert('bonjour à tous');
}
function jouer() {
  alert('vous avez gagné');
}
```

Ce fichier externe doit néanmoins être référencé par une balise de référence placée dans l'en-tête de la page HTML (voir code 19-4).

Code 19-4 : Balise de référence au fichier externe :

```
<script type="text/javascript" src="fonctionsMachine.js"></script>
```

Les commentaires

Ne confondez pas les commentaires

Même si les commentaires d'un programme JavaScript peuvent se retrouver dans une page HTML, il ne faut pas confondre les commentaires HTML (voir ci-dessous) avec ceux réservés au JavaScript.

```
<!-- ceci est un commentaire HTML -->
```

Commentaires de simple ligne //

Si on désire insérer un commentaire sur une ligne ou à la fin d'une instruction, il faut écrire deux barres obliques `//` devant celui-ci (comme en PHP) :

Code 19-5 : commentaires simple ligne :

```
alert("Bonjour"); //Ici c'est un commentaire en bout de ligne
// Ici c'est un commentaire sur une ligne complète
```

Commentaires multilignes /* et */

Si on désire insérer plusieurs lignes de commentaire, il faut utiliser le signe `/*` au début de la zone de commentaire et `*/` à la fin de celle-ci (comme en PHP) :

```
/*
ceci est un commentaire
sur plusieurs
lignes
     */
```

Utiliser les commentaires pour déboguer

Dans le cadre du dépannage d'un script JavaScript, vous pouvez utiliser les commentaires pour neutraliser une ligne ou un bloc de code. Cela permet de tester la page sans interpréter la partie neutralisée et d'identifier quel script produit l'erreur.

La hiérarchie JavaScript

Tous les éléments d'une page Web sont contenus dans le document HTML (`document`) qui est lui même contenu dans la fenêtre du navigateur (`window`). Théoriquement, il faudrait donc rappeler cette descendance à chaque fois que vous désirez cibler un document précis de la page mais, en pratique, il est possible de commencer le chemin hiérarchique d'un élément à partir de `document` car il est implicitement l'enfant de `window`.

Pour construire un chemin hiérarchique ciblant un élément précis de la page, il suffit d'ajouter en syntaxe pointée après `document` la hiérarchie des différents noms (valeur de l'attribut `name` de chaque élément) de ses éléments parents.

Par exemple, pour cibler le bouton du formulaire du code 19-2, nous pouvons écrire son chemin hiérarchique comme indiqué dans le code 19-6, car le bouton `go` est enfant du

formulaire `form`, lui même enfant de `document` (`go`, `form` et `document` étant les valeurs de l'attribut `name` de chaque balise).

Code 19-6 : ciblage de l'élément `go` (bouton) du formulaire du code 19-2 à l'aide de son chemin hiérarchique :

```
window.document.form.go
```

ou encore plus simplement :

```
document.form.go
```

Selon la structure de la page, ce chemin peut quelquefois être compliqué mais, heureusement, il existe une autre façon de cibler les éléments d'un formulaire. Pour cela, il suffit d'utiliser les méthodes JavaScript du DOM (revoir si besoin le chapitre consacré à ce sujet). Par exemple, si l'identifiant de l'élément est connu, nous pouvons cibler l'élément `go` de notre exemple précédent en utilisant la méthode `getElementById()` comme l'illustre l'exemple ci-dessous (attention, ici `go` est la valeur de l'attribut `id` et non de l'attribut `name`).

Code 19-7 : ciblage de l'élément `go` (bouton) du formulaire du code 19-2 à l'aide d'une méthode du DOM :

```
document.getElementById("go")
```

Dans le cadre de cet ouvrage, nous allons utiliser principalement cette seconde méthode pour référencer les éléments d'une page HTML.

L'importance de l'attribut id

Comme l'attribut `class`, l'attribut `id` permet d'attribuer un style à l'élément concerné à la différence près qu'un seul élément ne peut être identifié qu'avec un attribut `id` contrairement à l'attribut `class` qui permet d'identifier une série d'éléments ayant la même classe.

D'autre part, l'attribut `id` permet surtout au JavaScript de manipuler les attributs et le contenu de l'élément ainsi identifié. Nous utiliserons fréquemment cette particularité avec les différentes méthodes DOM dans cet ouvrage.

Les gestionnaires d'événements

La plupart des programmes sont déclenchés par des événements. Pour configurer un événement, il faut utiliser un gestionnaire d'événement (voir la sélection de quelques gestionnaires du tableau 19-1) afin de définir la fonction qui est appelée lorsque l'événement se produit. Pour cela, le gestionnaire d'événement devra être appliqué à une référence de l'élément concerné (revoir si besoin la partie précédente sur le ciblage d'un élément). Il suffit ensuite d'affecter à cet ensemble le nom de la fonction qui prend en charge l'événement (voir code 19-8 ci-dessous).

Code 19-8 : exemple de configuration d'un gestionnaire d'événement :

```
document.getElementById("go").onclick=jouer;
}
function jouer() {
  alert('vous avez gagné');
}
```

Cependant, la déclaration d'un gestionnaire ne peut être effectuée que lorsque l'élément concerné est préalablement chargé dans la page. Pour se prémunir de cela, il suffit alors

d'insérer la déclaration du gestionnaire d'événement dans un autre gestionnaire (`window.onload`) qui teste si le document contenant l'élément est bien chargé.

Code 19-9 : exemple de configuration d'un gestionnaire d'événement avec un test préalable du chargement de la page :

```
window.onload= chargement;
function chargement() {
  document.getElementById("go").onclick=jouer;
  alert('bonjour à tous');
}
function jouer() {
  alert('vous avez gagné');
)
```

Tableau 19-1 Liste d'une sélection de quelques gestionnaires d'événements JavaScript

Gestionnaire d'événement	action détectée
`onblur`	Perte de focus sur un élement HTML
`onclick`	Clic de la souris sur un élement HTML
`onfocus`	Prise de focus sur un élement HTML
`onkeydown`	Pression d'une touche du clavier
`onkeyup`	Relâchement d'une touche du clavier
`onkeypress`	Encodage d'une touche du clavier
`onload`	Chargement de la page par le navigateur
`onmousedown`	Pression du bouton de la souris
`onmousemove`	Déplacement de la souris
`onmouseover`	Survol d'un élément par la souris
`onreset`	Action du bouton Reset d'un formulaire
`onselect`	Sélection d'un texte dans un élément HTML
`onsubmit`	Action du bouton Submit d'un formulaire

Les variables

La variable et sa valeur

Les variables sont des symboles auxquels on affecte des valeurs. Après leur affectation, vous pouvez modifier les variables à tout moment au cours du déroulement du programme. La déclaration d'une variable n'est pas obligatoire en JavaScript car elle peut être créée lors de sa première affectation. De même, les variables prennent le type correspondant à la valeur qui leur est affectée.

Noms des variables

Les noms des variables JavaScript doivent respecter les contraintes suivantes :

- Toujours commencer par une lettre ou un caractère de soulignement « _ ».

- Hormis la première lettre, le nom d'une variable peut comporter un nombre indéterminé de lettres ou chiffres ainsi que les caractères « _ » ou « $ ».
- Les espaces ne sont pas autorisés.
- Ne pas utiliser les mots réservés (mots utilisés dans le code JavaScript comme par exemple `return`, `var`, `if`…).
- Le JavaScript étant sensible à la casse, il convient d'écrire les noms des variables avec la même casse que celle qui a été utilisée lors de sa déclaration (ainsi `monResultat` est différent de `monresultat`).

En pratique, il est judicieux de choisir un nom explicite et de se fixer une règle de nommage.

Types des variables

Les variables JavaScript peuvent être de plusieurs types.

Tableau 19-2 Les variables JavaScript peuvent être de différents types selon leur affectation

Type de variable	Description	Exemples

Chaîne de caractères (`string`)	Leurs valeurs sont des lettres, chiffres ou symboles. Pour affecter une valeur alphanumérique à une variable, vous devez l'encadrer par des guillemets ou des apostrophes.
	Si la chaîne comporte une apostrophe, il convient de l'échapper avec le symbole « \ » ou de l'encadrer avec des guillemets.
	Une chaîne peut être vide, dans ce cas elle doit s'écrire avec deux guillemets sans espace entre les deux (comme par exemple `""`)
		`var1="Dupond";`
		`var3="254";`
		`var5= "d'en face" ;`
		`var5= 'd\'en face';`

Numériques (`number`)	Leurs valeurs sont des nombres entiers ou décimaux. Pour affecter un entier à une variable, il ne doit pas être encadré par des guillemets. S'il s'agit d'une valeur décimale, c'est le point et non la virgule qui devra être utilisée.	`var1=152;//nombre entier`
		`var2=5.22;//nombre décimal`

Booléens (`boolean`)	Leurs valeurs sont soit `true` (vrai), soit `false` (faux). Ces valeurs sont affectées à la variable en utilisant une expression de condition (exemple : `var1==var2`).	`var3=(var1==2);` `/* si var1 est égale à 5, var3 est une variable booléenne et sa valeur est false dans ce cas */`

Comment connaître le type de donnée ?

L'opérateur `typeof`, retourne le type de donnée. Ainsi, si la donnée est de type numérique, chaîne de caractères, fonction ou objet, cet opérateur retourne respectivement `number`, `string`, `function` ou `object`.

Il est alors facile de mettre en place un test pour vérifier le type d'une donnée (`info` par exemple) avec une structure semblable à celle-ci :

```
if(typeof info =="number") { alert("c'est un nombre"); }
```

Variables simples

Une variable simple est une donnée qui ne peut contenir qu'une seule valeur. Il existe deux façons de déclarer une variable :

Explicitement en faisant précéder son nom du mot-clé `var` :

```
var monResultat;
```

Implicitement lors d'une première affectation :

```
monResultat=200;
```

Une variable peut être déclarée explicitement puis initialisée en lui affectant une valeur (qui détermine son type). Cependant, il est aussi possible de regrouper ce deux actions en une seule instruction comme l'illustrent les exemples ci-dessous.

Code 19-10 :

```
//déclaration puis initialisation d'une variable simple
var monResultat;
monResultat=200;

//déclaration et initialisation simultanée d'une variable simple
var monResultat=200;
```

Attention aux déclarations de variables dans une fonction

Si vous déclarez une variable dans le bloc d'une fonction, il est très important de bien faire attention au type de déclaration que vous utilisez. En effet, si vous la déclarez explicitement (ce que nous vous engageons à faire en général), la variable est locale (elle ne peut être utilisée que dans le corps de la fonction concernée) alors que si vous la déclarez implicitement, elle est globale et est accessible partout dans le script.

Les variables globales sont parfois nécessaires mais peuvent devenir dangereuses si elles rentrent en conflit avec d'autres variables de même nom dans une autre fonction. Nous vous conseillons donc de toujours déclarer vos variables avec le mot-clé `var` sauf si cela est réellement indispensable pour le fonctionnement de votre système.

Les tableaux (Array)

Les tableaux sont des séries de valeurs regroupées sous le même identifiant. On distingue deux types de tableaux : les tableaux indicés (chaque élément de la série est identifié par un index numérique entre crochets comme par exemple : `tab1[0]`) et les tableaux associatifs (chaque élément de la série est identifié par une clé texte entre crochets comme par exemple : `tab2['jean']`).

Tableaux indicés

Pour déclarer un tableau indicé, il faut créer un objet `Array` avec l'opérateur `new` (attention à bien respecter la casse et à mettre un « A » majuscule à `Array` lors de la création du tableau).

Exemple de déclaration d'un tableau indicé :

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

Une fois le tableau déclaré, nous pouvons ensuite l'initialiser avec des valeurs. Pour cela, il suffit d'utiliser la syntaxe avec crochets (`tab1[i]`) en précisant l'index correspondant (attention, l'index commence à 0).

Code 19-11 : exemple d'initialisation d'un tableau indicé contenant des prénoms :

```
prenoms[0] = "Jean";
prenoms[1] = "Alain";
prenoms[2] = "Marie";
```

Il est aussi possible d'initialiser le tableau lors de sa création, comme l'illustre le code ci-dessous :

```
var prenoms = new Array("Jean","Alain","Marie");
```

On peut ensuite accéder aux valeurs d'un tableau en utilisant la même syntaxe à crochets, comme dans l'exemple ci-dessous :

```
document.write('Le prénom est :'+prenoms[1]);
```

Si vous désirez afficher tout le contenu d'un tableau, vous pouvez alors utiliser une boucle en exploitant la propriété `length` qui indique le nombre d'éléments dans un tableau.

Code 19-12 : affichage du contenu d'un tableau avec `for` :

```
for(i=0; i<prenoms.length; i++) {
  document.write('Le prénom est :'+prenoms[i]+'<br />');
}
```

Une autre solution consiste à utiliser une structure `for-in` comme l'illustre le code ci-dessous.

Code 19-13 : affichage du contenu d'un tableau avec `for-in` :

```
for(i in prenoms) {
  document.write('Le prénom est :'+prenoms[i]+'<br />');
}
```

Tableaux associatifs

Il est également possible de remplacer les index par un nom explicite (la clé). Dans ce cas, le tableau est dit associatif (exemple `tabAssoc["cle"]`).

Pour déclarer un tableau associatif, il faut commencer par créer un tableau des clés avec la même syntaxe que pour un tableau indicé.

Code 19-14 : déclaration des clés d'un tableau associatif :

```
var agences = new Array("centre","nord","sud");
```

Puis, il faut initialiser chaque valeur du tableau en indiquant la clé entre les crochets.

Code 19-15 : initialisation des valeurs d'un tableau associatif :

```
agences["centre"] = "Paris";
agences["nord"] = "Lille";
agences["sud"] = "Marseille";
```

Les tableaux associatifs permettent ensuite d'accéder directement à la donnée d'un tableau en précisant simplement sa clé.

Code 19-16 : utilisation des valeurs d'un tableau associatif :

```
document.write('La ville est :'+agences["nord"]);//affiche Lille
```

Si, toutefois, vous désirez connaître la clé d'une entrée du tableau, il suffit simplement d'indiquer son index numérique comme pour les tableaux indicés.

Code 19-17 : utilisation des clés d'un tableau associatif :

```
document.write('Le secteur est :'+agences[1]);//affiche nord
```

Tableaux multidimensionnels

Pour créer un tableau à plusieurs dimensions, il suffit de déclarer une autre structure de tableau à la place des différentes variables du tableau principal. Le tableau principal se comporte alors comme un tableau à deux dimensions (voir le tableau 19-3 et l'exemple ci-dessous).

Tableau 19-3 Matrice correspondante à l'exemple ci-dessous

[x][y]	[y]=[0]	[y]=[1]
[x]=[0]	A1	A2
[x]=[1]	B1	B2

Code 19-18 : initialisation et utilisation d'un tableau à deux dimensions :

```
//initialisation d'un tableau à 2 dimensions
var ligneA= new Array("A1","A2");
var ligneB= new Array("B1","B2");
var tableauprincipal=new Array (ligneA,ligneB);
//utilisation de ses éléments
document.write(tableauprincipal[0][0]); //affiche A1
document.write(tableauprincipal[0][1]); //affiche A2
document.write(tableauprincipal[1][0]); //affiche B1
document.write(tableauprincipal[1][1]); //affiche B2
```

Les objets

Les objets sont des données qui peuvent être regroupées en une seule entité contenant de nombreuses propriétés. Pour déclarer un objet, il faut créer un objet `Object` avec l'opérateur `new`.

Exemple de déclaration d'un objet :

```
var voiture = new Object();
```

Une fois l'objet déclaré, nous pouvons ensuite lui affecter des propriétés. Pour cela, il suffit d'utiliser la syntaxe pointée en précisant le nom de la propriété à créer.

Code 19-19 : exemple d'ajout de propriété à un objet :

```
voiture.couleur="rouge";
voiture.puissance="200ch";
```

JavaScript propose néanmoins une autre alternative (plus concise) à cette technique pour déclarer et initialiser un objet. C'est d'ailleurs cette syntaxe qui est utilisée pour les objets JSON. Le code équivalent aux trois lignes précédentes serait alors le suivant :

```
var voiture = { couleur : "rouge", puissance : "200ch"}
```

Pour utiliser ensuite les propriétés d'un objet, il suffit de rappeler le nom de l'objet suivi de celui de la propriété concernée en syntaxe pointée.

Code 19-20 : exemple d'utilisation des propriétés d'un objet :

```
document.write(' la couleur de la voiture est ' + voiture.couleur + '<br>');
document.write(' la puissance de la voiture est ' + voiture.puissance + '<br>');
```

L'exécution du code 19-20 affichera les informations ci-dessous à l'écran.

```
la couleur de la voiture est rouge
la puissance de la voiture est 200ch
```

Instructions et point-virgule

On appelle « instruction » une expression (pouvant être constituée de variables, constantes ou d'opérateurs) qui est terminée par un point-virgule. Cependant, en JavaScript, le point-virgule n'est pas obligatoire si vous retournez à la ligne entre chaque expression (nous vous conseillons quand même de toujours mettre un point-virgule à la fin de toutes vos instructions, surtout si vous débutez en JavaScript).

Code 19-21 : exemples d'instructions :

```
resultat=100;
alert(resultat);
// ces deux instruction affectent la valeur 100 à la variable resultat puis
// l'affichent dans une boîte d'alerte, le point-virgule au bout de chaque ligne est
// facultatif dans ce cas.
resultat=100; alert(resultat);
// ces deux instructions étant sur la même ligne, le point-virgule est dans ce cas
// obligatoire.
```

Les opérateurs

Les opérateurs permettent de lier des variables ou des expressions. Il existe différentes familles d'opérateurs selon les fonctions réalisées ou les expressions avec lesquelles on peut les employer (affectation, arithmétique, comparaison, logique ou de chaîne).

Opérateurs d'affectation

L'opérateur d'affectation est le plus courant. On peut aussi l'utiliser sous une forme compacte intégrant une opération arithmétique puis une affectation.

Tableau 19-4 Opérateurs d'affectation

Symbole	Définition
=	Affectation de base
+=	Addition puis affectation
-=	Soustraction puis affectation
*=	Multiplication puis affectation
/=	Division puis affectation

L'opérateur d'affectation de base permet d'attribuer une valeur issue d'une expression. Les autres opérateurs d'affectation permettent en outre de réaliser des opérations arithmétiques couplées avec l'affectation proprement dite.

Code 19-22 : exemples d'opérateurs d'affectation :

```
var1=0; // affectation de base (initialisation de $var1 à 0)
var1+=2; // ici var1 vaut 2 (0+2)
var1+=14; // et maintenant var1 vaut 16 (2+14)
```

Opérateurs arithmétiques

Lorsqu'on gère des variables de type numérique, on dispose d'opérateurs arithmétiques qui peuvent réaliser toutes les opérations mathématiques standard.

Enfin, l'incrémentation et la décrémentation (addition ou soustraction d'une unité) sont souvent utilisées en programmation et JavaScript ainsi que PHP fournissent des opérateurs spécifiques pour cela (++ et --).

Tableau 19-5 Opérateurs arithmétiques

Symbole	Définition
+	Addition
-	Soustraction
/	Division
*	Multiplication
%	Modulo : l'expression a % b retourne le reste de la division de a par b
++	Incrément (a++ ou ++a)
--	Décrément (a-- ou --a)

Code 19-23 : exemples d'opérateurs arithmétiques :

```
var1=5+2; // addition de deux valeurs numériques
var2=2+var1; // addition d'une valeur numérique et d'une variable
++var3; // après cette incrémentation, la variable est égale à "var3+1"
```

Opérateurs de comparaison

Les opérateurs de comparaison sont utilisés dans les expressions de condition des structures de programme. Ils permettent de comparer deux expressions. L'expression qui résulte de cette comparaison est égale à `true` lorsque la condition à contrôler est vérifiée et à `false` dans le cas contraire.

Tableau 19-6 Opérateurs de comparaison

Symbole	Définition
==	Égal
<	Strictement inférieur
>	Strictement supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal
!=	Différent

L'opérateur de comparaison permet d'élaborer des expressions de condition que vous pouvez utiliser dans les instructions de structure du programme (`if`, `while`, `for`…).

Code 19-24 : exemple d'utilisation d'une expression de condition :

```
if(var1>2) // teste l'expression de condition
  {document.write("le test est positif"); }
```

Opérateurs logiques

Les opérateurs logiques permettent de composer des expressions de condition complexes à partir de variables booléennes ou d'autres expressions de condition. Vous pouvez utiliser les parenthèses pour forcer les priorités entre les opérateurs ou pour améliorer la lisibilité du code en encadrant les expressions de condition.

Tableau 19-7 Opérateurs logiques

Symbole	Exemple	Fonction	Définition
`&&`	`var1 && var2`	ET	Renvoie `true` (vrai) si les deux variables var1 ET var2 sont `true`.
`||`	`var1 || var2`	OU	Renvoie `true` (vrai) si au moins l'une des deux variables var1 OU var2 est `true`.
`!`	`!var1`	Négation	Renvoie la négation de var1.

L'opérateur logique permet de relier logiquement deux expressions booléennes.

Code 19-25 : Exemple d'utilisation d'opérateurs logiques :

```
if((var1>2) && (var1<5))
  {document.write("la variable var1 est plus grande que 2 mais inférieure à 5");}
```

Opérateur de concaténation

L'opérateur de concaténation est souvent utilisé pour former des expressions à partir d'éléments de différents types (variable avec du texte par exemple) ou à partir d'autres expressions. L'opérateur utilisé pour relier ces expressions est le signe + (contrairement au PHP pour lequel c'est le point).

Tableau 19-8 Opérateur de concaténation

Syntaxe
`expression3 = expression1 + expression2 ;`

Code 19-26 : exemple d'utilisation d'un opérateur de concaténation :

```
// si var1 est égale à 50, alors var3 est égale à "50 euros"
var3=var1+"euros" ;
```

Les fonctions

Une fonction permet d'exploiter une même partie de code à plusieurs reprises dans un programme, ce qui est très intéressant pour les routines standards souvent utilisées en programmation.

Déclaration d'une fonction

La déclaration d'une fonction comporte une tête et un corps. Le mot-clé `function` est placé dans la tête de la fonction, suivi du nom de celle-ci et, entre parenthèses, de la liste des arguments attendus séparés par une virgule (lorsque la fonction ne comporte pas d'argument, les parenthèses sont vides). La tête de la fonction est suivie du corps encadré par des accolades. Les fonctions ont généralement une valeur de retour, désignée par le mot-clé `return`, suivi du résultat retourné dans le programme.

Tableau 19-9 Déclaration d'une fonction

Syntaxe de la déclaration d'une fonction

```
function nom_de_fonction (arg1,arg2…)
{
  instruction1;
  instruction2;
  …
  [return res;]
}
```

Appel d'une fonction

Une fois que la fonction est déclarée, il faut l'appeler dans le programme de sorte à ce que le script puisse s'exécuter. L'appel d'une fonction se fait par le nom de la fonction avec ses parenthèses dans lesquelles il est possible de passer des paramètres si besoin. Si un résultat est renvoyé par le mot-clé `return`, il se substitue alors à l'appel de la fonction. En général, dans ce cas l'appel de la fonction est alors affecté à une variable afin de récupérer le résultat ainsi retourné.

Tableau 19-10 Utilisation d'une fonction

Syntaxe de l'utilisation d'une fonction

```
nom_de_fonction (arg1,arg2…) ;
```

Code 19-27 : exemple de déclaration puis d'appel d'une fonction nommée `moyenne()` pour le calcul de la moyenne de deux valeurs :

```
//Déclaration de la fonction -------------------------------------
function moyenne (a,b) //tête de la déclaration
{ //début du corps
  var res=(a+b)/2; //instructions de la fonction
  return res; //information retournée au programme
} //fin du corps
//Utilisation de la fonction dans le programme ------------------
moncalcul=moyenne(4,6); //appel de la fonction
document.write("la moyenne de 4 et de 6 est égale à "+moncalcul);
```

Variables locales ou globales

Variable globale déclarée dans une fonction

Selon le type de déclaration de la variable dans le corps de la fonction (explicite avec `var` ou implicite sans `var`) elle est locale ou globale. Une variable locale ne peut être utilisée que dans le corps de la fonction, alors qu'une variable globale peut être utilisée dans tous les scripts (on appelle cela la portée d'une variable).

Code 19-28 : exemple d'utilisation d'une variable locale :

```
//Déclaration de la fonction -------------------------------------
function moyenne (a,b)
{
  var res=(a+b)/2; //déclaration de la variable locale "res"
  return res;
}
//Utilisation de la fonction dans le programme -----------------
var resultat=moyenne(4,6); //appel de la fonction
//utilisation du résultat renvoyé par la fonction.
document.write("la moyenne de 4 et de 6 est égale à "+ resultat);
```

Code 19-29 : exemple de déclaration d'une variable globale dans une fonction :

```
//Déclaration de la fonction -------------------------------------
function moyenne (a,b)
{
  res=(a+b)/2; //déclaration de la variable globale "res"
}
//Utilisation de la fonction dans le programme -----------------
moyenne(4,6); //appel de la fonction
//utilisation de la variable globale res.
document.write("la moyenne de 4 et de 6 est égale à "+res);
```

Variable globale déclarée en début de programme

Les variables déclarées en dehors des fonctions (déclarées en général au début du programme) sont globales, qu'elles soient déclarées d'une manière explicite ou implicite (avec ou sans le mot-clé `var`).

Code 19-30 : exemple de déclaration d'une variable globale en début de programme :

```
//Déclaration de la fonction -------------------------------------
var unite = "euros";
function moyenne (a,b)
{
  var res=(a+b)/2; //déclaration de la variable locale "res"
  res += unite;
  return res;
}
//Utilisation de la fonction dans le programme -----------------
var resultat=moyenne(4,6); //appel de la fonction
//utilisation du résultat renvoyé par la fonction.
document.write(resultat);//affiche 5 euros
```

Structures de programme

Structures de choix

Les structures de choix sont utilisées pour traiter les alternatives logiques au cours de l'exécution du script, afin d'orienter le déroulement du programme en fonction du résultat de l'alternative. Elles comprennent en général une expression de condition. Les expressions de condition sont constituées de variables ou de constantes reliées par des opérateurs logiques.

Structures de choix avec if

Si l'expression de condition entre parenthèses est vraie, l'instruction qui suit est exécutée, sinon il ne se passe rien et le programme continue de se dérouler après le bloc du `if`.

Tableau 19-11 Instruction conditionnelle `if`

Syntaxe

`if (expression_de_condition)`
`{`
`  instruction1;`
`  instruction2;`
`  …`
`}`

Forme simplifiée (s'il n'y a qu'une seule instruction à traiter) : `if (expression_de_condition) instruction1;`

Code 19-31 : exemples de structure `if` :

```
if (var1>4)
  document.write("la valeur est supérieure à 4");
/* ci-dessus un exemple de structure "if" avec une seule instruction */
//-------------------------------------
if (var1>4)
{//début du bloc if
  document.write(la valeur est supérieure à 4");
  document.write("<br> elle est exactement égale à $var1");
}//fin du bloc if
//ci-dessus un exemple de structure "if" avec un bloc d'instructions
```

Structures de choix avec if et else

La structure de choix utilisant l'instruction `if` ne traite que les structures de programme où la condition est vraie ; dans le cas contraire, aucune instruction n'est exécutée. Avec l'instruction `else`, vous pouvez définir les instructions à exécuter dans le cas où la condition testée serait fausse. Ces instructions sont regroupées dans un autre bloc qui suit l'instruction `else`.

Tableau 19-12 Instructions conditionnelles if et else

Syntaxe

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

Code 19-32 : exemple de structure `if-else` :

```
if (var1>4)
{
  document.write("la valeur est supérieure à 4");
  document.write("<br> elle est exactement égale à "+var1);
}
else
{//début du bloc else
  document.write("la valeur est inférieure ou égale à 4");
  document.write("<br> elle est exactement égale à "+var1);
  }//fin du bloc else
//ci-dessus un exemple de structure "if" avec "else"
```

Structures de boucle

Lorsqu'un ensemble d'instructions doit être exécuté plusieurs fois en fonction d'une condition, il faut alors utiliser une structure de boucle.

Structures de boucle avec while

La structure la plus simple est réalisée à l'aide de l'instruction `while`. Le bloc d'instructions est exécuté et répété tant que l'expression de condition retourne `true`. Lorsque la condition est ou devient fausse (`false`), le programme sort de la boucle pour exécuter les instructions qui se trouvent après la fin du bloc. Dans cette structure, il est fréquent d'utiliser une variable dédiée pour le compteur de boucle (exemple : `i`). Vous devez initialiser cette variable avant la boucle. Elle est ensuite testée dans l'expression de condition, puis incrémentée (ou décrémentée selon les cas) dans le corps de boucle. Pour sortir de la boucle, il faut faire évoluer le compteur de boucle, on utilise pour cela un opérateur d'incrémentation ou de décrémentation (`i++` ou `i--`). Choisissez le bon type d'opérateur, en fonction de la valeur de l'initialisation du compteur et de l'expression de condition choisie, sinon vous risquez d'obtenir une boucle infinie.

Tableau 19-13 Instruction de boucle `while`

Syntaxe

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

Code 19-33 : exemple de boucle `while` :

```
i=5; // initialisation du compteur de boucle à 5
while(i>0)
{
  document.write("Encore "+i+" tour(s) à faire <br>");
  i--; // décrémentation du compteur de boucle
}
document.write("Voilà, c'est enfin terminé");
/* ci-dessus un exemple qui affiche cinq fois le même texte (tant que i est
➥supérieur à 0) avant d'afficher le texte final. */
```

Structures de boucle avec for

L'instruction `for` est une seconde solution plus compacte pour traiter les boucles. Sa syntaxe est cependant radicalement différente de celle de la structure précédente, car les parenthèses de l'instruction contiennent trois expressions différentes, séparées par des points-virgules. Cette syntaxe très compacte est particulièrement appréciable quant à la lisibilité du code.

Tableau 19-14 Instruction de boucle `for`

Syntaxe	

`for (expression1;expression2;expression3)`
`{`
`  instruction1;`
`  instruction2;`
`  …`
`}`	

Légende	`expression1` : expression évaluée en début de boucle. Cette expression est fréquemment utilisée pour initialiser le compteur de boucle à l'aide de l'opérateur d'affectation (exemple : `$i = 5`). `expression2` : expression évaluée au début de chaque passage de boucle. Si le résultat de l'évaluation est `true`, le bloc d'instructions de la boucle est de nouveau exécuté. Dans le cas contraire, le programme sort de la boucle pour exécuter les instructions qui suivent le bloc. Cette expression est fréquemment utilisée pour tester le compteur de boucle à l'aide d'un opérateur de comparaison (exemple : `$i > 0`). `expression3` : expression évaluée à la fin de chaque boucle. Cette expression est fréquemment utilisée pour incrémenter ou décrémenter le compteur de boucle à l'aide d'un opérateur d'auto-incrémentation ou de décrémentation (exemple : `i--`).

Code 19-34 : exemple de bloc `for` :

```
for (i=5;i>0;i--)
{
  document.write("Encore "+i+" tour(s) à faire <br>");
}
document.write("Voilà, c'est enfin terminé");
//ci-dessus un exemple qui réalise la même boucle que celle donnée en exemple pour
➥l'instruction "while".
```

Structures de boucle avec for-in

La boucle `for-in` est dédiée à la manipulation des objets. Elle permet en effet de lire rapidement les différentes propriétés d'un objet sans avoir à écrire beaucoup de code.

Tableau 19-15 Instruction de boucle `for-in`

Syntaxe

`for(`*`propriete`* `in` *`objet`*`)`
`{`
`  instruction utilisant` *`propriete`*`;`
`}`

Voici un exemple utilisant l'instruction `for-in` pour afficher les propriétés de l'objet voiture.

Code 19-35 : exemple d'utilisation de la boucle `for-in` :

```
//création et configuration de l'objet voiture
var voiture = new Object();
voiture.couleur="rouge";
voiture.puissance="200ch";
//affichage des propriétés de l'objet avec for-in
for(var propriete in voiture){
 document.write(propriete + ' = ' + voiture[propriete] + '<br>');
}
```

Le code 19-35 affiche les informations ci-dessous à l'écran :

```
couleur = rouge
puissance = 200ch
```

Structures d'exception try-catch

Lors de son exécution, un programme peut générer des erreurs (appelées aussi des exceptions). JavaScript permet désormais de capturer ces exceptions susceptibles de survenir et de les contrôler en utilisant une structure `try-catch`.

Ainsi, si le bloc de code de l'instruction `try` ne génère aucune exception, le bloc `catch` est ignoré et le programme continue après la structure `try-catch`. Dans le cas contraire, si une exception survient, le programme est de suite dérouté vers le bloc `catch` qui est exécuté à son tour. Il est ainsi possible d'imbriquer de multiples structures `try-catch` afin de tester successivement plusieurs instructions comme dans le programme que nous utilisons pour définir quelle est la bonne instruction pour instancier un objet `XMLHttpRequest` selon le navigateur utilisé (voir code 19-36).

Tableau 19-16 Instruction d'exception `try-catch`

Syntaxe

`try`
`{`
`  //code à tester`
`} catch (Erreur) {`
`  //traitement à appliquer en cas d'exception sur le code tester`
`}`

Code 19-36 : exemple d'utilisation de l'instruction `try-catch` :

```
try {//test pour les navigateurs : Mozilla, Opéra...
  resultat= new XMLHttpRequest();
}
catch (Error) {
  try {//test pour les navigateurs Internet Explorer > 5.0
    resultat= new ActiveXObject("Msxml2.XMLHTTP");
  }
  catch (Error) {
    try {//test pour le navigateur Internet Explorer 5.0
      resultat= new ActiveXObject("Microsoft.XMLHTTP");
    }
    catch (Error) {
     resultat= null;
    }
  }
}
```

20

Gestion du DOM avec JavaScript

Ce chapitre est dédié à la programmation du DOM (*Document Object Model*). Vous y trouverez les descriptions de nombreux attributs et méthodes qui vous permettront de manipuler des éléments HTML, XML, des styles CSS et des événements à l'aide de JavaScript.

Les spécifications du DOM

Le DOM permet de modifier dynamiquement une page XHTML sans avoir à solliciter le serveur avec des requêtes HTTP traditionnelles. Ces techniques de programmation sont fréquemment employées dans les applications Ajax pour actualiser les éléments d'une page XHTML sans nécessiter son rechargement. Mais le DOM ne sert pas exclusivement à manipuler les éléments d'un document XHTML, il peut aussi interagir sur tout type de document texte structuré, comme le XML ou encore sur les styles ou les événements d'un document XHTML.

Dans la version 2 du DOM, la norme W3C est d'ailleurs divisée en plusieurs parties. L'une d'entre elles, le DOM Core, concerne plus particulièrement les documents XML et autres dérivés (dont le XHTML) alors qu'une autre, le DOM HTML, est dédiée exclusivement aux documents HTML. Une troisième partie, le DOM Style, concerne la gestion des styles des éléments. Enfin, nous verrons par la suite qu'une quatrième partie, le DOM Events, spécifie les différents événements utilisables dans une page Web et la manière de les gérer avec JavaScript.

L'arbre DOM

Avec le DOM, les éléments d'un document sont organisés hiérarchiquement, on parle alors d'arbre DOM du document. La version 2 du DOM définit tout un ensemble de méthodes qui permettent de naviguer dans cet arbre et de modifier les attributs et les contenus des éléments qui le constitue.

L'arbre DOM, une représentation du document en mémoire

Lors du chargement d'un document texte structuré (comme le XHTML ou le XML), le navigateur analyse les différents types de texte qui le constituent (balises, attribut, valeur …) et leur assigne à chacun un « nœud » puis les organise hiérarchiquement afin de former l'arbre DOM. L'arbre ainsi constitué est ensuite mémorisé dans la mémoire de l'ordinateur et correspond à la représentation réelle de ce que nous voyons à l'écran du navigateur.

Le DOM propose aussi une série de méthodes qui permet de modifier tous ces nœuds, voire d'en ajouter ou d'en supprimer. Ainsi, selon les méthodes utilisées par l'application, la représentation du document en mémoire (l'arbre DOM) pourra diverger de celle du code source initial. Il est donc important de bien prendre conscience que la représentation de l'arbre DOM peut être différente de celle du code source du même document en fonction des méthodes du DOM qui auront été utilisées par l'application après son chargement initial (voir figure 20-1).

Figure 20-1

À gauche : code source de la page Web. À droite : représentation de l'arbre DOM. Vous remarquerez que, malgré leurs similitudes, les deux représentations ne sont pas complément identiques (le contenu de la balise H1 ayant été modifié par une méthode DOM après le chargement de la page).

Terminologie d'un arbre DOM

- **Un arbre** est la structure hiérarchique qui relie les nœuds entre eux. Sa « racine » correspond au document (document) et son « tronc » à l'élément HTML (html) dans le cas d'une page Web. Les nœuds du document constituent le réseau de « branches » de l'arbre.

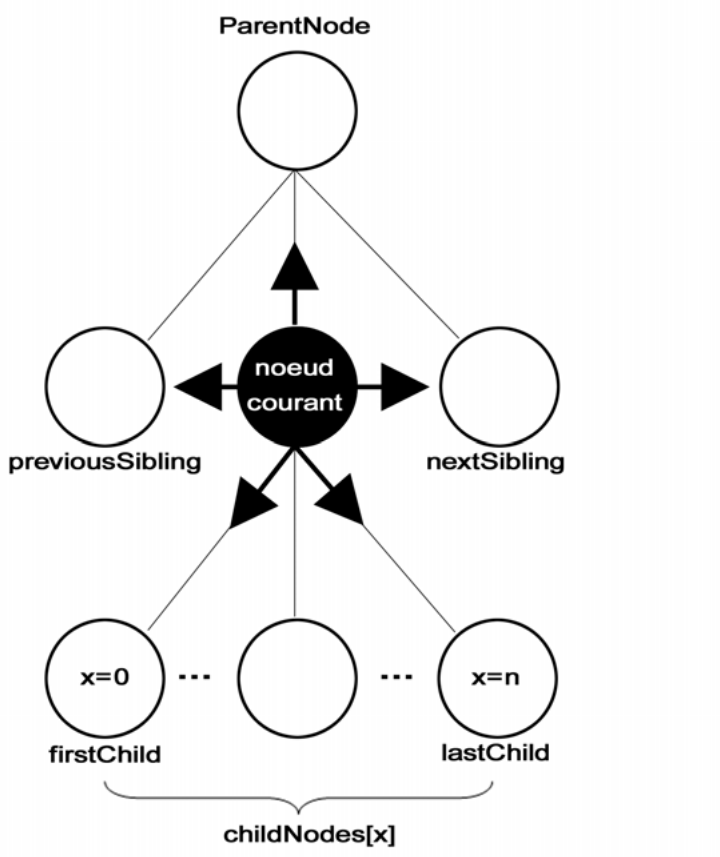
- **Un nœud** (*node*) est le composant de base résultant de la conversion de chaque zone de texte d'un document lors de son chargement dans le navigateur. Il existe différents types de nœuds, dont les principaux sont : élément (type 1), attribut (type 2) ou texte (type 3).

- **Le parent** (*parentNode*) d'un nœud est toujours unique. C'est celui duquel est issu le nœud courant.

- **Les enfants** (*childNodes*) d'un nœud Élément sont les différents nœuds de type élément ou texte issus du nœud élément courant. Parmi les enfants d'un nœud, on peut distinguer le premier enfant (*firstChild*) et le dernier enfant (*lastChild*). À noter qu'un nœud attribut n'a pas de nœud enfant et que celui d'un texte est toujours vide.
- **Les nœuds frères** (*sibling*) sont les nœuds qui ont le même parent. Parmi les frères d'un nœud courant, on peut distinguer le nœud frère immédiatement placé avant (*previousSibling*) le nœud courant et celui placé immédiatement après (*nextSibling*).

Figure 20-2

Relations entre les différents nœuds composant un arbre



Organisation d'un nœud élément XHTML

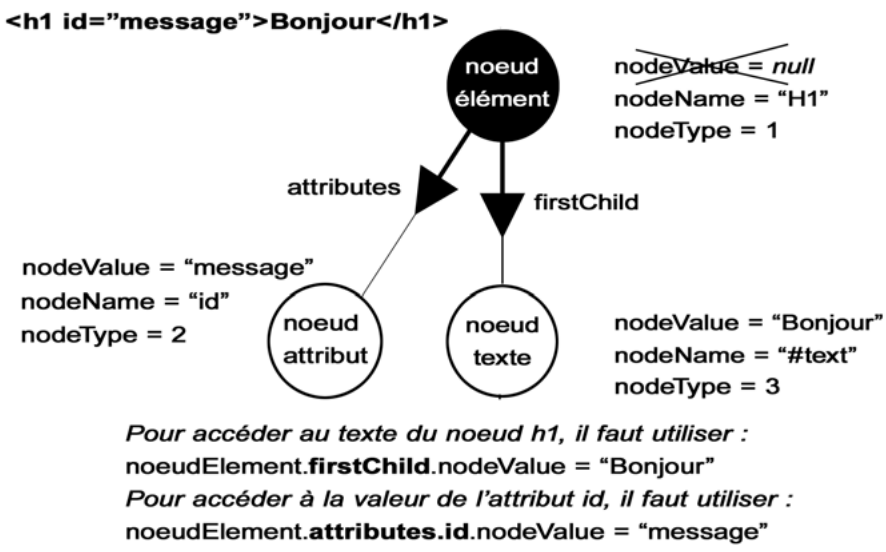
Dans les documents XHTML et XML, nous retrouverons très fréquemment des nœuds de type Élément. Si nous nous référons à la structure d'un élément XHTML (revoir la balise `h1` de la figure 20-1 par exemple), nous constatons qu'elle est composée d'un contenu et d'attributs (ces deux composants étant optionnels).

Si nous transposons cette structure dans l'arbre DOM, nous constatons que son organisation est semblable (voir figure 20-3). Il faut toutefois préciser que si le nœud attribut est bien rattaché au nœud élément, il n'est cependant pas considéré comme un de ses enfants et n'apparaîtra pas dans la liste `childNodes` du nœud élément mais sera accessible par `attributes`.

D'autre part, il est très important de comprendre que le contenu textuel d'un nœud élément n'est pas accessible par son attribut `nodeValue` (qui est égal à `null`) mais par l'intermédiaire d'un nœud de type texte supplémentaire qui aura pour seule fonction de contenir le texte du nœud élément concerné (voir figure 20-3).

Figure 20-3

Nœud élément et ses nœuds attribut et texte



Attributs de balise et attributs de nœud

Attention à ne pas confondre les attributs d'une balise (comme `id`) avec les différents attributs (ou propriétés) d'un objet nœud (comme `nodeValue` ou `nodeName`).

Pour illustrer la transposition d'une page XHTML dans un arbre DOM, reportez vous à la figure 20-4 et au code 20-1 correspondant au code source de la page concernée.

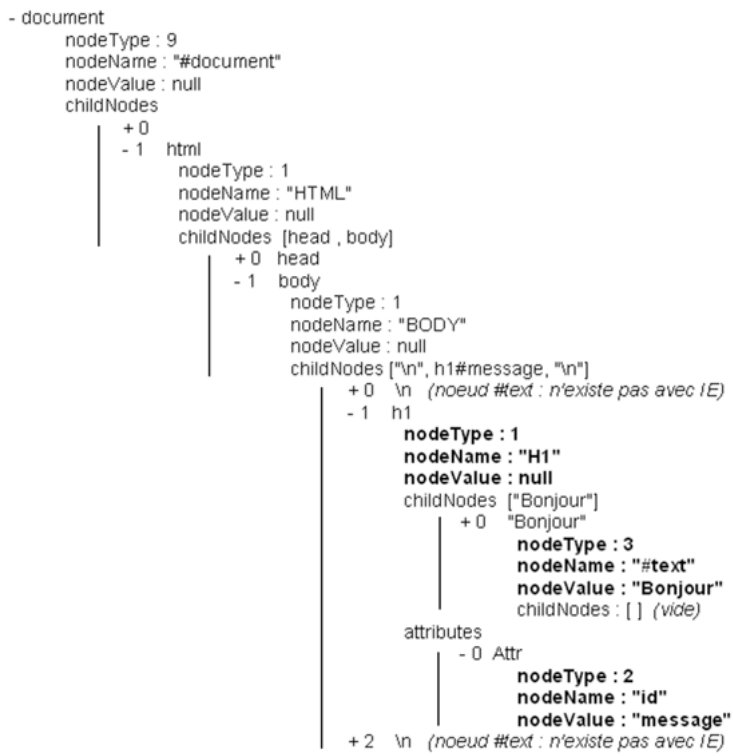
Pour bien comprendre le fonctionnement de l'arbre DOM, nous vous invitons à ouvrir cette page exemple avec Firefox (disponible dans les ressources en ligne du répertoire chapitre 20). Utilisez ensuite l'onglet DOM de l'extension Firebug pour dérouler l'arbre de la même manière que dans la figure 20-4.

Code 20-1 : page exemple `ArbreDOM.html` :

```
<?xml version="1.0"?>
<!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">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Titre de la page</title>
  </head>
  <body>
    <h1 id="message">Bonjour</h1>
  </body>
</html>
```

Figure 20-4

Déroulé de l'arbre correspondant à la page HTML du code 20-1



Connaître les informations d'un nœud

Contraintes d'usage des attributs et méthodes du DOM

Classes de l'objet :

Les différents attributs et méthodes présentés ci-après sont liés à des classes différentes (Node, NodeList, Element ou Document), il faut donc s'assurer que l'objet auquel vous allez les appliquer correspond bien à cette classe.

Document HTML ou XML :

Nous vous avons précisé précédemment que les spécifications du DOM sont découpées en différentes parties. Le DOM Core étant applicable à tous les documents XML et le DOM HTML est spécifique aux documents HTML. Même si la majorité des attributs et méthodes peuvent être utilisés de la même manière avec un document XML et avec un document HTML, il en existe néanmoins certains qui devront être appliqués exclusivement aux documents HTML (comme innerHTML ou les attributs d'affichage : offsetLeft ...).

Pour vous guider dans ce choix, nous rappellerons ces contraintes d'usage au début de chacune des présentations de cette partie.

La liste des attributs et méthodes n'est pas exhaustive

Pour simplifier leur usage, nous avons choisi de ne vous présenter ici que les principaux attributs et méthodes du DOM. Certains d'entre eux n'étant pas portables (utilisables avec tous les navigateurs) ou peu utilisés en pratique ne seront pas présentés mais nous vous invitons à vous référer à la documentation officielle du W3C si vous désirez avoir la liste complète des attributs et des méthodes du DOM.

nodeType : type du nœud

Contraintes d'usage :

Classe : attribut de la classe Node (applicable à tous les nœuds).

Document : XML ou HTML.

L'attribut `nodeType` permet de connaître le type du nœud. La valeur du type est comprise entre 1 et 12 (voir le tableau 20-1 énumérant les 4 principaux types).

Tableau 20-1 Les 4 principales valeurs de nodeType

Valeur de nodeType	Type de nœud correspondant
1	Element
2	Attribute
3	Text
9	Document

Pour récupérer le type d'un nœud, il suffit d'appliquer l'attribut à l'objet nœud en utilisant la syntaxe pointée ci-dessous (`monNoeud` étant l'objet nœud concerné).

Syntaxe :

```
monNoeud.nodeType ;
```

Utiliser nodeType pour filtrer les nœuds séparateur

L'API DOM du navigateur Internet Explorer n'est pas standard et ignore les nœuds texte vides utilisés comme séparateur entre chaque élément contrairement aux autres navigateurs compatibles W3C (comme Firefox). Selon le navigateur utilisé lors de la manipulation d'un nœud, il faut donc souvent tester s'il s'agit du nœud élément ou d'un nœud séparateur pour assurer la compatibilité pour tous les navigateurs (revoir la figure 20-4 pour localiser les nœuds texte qui encadrent le nœud `h1`).

Pour filtrer ces nœuds séparateurs, une solution simple consiste à utiliser une structure de test `if()` comparant le `nodeType` du nœud que l'on désire utiliser avec la valeur du type d'un nœud élément, soit 1 (voir exemple de code ci-dessous). Vous serez ainsi sûr que les instructions de manipulation du nœud placées entre les accolades seront appliquées à un nœud élément et non aux nœuds séparateurs qui l'encadrent (voir aussi le code 20-9 des exemples de la partie sur les attributs qui permettent de se déplacer dans un arbre si vous désirez avoir une application pratique de cette technique) :

```
if(monNoeud.nodeType==1) { … }
```

nodeName : nom du nœud

Contraintes d'usage :

Classe : attribut de la classe Node (applicable à tous les nœuds).

Document : XML ou HTML.

L'attribut `nodeName` permet de connaître le nom du nœud pour certains types spécifiques (attribut ou élément par exemple). Pour les autres types, cet attribut renverra le type du nœud précédé d'un caractère # (document ou texte par exemple). Les valeurs des noms étant différentes pour chaque type de nœud, il existe aussi 12 valeurs possibles de cet attribut selon le type du nœud (voir le tableau 20-2 avec les 4 principales valeurs).

Tableau 20-2 Les 4 principales valeurs de nodeName

Valeur de nodeName	Type de nœud correspondant
Nom de la balise de l'élément	Element
Nom de l'attribut	Attribute
#text	Text
#document	Document

Pour récupérer le nom d'un nœud, il suffit d'appliquer l'attribut à l'objet nœud en utilisant la syntaxe pointée ci-dessous (`monNoeud` étant l'objet nœud concerné).

Syntaxe :

```
monNoeud.nodeName ;
```

nodeValue : valeur du nœud

Contraintes d'usage :

Classe : attribut de la classe Node (applicable à tous les nœuds).

Document : XML ou HTML.

L'attribut `nodeValue` permet de connaître la valeur du nœud pour certains types spécifiques (attribut ou texte par exemple). Pour les autres types, cet attribut renverra l'état `null` (Document ou Element par exemple). Les valeurs des nœuds étant différentes pour chaque type de nœud, il existe aussi 12 valeurs possibles de cet attribut selon le type du nœud (voir le tableau 20-3 avec les 4 principales valeurs).

Tableau 20-3 Les 4 principales valeurs de nodeValue

Valeur de nodeValue	Type de nœud correspondant
null	Element
Valeur de l'attribut	Attribute
Valeur du texte	Text
null	Document

Pour récupérer la valeur d'un nœud, il suffit d'appliquer l'attribut à l'objet nœud en utilisant la syntaxe pointée ci-dessous (`monNoeud` étant l'objet nœud concerné).

Syntaxe :

```
monNoeud.nodeValue ;
```

À noter que pour récupérer le contenu d'un nœud élément, (comme `Bonjour` dans l'exemple du nœud élément suivant `<p>Bonjour</p>`) il ne faut pas utiliser l'attribut de la valeur du nœud élément mais accéder à la valeur du nœud enfant de type texte comme l'illustre le code ci-dessous (`monNoeudElement` étant l'objet nœud de type élément concerné) :

```
monNoeudElement.firstChild.nodeValue ;
```

id : valeur de l'identifiant d'un nœud

Contraintes d'usage :
Classe : attribut de la classe Element (applicable à un objet document exclusivement).
Document : XML ou HTML.

L'attribut `id` permet de connaître la valeur de l'identifiant d'un nœud. Si le nœud n'a pas d'identifiant, l'utilisation de cet attribut ne génère pas d'erreur, dans ce cas la valeur retournée est "".

Pour récupérer l'identifiant d'un nœud, il suffit d'appliquer l'attribut à l'objet nœud en utilisant la syntaxe pointée ci-dessous (`monNoeud` étant l'objet nœud concerné).

Syntaxe :

```
monNoeud.id ;
```

className : valeur de la classe d'un nœud

Contraintes d'usage :
Classe : attribut de la classe Element (applicable à un objet document exclusivement).
Document : XML ou HTML.

L'attribut `className` permet de connaître la valeur de la classe d'un nœud. Si le nœud n'a pas de classe, l'utilisation de cet attribut ne génère pas d'erreur, dans ce cas la valeur retournée est "".

Pour récupérer la classe d'un nœud, il suffit d'appliquer l'attribut à l'objet nœud en utilisant la syntaxe pointée ci-dessous (`monNoeud` étant l'objet nœud concerné).

Syntaxe :

```
monNoeud.className ;
```

offsetXxxx : dimensions et coordonnées d'un Element

Contraintes d'usage :
Classe : attribut de la classe Element (applicable à un objet document exclusivement).
Document : HTML.

Le DOM HTML propose une série d'attributs qui permettent de connaître la dimension (hauteur ou largeur) ou la position (espace séparant l'élément parent des bords de l'élément concerné) d'un nœud élément HTML : voir le tableau 20-4 avec les différents attributs disponibles. Ces attributs concernent la présentation d'un élément, dans ces conditions ils ne peuvent pas être utilisés dans un document XML et ne sont disponibles que pour un élément de document HTML. Les attributs de position d'un élément s'appuient sur son élément parent, aussi il est souvent utile de pouvoir accéder à cet

élément parent. Pour cela, le DOM HTML propose aussi l'attribut `offsetParent` pour accéder à cet élément parent.

Tableau 20-4 Les principaux attributs de position d'un élément

Attribut de l'element	Désignation
offsetHeight	Dimension : hauteur d'un élément
offsetWidth	Dimension : largeur d'un élément
offsetLeft	Position : espace séparant l'élément parent du bord gauche de l'élément concerné
offsetTop	Position : espace séparant l'élément parent du bord supérieur de l'élément concerné
offsetParent	Elément parent : pointe sur l'élément hiérarchiquement supérieur de l'élément concerné. Si il n'y a pas d'élément supérieur, la valeur null est renvoyée.

Pour récupérer la valeur d'une dimension ou d'une position d'un élément HTML, il suffit d'appliquer l'attribut à l'objet `Element` en utilisant la syntaxe pointée ci-dessous (`monElement` étant l'objet `Element` concerné).

Syntaxe :

```
monElement.offsetWidth ;
```

Accéder à un nœud de l'arbre

getElementById(id) : récupère un élément par son identifiant

Contraintes d'usage :
Classe : méthode de la classe Document (applicable à un objet document exclusivement).
Document : XML ou HTML.

Il est facile de référencer un objet `Element` si l'on connaît son identifiant à l'aide de la méthode `getElementById()`. L'identifiant d'un élément étant toujours unique dans un même document, il suffit d'appeler cette méthode en passant le `id` de l'élément en paramètre. La méthode retourne alors l'élément unique correspondant.

Syntaxe :

```
document.getElementById(nomDeIdentifiant) ;
```

Dans l'exemple ci-dessous, nous utilisons cette technique pour afficher le texte contenu dans un élément `div` d'identifiant « message ». L'information sera affichée dans la console de Firebug.

Code JavaScript 20-2 :

```
console.info('Le message est:'+ document
➥.getElementById("message").firstChild.nodeValue);
```

Affiche dans la console de Firebug la ligne suivante :

```
Le message est: Bonjour
```

Si la balise ci-dessous est dans la page HTML

```
<div id="message">Bonjour</div>
```

Attention :

Avec cette méthode, nous réaliserons toujours la recherche dans tout le document (applicable uniquement à la classe Document). Il ne faut donc pas oublier de faire précéder l'appel de la méthode du préfixe « document ».

getElementsByTagName(tagName) : récupère la liste d'éléments d'une même balise

Contraintes d'usage :

Classe : méthode de la classe Document ou Element (applicable à un objet document ou à un objet élément).

Document : XML ou HTML.

La méthode qui permet d'accéder à un élément par son identifiant étant limitée à un élément unique (un même `id` ne peut être appliqué qu'à un seul élément), il est quelquefois intéressant de pouvoir accéder à une liste d'éléments possédant le même nom de balise.

Attention au « s » :

Contrairement à la méthode getElementById(id) qui ne renverra toujours qu'un seul élément car un id est unique dans un document, il n'en est pas de même avec les méthodes getElementsByTagName(tagName) et getElementsByName(name) qui, elles utilisent le nom d'une balise ou celui de l'élément pour les recherches et pourront donc renvoyer une liste de plusieurs éléments correspondants aux critères. Il est donc normal que les noms des méthodes "getElements…" prennent un « s » dans ce cas : ne l'oubliez pas dans vos futurs programmes.

La méthode `getElementsByTagName()` permet d'effectuer ce type de recherche, il suffit d'appeler cette méthode en passant le nom de la balise recherchée en paramètre. La méthode retourne alors un tableau des différents éléments correspondants.

Syntaxe :

```
document.getElementsByTagName(nomDeBalise) ;
```

Parcourir une liste de nœuds

Pour parcourir tous les nœuds du tableau retourné par la méthode getElementsByTagName() ou getElementsByName(), vous pouvez par exemple utiliser le code ci-dessous (dans notre exemple la liste a été préalablement sauvegardée dans une variable listeElements) :

```
for(var i=0 ; i<listeElements.length ; i++) {
  var element = listeElements[i];
  console.info("L'élément "+i+" est "+ element);
}
```

Recherche dans tout le document

Dans l'exemple ci-dessous (voir fichier `exempleTagName1.html`), nous utilisons cette technique pour rechercher, dans le document entier, tous les éléments dont le nom de balise est `li` (dans ce cas, nous appliquerons la méthode à l'objet `document`).

Code JavaScript 20-3 :

```
function test(){
  var listeElements=document.getElementsByTagName("li");
  for(var i=0 ; i<listeElements.length ; i++) {
    var element = listeElements[i];
    console.info("L'élément "+i+" est "+ element.firstChild.nodeValue);
  }
}
window.onload = test;
```

Affiche dans la console les lignes ci-dessous :

```
L'élément 0 est : option A1
L'élément 1 est : option A2
L'élément 2 est : option B1
L'élément 3 est : option B2
```

Si les balises ci-dessous sont dans la page HTML :

```
<ul id="listeA" >
  <li>option A1</li>
  <li>option A2 </li>
</ul>
<ul id="listeB" >
  <li>option B1</li>
  <li>option B2 </li>
</ul>
```

Recherche dans un élément particulier

Cette même méthode peut aussi s'appliquer à un objet `Element` spécifique. Dans ce cas, il faudra appliquer cette méthode directement à l'élément et la recherche sera effectuée uniquement sur les éléments qui constituent la descendance de l'élément concerné.

Si nous appliquons cette seconde technique à l'exemple précédent, le code JavaScript serait alors le suivant (voir fichier `exempleTagName2.html`).

Code JavaScript 20-4 :

```
function test(){
  var zoneRecherche=document.getElementById("listeA");
  var listeElements=zoneRecherche.getElementsByTagName("li");
  for(var i=0 ; i<listeElements.length ; i++) {
   var element = listeElements[i];
   console.info("L'élément "+i+" est "+ element.firstChild.nodeValue);
  }
}
window.onload = test;
```

Affiche dans la console les lignes ci-dessous :

```
L'élément 0 est : option A1
L'élément 1 est : option A2
```

Si les balises dans la page HTML sont les mêmes que dans le précédent exemple.

getElementsByName(name) : récupère la liste d'éléments portant le même nom

Contraintes d'usage :
Classe : méthode de la classe Document (applicable à un objet document exclusivement).
Document : XML ou HTML.

La méthode `getElementsByName()` permet d'effectuer une recherche sur la valeur de l'attribut `name` d'un élément. Il suffit d'appeler cette méthode en passant la valeur recherchée en paramètre. La méthode retourne alors un tableau des différents éléments correspondants.

Syntaxe :

```
document.getElementsByName(nomDeElement) ;
```

Dans l'exemple ci-dessous (voir fichier `exempleName.html`), nous utilisons cette technique pour rechercher tous les éléments d'une série de boutons radio dans le document (les attributs `name` des boutons de cette série seront donc identiques et égaux à la valeur `choix`). Nous profiterons aussi de la boucle `for` pour faire un test à chaque tour de boucle et vérifier si l'un des boutons radio est coché.

Code JavaScript 20-5 :

```
function test(){
  var listeElements=document.getElementsByName("choix");
  for(var i=0 ; i<listeElements.length ; i++) {
    var element = listeElements[i];
    console.info("L'élément "+i+" est "+ element.value);
    if(element.checked)
      console.info("L'élément sélectionné est "+ element.value);
 }
}
window.onload = test;
```

Affiche dans la console les lignes ci-dessous :

```
L'élément 0 est : choix1
L'élément 1 est : choix2
L'élément 2 est : choix3
```

Si les balises dans la page HTML sont les suivantes :

```
<p id="zoneChoix">
  <input type="radio" name="choix" value="choix1" />Choix1<br />
  <input type="radio" name="choix" value="choix2" />Choix2 <br />
  <input type="radio" name="choix" value="choix3" />Choix3 <br />
</p>
```

Lors du chargement initial de la page, aucun des choix n'est encore sélectionné. L'instruction de test `if(element.checked)` renverra donc une réponse négative à chaque tour de la boucle `for` car aucun bouton radio n'est coché. Par contre, si vous sélectionnez maintenant le choix 2, par exemple, et que vous réactualisez la page, vous obtiendrez alors les lignes ci-dessous :

```
L'élément 0 est : choix1
L'élément 1 est : choix2
L'élément sélectionné est : choix2
L'élément 2 est : choix3
```

getAttribute(attributeName) : récupère la valeur d'un attribut

Contraintes d'usage :
Classe : méthode de la classe Element (applicable à un objet élément exclusivement).
Document : XML ou HTML.

La méthode `getAttribute()` permet de récupérer la valeur d'un attribut. Il suffit d'appeler cette méthode en passant le nom de l'attribut recherché en paramètre (`attributeName`). La méthode retourne alors la valeur correspondante à l'attribut.

Syntaxe :

```
elementPere.getAttribute(nomDeAttribut) ;
```

Dans l'exemple ci-dessous (voir fichier `exempleGetAttribute.html`), nous utilisons cette technique pour rechercher la valeur de l'attribut `value` de la (ou les) case(s) à cocher sélectionnée(s).

Pour cela nous utiliserons la méthode `getElementsByTagName("input")` pour récupérer la liste des éléments `input` du document. Ensuite, nous récupérons la valeur de chaque case dans une variable `valeurCase` avec la méthode `element.getAttribute("value")`. Enfin, nous testons si la case est cochée et affichons dans ce cas la valeur de la case concernée.

À noter qu'une autre alternative plus simple serait d'utiliser l'attribut `value` des éléments pour récupérer directement la valeur de l'attribut au lieu de la méthode `element.getAttribute("value")`. Aussi, en pratique, cette méthode est principalement utilisée avec des documents XML pour lesquels les attributs d'un élément ne sont pas directement accessibles par le biais ses propriétés DOM.

Code JavaScript 20-6 :

```
function test(){
   var listeElements=document.getElementsByTagName("input");
   for(var i=0 ; i<listeElements.length ; i++) {
     var element = listeElements[i];
     var valeurCase = element.getAttribute("value");
     if(element.checked) console.info("L'élément "+valeurCase+" est sélectionné ");
   }
}
window.onload = test;
```

Affiche dans la console les lignes ci-dessous si la case de l'article B est cochée :

```
L'élément articleB est sélectionné
```

Si les balises du code source de la page HTML sont les suivantes :

```
<input name="choix1" type="checkbox" id="choix1" value="articleA" />
article A<br />
<input name="choix2" type="checkbox" id="choix2" value="articleB" />
article B<br />
```

length : indique le nombre d'élément d'une liste de nœuds

Contraintes d'usage :

Classe : attribut de la classe NodeList (applicable exclusivement sur un objet NodeList).

Document : XML ou HTML.

Les deux méthodes `getElementsByName()` et `getElementsByTagName()` que nous venons de présenter, mais aussi l'attribut `childNodes` que nous verrons plus loin, retournent une liste d'éléments résultat dans un objet `NodeList`. Pour exploiter les éléments de cette liste, il est pratique de connaître le nombre d'éléments contenus dans le tableau `NodeList` (comme nous l'avons déjà utilisé dans le script pour parcourir la liste de nœuds, revoir encadré précédent). Pour obtenir cette information, il suffit d'appliquer l'attribut `length` à l'objet `NodeList` comme dans l'exemple ci-dessous (voir fichier `exempleLength.html`) :

Syntaxe :

```
listeElements.length ;
```

Code JavaScript 20-7 :

```
var listeElements=document.getElementsByTagName("li");
console.info("Le nombre d'éléments li est "+ listeElements.length);
```

Balises de la page HTML :

```
<ul id="listeA" >
  <li>option A1</li>
  <li>option A2 </li>
</ul>
```

La console affichera alors la ligne suivante :

```
Le nombre d'éléments li est 2
```

Se déplacer dans les nœuds de l'arbre

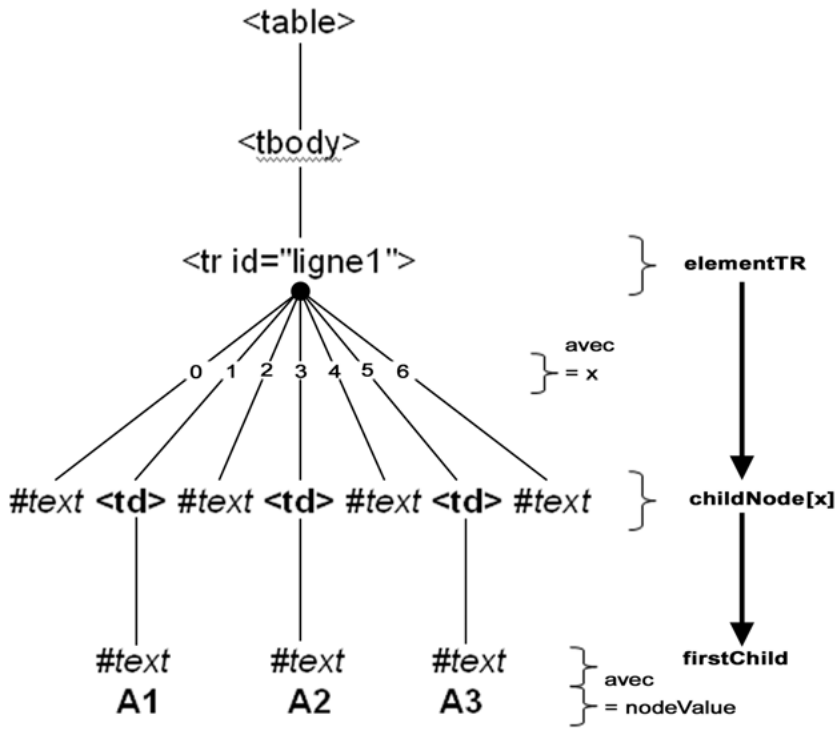
Les attributs de cette partie permettent de se déplacer dans l'arbre d'un document. Afin de bien comprendre le fonctionnement de ces différents attributs, nous vous conseillons de revoir la partie consacrée à la terminologie d'un arbre DOM présentée au début de ce chapitre.

Dans les exemples de cette partie, nous allons utiliser un tableau HTML de 3 cellules. La figure 20-5 ci-dessous illustre la hiérarchie des éléments qui composent ce tableau afin

que vous puissiez plus facilement vous repérer dans les déplacements effectués dans l'arbre grâce aux différents attributs présentés.

Figure 20-5

Hiérarchie de la structure du tableau HTML utilisée dans les exemples de cette partie



childNodes : récupère la liste des nœuds enfants

Contraintes d'usage :

Classe : attribut de la classe Node (applicable à tous les nœuds).

Document : XML ou HTML.

L'attribut `childNodes` permet de récupérer la liste des nœuds enfants d'un élément dans un tableau de variables. Si l'élément concerné n'a pas d'enfant, l'attribut renvoie alors un tableau vide.

Syntaxe :

```
elementPere.childNodes[indice] ;
```

Dans l'exemple ci-dessous (voir fichier `exempleChildNodes.html`) nous affichons successivement dans une fenêtre d'alerte tous les éléments récupérés à l'aide de l'attribut `childNodes` appliqué à l'élément TR d'un tableau HTML de trois cellules. Nous effectuerons une première série de tests avec le navigateur Firefox puis avec le navigateur Internet Explorer (en utilisant l'extension `IE Tab` de Firefox par exemple).

Code JavaScript 20-8 :

```
function test(){
  var elementTR=document.getElementById("ligne1");
  for(var i=0 ; i<elementTR.childNodes.length ; i++) {
    var elementTD = elementTR.childNodes[i];
    alert("L'élément "+i+" est "+elementTR.childNodes[i].nodeName);
  }
}
window.onload = test;
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
  <tr id="ligne1">
    <td>A1</td>
    <td>A2</td>
    <td>A3</td>
  </tr>
</table>
```

Résultats obtenus avec Firefox :

```
L'élément 0 est #text
L'élément 1 est TD
L'élément 2 est #text
L'élément 3 est TD
L'élément 4 est #text
L'élément 5 est TD
L'élément 6 est #text
```

Résultats obtenus avec Internet Explorer :

```
L'élément 0 est TD
L'élément 1 est TD
L'élément 2 est TD
```

Comme nous l'avons vu précédemment, l'arbre DOM des nœuds étant différent avec le navigateur Internet Explorer (IE ignore les nœuds texte séparateurs alors que Firefox les intègre dans l'arbre), il convient de modifier notre code en faisant un test sur le type de nœud avant de leur appliquer le traitement voulu (voir fichier `exempleChildNodes2.html`).

Désormais, comme nous sommes sûr de n'avoir que des éléments TD lors du traitement, nous pouvons maintenant exploiter sans générer d'erreur la valeur du contenu de chaque cellule (`firstChild.nodeValue`**)** en l'affichant dans chaque message de la boîte d'alerte.

Code JavaScript 20-9 :

```
function test(){
  var elementTR=document.getElementById("ligne1");
  for(var i=0 ; i<elementTR.childNodes.length ; i++) {
    var elementTD = elementTR.childNodes[i];
    if(elementTD.nodeType == 1)
```

```
    alert("L'élément "+elementTR.childNodes[i].nodeName+" contient la valeur
    ➥"+elementTR.childNodes[i].firstChild.nodeValue);
  }
}
window.onload = test;
```

Si nous renouvelons nos essais avec les deux navigateurs, nous constatons que nous obtenons maintenant des résultats identiques pour les deux navigateurs :

```
L'élément TD contient la valeur A1
L'élément TD contient la valeur A2
L'élément TD contient la valeur A3
```

parentNode : retourne le nœud parent

Contraintes d'usage :
Classe : attribut de la classe Node (applicable à tous les nœuds).
Document : XML ou HTML.

L'attribut `parentNode` d'un nœud retourne son nœud parent. Si toutefois le nœud concerné n'a pas de nœud parent (ce qui est rare), l'attribut retourne l'état `null`.

Syntaxe :

```
noeud.parentNode ;
```

Dans l'exemple ci-dessous (voir fichier `exempleParentNode.html`) nous affichons successivement dans la console de Firebug l'identifiant commun de l'élément TR, père des 3 cellules TD du tableau HTML.

Code JavaScript 20-10 :

```
function test(){
  var listeTD=document.getElementsByTagName("td");
  for(var i=0 ; i<listeTD.length ; i++) {
   var elementTD = listeTD[i];
   var parentDeTD = elementTD.parentNode;
   console.info("L'élément parent de la cellule TD "+elementTD.firstChild.nodeValue+"
   ➥est identifié par le id : "+ parentDeTD.id);
  }
}
window.onload = test;
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
  <tr id="ligne1">
    <td>A1</td>
    <td>A2</td>
    <td>A3</td>
  </tr>
</table>
```

Résultats obtenus :

```
L'élément parent de la cellule TD A1 est identifié par le id : ligne1
L'élément parent de la cellule TD A2 est identifié par le id : ligne1
L'élément parent de la cellule TD A3 est identifié par le id : ligne1
```

nextSibling : retourne le nœud frère suivant

Contraintes d'usage :
Classe : attribut de la classe Node (applicable à tous les nœuds).
Document : XML ou HTML.

L'attribut `nextSibling` d'un nœud retourne le nœud frère situé immédiatement après le nœud courant. Si le nœud concerné n'a pas de nœud suivant, l'attribut retourne l'état `null`.

Syntaxe :

```
noeudCourant.nextSibling ;
```

À noter qu'ici aussi, il faut tenir compte de la différence des arbres DOM générés par les navigateurs (IE ignore les nœuds texte séparateurs alors que Firefox les intègre dans l'arbre). Pour pallier ce problème nous ajoutons un test du type de l'élément retourné par le premier attribut `nextSibling`. Si celui-ci n'est pas un élément (nœud de type 1), nous appliquons un second déplacement au nœud suivant pour court-circuiter le nœud séparateur de Firefox.

Il existe une autre solution pour que votre traitement de l'arbre puisse fonctionner sous IE comme sous Firefox. Cette technique réalise un test de l'objet `all` spécifique à Microsoft qui n'est défini que sous IE. La valeur du test (`document.all`) renverra donc `true` avec IE et `false` pour les autres navigateurs. Ainsi, avec cette technique, vous pourrez dissocier le traitement à effectuer avec IE et celui de Firefox comme dans l'exemple du code 20-12 qui représente une solution alternative au code 20-11.

Dans l'exemple ci-dessous (voir fichier `exempleNextSibling.html`) nous affichons le nœud suivant du nœud d'identifiant `celluleA2` dans la console de Firebug.

Code JavaScript 20-11 :

```
function test(){
  var noeudCourant=document.getElementById("celluleA2");
  var noeudSuivant=noeudCourant.nextSibling ;
  if(noeudSuivant.nodeType != 1)
    var noeudSuivant=noeudSuivant.nextSibling ;
  console.info("L'élément suivant de la cellule A2 est "+noeudSuivant.id);
}
window.onload = test;
```

Code JavaScript 20-12 : solution alternative au code 20-11

```
function test(){
  var noeudCourant=document.getElementById("celluleA2");
  if(document.all)
   var noeudSuivant=noeudCourant.nextSibling ;
  else
  var noeudSuivant= noeudCourant.nextSibling.nextSibling ;
```

```
 console.info("L'élément suivant de la cellule A2 est "+noeudSuivant.id);
}
window.onload = test;
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
  <tr id="ligne1">
   <td id="celluleA1">A1</td>
   <td id="celluleA2">A2</td>
   <td id="celluleA3">A3</td>
  </tr>
</table>
```

Résultat obtenu dans la console :

```
L'élément suivant de la cellule A2 est celluleA3
```

previousSibling : retourne le nœud frère précédent

Contraintes d'usage :
Classe : attribut de la classe Node (applicable à tous les nœuds).
Document : XML ou HTML.

L'attribut `previousSibling` d'un nœud retourne le nœud frère situé immédiatement avant le nœud courant. Si le nœud concerné n'a pas de nœud précédent, l'attribut retourne l'état `null`.

Syntaxe :

```
noeudCourant.previousSibling
```

À noter qu'ici aussi, il faut tenir compte de la différence des arbres DOM générés par les navigateurs (IE ignore les nœuds texte séparateurs alors que Firefox les intègre dans l'arbre). Pour pallier ce problème nous ajoutons un test du type de l'élément retourné par le premier attribut `previousSibling`. Si celui-ci n'est pas un élément (nœud de type 1), nous appliquons un second déplacement au nœud précédent pour court-circuiter le nœud séparateur de Firefox.

Dans l'exemple ci-dessous (voir fichier `exemplePreviousSibling.html`) nous affichons le nœud précédent du nœud d'identifiant `celluleA2` dans la console de Firebug.

Code JavaScript 20-13 :

```
function test(){
 var noeudCourant=document.getElementById("celluleA2");
 var noeudPrecedent=noeudCourant.previousSibling ;
 if(noeudPrecedent.nodeType != 1)
   var noeudPrecedent=noeudPrecedent.previousSibling ;
 console.info("L'élément précédent de la cellule A2 est "+noeudPrecedent.id);
}

window.onload = test;
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
 <tr id="ligne1">
```

```
  <td id="celluleA1">A1</td>
  <td id="celluleA2">A2</td>
  <td id="celluleA3">A3</td>
 </tr>
</table>
```

Résultat obtenu dans la console :

```
L'élément précédent de la cellule A2 est celluleA1
```

firstChild : retourne le premier nœud enfant

Contraintes d'usage :
Classe : attribut de la classe Node (applicable à tous les nœuds).
Document : XML ou HTML.

L'attribut `firstChild` d'un nœud retourne le premier nœud enfant du nœud courant. Si le nœud concerné n'a pas de nœud enfant, l'attribut retourne l'état `null`.

Syntaxe :

```
noeudPere.firstChild ;
```

À noter qu'ici aussi, il faut tenir compte de la différence des arbres DOM générés par les navigateurs (IE ignore les nœuds texte séparateurs alors que Firefox les intègre dans l'arbre). Pour pallier ce problème nous ajoutons un test du type de l'élément retourné par l'attribut `firstChild`. Si celui-ci n'est pas un élément (nœud de type 1), nous appliquons un déplacement au nœud suivant avec l'attribut `nextSibling` pour court-circuiter le nœud séparateur de Firefox.

Dans l'exemple ci-dessous (voir fichier `exempleFirstChild.html`) nous affichons le premier nœud enfant du nœud d'identifiant `ligne1` dans la console de Firebug.

Code JavaScript 20-14 :

```
function test(){
 var noeudCourant=document.getElementById("celluleA2");
 var noeudPrecedent=noeudCourant.previousSibling ;
 if(noeudPrecedent.nodeType != 1)
   var noeudPrecedent=noeudPrecedent.previousSibling ;
 console.info("L'élément précédent de la cellule A2 est "+noeudPrecedent.id);
}
window.onload = test;
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
 <tr id="ligne1">
  <td id="celluleA1">A1</td>
  <td id="celluleA2">A2</td>
  <td id="celluleA3">A3</td>
 </tr>
</table>
```

Résultat obtenu dans la console :

```
L'élément articleB est sélectionné
```

lastChild : retourne le dernier nœud enfant

Contraintes d'usage :
Classe : attribut de la classe Node (applicable à tous les nœuds).
Document : XML ou HTML.

L'attribut `lastChild` d'un nœud retourne le dernier nœud enfant du nœud courant. Si le nœud concerné n'a pas de nœud enfant, l'attribut retourne l'état `null`.

Syntaxe :

```
noeudPere.lastChild
```

À noter qu'ici aussi, il faut tenir compte de la différence des arbres DOM générés par les navigateurs (IE ignore les nœuds texte séparateurs alors que Firefox les intègre dans l'arbre). Pour pallier ce problème nous ajoutons un test du type de l'élément retourné par l'attribut `lastChild`. Si celui-ci n'est pas un élément (nœud de type 1), nous appliquons un déplacement au nœud précédent avec l'attribut `previousSibling` pour court-circuiter le nœud séparateur de Firefox.

Dans l'exemple ci-dessous (voir fichier `exempleLastChild.html`) nous affichons le dernier nœud enfant du nœud d'identifiant `ligne1` dans la console de Firebug.

Code JavaScript 20-15 :

```
function test(){
  var noeudCourant=document.getElementById("ligne1");
  var dernierEnfant=noeudCourant.lastChild ;
  if(dernierEnfant.nodeType != 1)
   var dernierEnfant=dernierEnfant.previousSibling ;
  console.info("Le dernier élément enfant de la ligne 1 est "+dernierEnfant.id);
}
window.onload = test;
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
  <tr id="ligne1">
   <td id="celluleA1">A1</td>
   <td id="celluleA2">A2</td>
   <td id="celluleA3">A3</td>
  </tr>
</table>
```

Résultat obtenu dans la console :

```
L'élément articleB est sélectionné
```

hasChildNodes : retourne true s'il y a des nœuds enfants

Contraintes d'usage :
Classe : attribut de la classe Node (applicable à tous les nœuds).
Document : XML ou HTML.

L'attribut `hasChildNodes` d'un nœud retourne la valeur booléenne `true` si le nœud courant possède au moins un nœuds enfants et `false` dans le cas contraire.

Syntaxe :

```
noeudPere.hasChildNodes
```

Dans l'exemple ci-dessous (voir fichier `exempleHasChildNodes.html`) nous vérifierons que le nœud d'identifiant `ligne1` a bien des enfants avant de lui appliquer un traitement. La réponse du test est affichée dans la console de Firebug.

Code JavaScript 20-16 :

```
function test(){
  var noeudCourant=document.getElementById("ligne1");
  if(noeudCourant.hasChildNodes)
  {
    console.info("Nous confirmons que l'élément de la ligne 1 a bien des enfants");
    var dernierEnfant=noeudCourant.lastChild ;
    if(dernierEnfant.nodeType != 1)
      var dernierEnfant=dernierEnfant.previousSibling ;
    console.info("Le dernier élément enfant de la ligne 1 est "+dernierEnfant.id);
  }
  else
    console.info("Attention, l'élément de la ligne 1 n'a pas d'enfants");
}
window.onload = test;
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
 <tr id="ligne1">
  <td id="celluleA1">A1</td>
  <td id="celluleA2">A2</td>
  <td id="celluleA3">A3</td>
 </tr>
</table>
```

Résultat obtenu dans la console :

```
Nous confirmons que l'élément de la ligne 1 a bien des enfants
Le dernier élément enfant de la ligne 1 est celluleA3
```

Créez des fonctions de navigation dans l'arbre pour capitaliser vos codes

Afin de capitaliser son code et éviter de refaire les mêmes procédures à chaque utilisation d'un attribut DOM, il est intéressant de créer ses propres fonctions de déplacement en intégrant évidemment les tests du type des éléments retournés afin que la fonction puisse être utilisée avec tous les navigateurs de la même manière.

Pour illustrer la création de ce type de fonction, nous vous proposons de créer une fonction qui permet d'accéder au dernier élément enfant d'un nœud particulier qui sera passé en paramètre de la fonction (voir code 20-17). Vous pourrez ensuite créer vous-même sur ce même modèle d'autres fonctions utilisables avec tous les navigateurs pour vous déplacer dans l'arbre DOM.

Code JavaScript 20-17 : fonction `dernierEnfant(nœudCourant)` :

```
function dernierEnfant(noeudCourant){
  if(noeudCourant.hasChildNodes)
  {
```

```
    var dernierEnfant=noeudCourant.lastChild ;
    if(dernierEnfant.nodeType != 1)
      var dernierEnfant=dernierEnfant.previousSibling ;
  }
  else
   dernierEnfant= null;
  return dernierEnfant;
}
```

Pour exploiter cette fonction, vous pouvez par exemple utiliser le code suivant :

Code JavaScript 20-18 :

```
function test(){
  var noeudCourant=document.getElementById("ligne1");
  dernierEnfant=dernierEnfant(noeudCourant);
  console.info("Le dernier élément enfant de la ligne 1 est "+dernierEnfant.id);
}
window.onload = test;
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
 <tr id="ligne1">
  <td id="celluleA1">A1</td>
  <td id="celluleA2">A2</td>
  <td id="celluleA3">A3</td>
 </tr>
</table>
```

Résultat obtenu dans la console :

```
Le dernier élément enfant de la ligne 1 est celluleA3
```

Pour information, il existe une seconde alternative au code 20-17 qui s'appuyait sur le type de l'élément à traiter pour solutionner les problème de compatibilité entre IE et Firefox. Celle-ci utilise une structure `if` qui teste l'objet `all` spécifique à Microsoft (voir code 20-19). Ainsi le test (`document.all`) renvoie une valeur `true` uniquement dans le cas d'un navigateur IE et nous pouvons ainsi appliquer un traitement différent selon le type de navigateur.

Code JavaScript 20-19 :

```
function dernierEnfant(noeudCourant){
 if(noeudCourant.hasChildNodes)
 {
  //pour IE
   if (document.all) var dernierEnfant=noeudCourant.lastChild ;
  //pour FF
   else var dernierEnfant=noeudCourant.lastChild.previousSibling ;
 }
   else
  dernierEnfant= null;
 return dernierEnfant;
}
```

Modifier les nœuds de l'arbre

Dans les parties précédentes, nous avons vu comment récupérer des informations d'un nœud et se déplacer de nœud en nœud dans un arbre DOM, nous allons maintenant vous présenter les différentes méthodes qui vont vous permettre de modifier l'arbre en agissant sur les éléments (création, modification, duplication, suppression), sur les attributs des éléments, sur le contenu des éléments voire même directement sur les styles des éléments.

createElement(nomBalise) : création d'un élément

Contraintes d'usage :
Classe : attribut de la classe Document (applicable au nœud document exclusivement).
Document : XML ou HTML.

La méthode `createElement(nomBalise)` permet de créer un nœud de type élément et dont le nom de la balise sera le texte passé en paramètre lors de l'appel de la méthode. Dès sa création, il est possible de modifier ses attributs mais il faudra ensuite le rattacher à l'arbre par une méthode `appendChild()` ou `insertBefore()` pour qu'il apparaisse dans la page HTML.

Syntaxe :

```
document.createElement(nomDeLaBalise) ;
```

Exemple de création d'un nouvel élément dont le nom de balise sera TD suivi de la configuration de son attribut id :

Code JavaScript 20-20 :

```
elementTD=document.createElement("TD");
elementTD.id="celluleA4";
```

createTextNode(contenu) : création d'un nœud texte

Contraintes d'usage :
Classe : attribut de la classe Document (applicable au nœud document exclusivement).
Document : XML ou HTML.

La méthode `createTextNode(contenu)` permet de créer un nœud de type texte et dont son contenu sera le texte passé en paramètre lors de l'appel de la méthode. Dès sa création, il est possible de modifier ses attributs mais il faudra ensuite le rattacher à un élément déjà présent dans l'arbre par une méthode `appendChild()` ou `insertBefore()` pour qu'il apparaisse dans la page HTML.

Syntaxe :

```
document.createTextNode(contenuDuTexte) ;
```

Exemple de création d'un nouveau nœud texte dont le contenu est "A4" :

Code JavaScript 20-21 :

```
noeudTexte=document.createTextNode("A4");
```

setAttribute(nom,valeur) : création ou modification d'un attribut

Contraintes d'usage :

Classe : attribut de la classe Element (applicable au nœud élément exclusivement).

Document : XML ou HTML.

La méthode `setAttribute(nom,valeur)` permet de créer ou de modifier l'attribut d'un nœud de type élément. Pour cela deux paramètres seront nécessaires, le premier (`nom`) indiquera le nom de l'attribut à créer ou à modifier et le second (`valeur`) contiendra la valeur à donner à l'attribut précédemment nommé.

Syntaxe :

```
noeudPere.setAttribute(nomDeAttribut,valeurDeAttribut)
```

Pour illustrer l'utilisation de cette méthode, nous vous proposons de créer un attribut de classe à l'élément de la cellule A2 (voir fichier `exempleSetAttribute.html`). Cette action sera déclenchée par l'appui sur un bouton qui changera la couleur du fond de la cellule concernée en rouge.

À noter que la gestion des classes étant différente sous IE et Firefox, nous devrons doubler la commande afin que la modification soit effective pour les deux navigateurs (les méthodes ne générant pas d'erreur dans IE comme dans Firefox, il n'est pas nécessaire d'utiliser un détecteur de navigateur comme `if(document.all)` dans ce cas).

Code JavaScript 20-22 :

```
function test(){
 elementTD=document.getElementById("celluleA2");
 elementTD.setAttribute("class","classeCellule");//pour FF
 elementTD.setAttribute("className","classeCellule");//pour IE
}
```

Style de la page :

```
.classeCellule {
  background-color: #FF0000;
}
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
 <tr id="ligne1">
  <td id="celluleA1">A1</td>
  <td id="celluleA2">A2</td>
  <td id="celluleA3">A3</td>
 </tr>
</table>
<input type="button" value="CHANGE LE STYLE DE A2" onclick="test();">
```

appendChild(noeud) : insertion d'un nœud après le dernier enfant

Contraintes d'usage :

Classe : attribut de la classe Node (applicable à tous les nœuds).

Document : XML ou HTML.

La méthode `appendChild(noeud)` permet d'insérer le nœud passé en paramètre après le dernier nœud enfant du nœud auquel il est appliqué.

Syntaxe :

```
noeudPere.appenChild(noeud) ;
```

Pour illustrer son usage, nous allons l'exploiter pour relier le nœud texte créé précédemment au nœud élément dans un premier temps. Puis nous insérerons l'ensemble après le dernier élément enfant de l'élément TR du tableau (soit donc après la cellule A3).

Ce traitement étant déclenché par l'appui sur un bouton, vous pourrez constater son action lors de vos tests par l'ajout d'une cellule supplémentaire nommée A4 après la cellule A3 (voir fichier `exempleAppendChild.html`).

Code JavaScript 20-23 :

```
function test(){
 //création du noeud élément
 elementTD=document.createElement("TD");
 elementTD.id="celluleA4";
 //création du noeud texte
 noeudTexte=document.createTextNode("A4");
 //attachement du noeud texte au noeud élément
 elementTD.appendChild(noeudTexte);
 //attachement du noeud élément à la suite des autres cellules
 elementTR=document.getElementById("ligne1");
 elementTR.appendChild(elementTD);
}
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
 <tr id="ligne1">
  <td id="celluleA1">A1</td>
  <td id="celluleA2">A2</td>
  <td id="celluleA3">A3</td>
 </tr>
</table>
<input type="button" value="AJOUTER UNE CELLULE" onclick="test();">
```

insertBefore(nouveauNoeud,noeud) : insertion d'un nœud avant un autre nœud

Contraintes d'usage :

Classe : attribut de la classe Node (applicable à tous les nœuds).

Document : XML ou HTML.

La méthode `insertBefore(nouveauNoeud,noeud)` permet d'insérer le nœud passé dans le premier paramètre (`nouveauNoeud`) avant un autre nœud, qui lui, sera passé dans le second paramètre de la méthode (`noeud`).

Syntaxe :

```
noeudPere.insertBefore(nouveauNoeud,noeud) ;
```

Pour la démonstration, nous allons reprendre la base de l'exemple précédent mais nous allons cette fois ajouter la nouvelle cellule A4 avant la première cellule A1 du tableau.

Ce traitement étant déclenché par l'appui sur un bouton, vous pourrez constater son action lors de vos tests par l'ajout d'une cellule supplémentaire nommée A4 avant la cellule A1 (voir fichier `exempleInsertBefore.html`).

Code JavaScript 20-24 :

```
function test(){
 //création du noeud élément
 elementTD=document.createElement("TD");
 elementTD.id="celluleA4";
 //création du noeud texte
 noeudTexte=document.createTextNode("A4");
 //attachement du noeud texte au noeud élément
 elementTD.appendChild(noeudTexte);
 //attachement du noeud élément à la suite des autres cellules
 elementTR=document.getElementById("ligne1");
 premierTD=document.getElementById("celluleA1");
 elementTR.insertBefore(elementTD,premierTD);
}
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
 <tr id="ligne1">
  <td id="celluleA1">A1</td>
  <td id="celluleA2">A2</td>
  <td id="celluleA3">A3</td>
 </tr>
</table>
<input type="button" value="AJOUTER UNE CELLULE" onclick="test();">
```

replaceChild(nouveauNoeud,noeud) : remplacement d'un nœud par un autre nœud

Contraintes d'usage :

Classe : attribut de la classe Node (applicable à tous les nœuds).

Document : XML ou HTML.

La méthode `replaceChild(nouveauNoeud,noeud)` permet de remplacer le nœud passé dans le second paramètre (`noeud`) par un autre nœud, qui lui, sera passé dans le premier paramètre de la méthode (`nouveauNoeud`).

Syntaxe :

```
noeudPere.replaceChild(nouveauNoeud,noeud) ;
```

Pour la démonstration, nous allons reprendre la base de l'exemple précédent mais nous allons cette fois remplacer la première cellule A1 du tableau par le nouveau nœud de la cellule A4.

Ce traitement étant déclenché par l'appui sur un bouton, vous pourrez constater son action lors de vos tests par le remplacement de la cellule A1 par celle nouvellement créée et nommée A4 (voir fichier `exempleReplaceChild.html`).

Code JavaScript 20-25 :

```
function test(){
 //création du noeud élément
 elementTD=document.createElement("TD");
 elementTD.id="celluleA4";
 //création du noeud texte
 noeudTexte=document.createTextNode("A4");
 //attachement du noeud texte au noeud élément
 elementTD.appendChild(noeudTexte);
 //attachement du noeud élément à la suite des autres cellules
 elementTR=document.getElementById("ligne1");
 premierTD=document.getElementById("celluleA1");
 elementTR.replaceChild(elementTD,premierTD);
}
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
 <tr id="ligne1">
  <td id="celluleA1">A1</td>
  <td id="celluleA2">A2</td>
  <td id="celluleA3">A3</td>
 </tr>
</table>
<input type="button" value="REMPLACE UNE CELLULE" onclick="test();">
```

removeChild(noeud) : suppression d'un nœud

Contraintes d'usage :
Classe : attribut de la classe Node (applicable à tous les nœuds).
Document : XML ou HTML.

La méthode `removeChild(noeud)` permet de supprimer le nœud passé en paramètre qui doit être un nœud enfant du nœud auquel est appliquée la méthode.

Syntaxe :

```
noeudPere.insertBefore(noeudEnfantAsupprimer) ;
```

Pour la démonstration, nous allons reprendre la base de l'exemple précédent mais nous allons cette fois supprimer la première cellule A1 du tableau.

Ce traitement étant déclenché par l'appui sur un bouton, vous pourrez constater son action lors de vos tests par la suppression de la cellule A1 (voir fichier `exempleRemoveChild.html`).

Code JavaScript 20-26 :

```
function test(){
 //suppression du noeud élément de la cellule A1
 elementTR=document.getElementById("ligne1");
 premierTD=document.getElementById("celluleA1");
 elementTR.removeChild(premierTD);
}
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
  <tr id="ligne1">
   <td id="celluleA1">A1</td>
   <td id="celluleA2">A2</td>
   <td id="celluleA3">A3</td>
  </tr>
</table>
<input type="button" value="SUPPRIME UNE CELLULE" onclick="test();">
```

Si nous désirons supprimer tous les nœuds enfants d'un élément spécifique, il faudra alors utiliser une boucle `while` de manière à appliquer la méthode `removeChild()` tant qu'il existera des nœuds enfants. Dans l'exemple ci-dessous, nous avons réalisé une fonction `supprimerContenu()` qui supprime tout le contenu d'un élément. Cette fonction pourra être réutilisée par la suite dans vos futurs développements.

Code JavaScript 20-27 :

```
function supprimerContenu(element) {
  if (element != null) {
    while(element.firstChild)
      element.removeChild(element.firstChild);
  }
}
```

cloneChild(option) : clonage d'un nœud

Contraintes d'usage :

Classe : attribut de la classe Node (applicable à tous les nœuds).

Document : XML ou HTML.

La méthode `cloneChild(noeud)` permet de cloner le nœud auquel est appliquée la méthode. Le paramètre permet de définir s'il faut inclure les nœuds enfant (option = `true`) ou non (option = `false`) dans le nœud ainsi cloné.

Syntaxe :

```
noeudAcloner.cloneChild(true ou false) ;
```

Pour la démonstration, nous allons reprendre la base de l'exemple précédent mais nous allons cette fois dupliquer la cellule A3 puis la renommer en A4 avant de l'ajouter à la suite des cellules actuelles du tableau.

Ce traitement étant déclenché par l'appui sur un bouton, vous pourrez constater son action lors de vos tests par l'ajout de la cellule A4 à la suite de la cellule A3 (voir fichier `exempleCloneChild.html`).

Code JavaScript 20-28 :

```
function test(){
 dernierTD=document.getElementById("celluleA3");
 //clonage du noeud élément de la cellule A3
 clonageTD=dernierTD.cloneNode(true);
 //configuration de la cellule clonée
 clonageTD.firstChild.nodeValue="A4";
```

```
 clonageTD.firstChild.id="celluleA4";
 //ajout de la cellule clonée à la suite de A3
 elementTR=document.getElementById("ligne1");
 elementTR.appendChild(clonageTD);
}
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
  <tr id="ligne1">
   <td id="celluleA1">A1</td>
   <td id="celluleA2">A2</td>
   <td id="celluleA3">A3</td>
  </tr>
</table>
<input type="button" value="CLONER UNE CELLULE" onclick="test();">
```

style : modifier le style d'un nœud Element

Contraintes d'usage :

Classe : attribut de la classe Element (applicable aux nœuds élément exclusivement).

Document : HTML.

L'attribut `style` permet de lire, de modifier ou de supprimer le style du nœud.

Pour manipuler le style d'un nœud il suffit d'appliquer l'attribut `style` à l'objet nœud suivi de la propriété du style désiré en utilisant la syntaxe pointée ci-dessous (`monNoeud` étant l'objet nœud courant).

Syntaxe :

```
monNoeud.style.nomDuStyle ;
```

Les noms des styles pouvant être gérés avec le DOM sont semblables à ceux que nous avons coutume d'utiliser dans les feuilles de styles CSS. Toutefois, si le style CSS comporte deux mots séparés par un tiret, il faut alors supprimer le tiret et fusionner les deux mots en mettant en majuscule la première lettre du second mot. À titre d'exemple, nous indiquons dans le tableau 20-5 quelques styles CSS courants et leur équivalent DOM.

Tableau 20-5 Quelques exemples de styles CSS et leur équivalent DOM

Style CSS	Style DOM equivalent
border-color	borderColor
background-color	backgroundColor
color	color
font-family	fontFamily
font-size	fontSize
height	height
text-align	textAlign
width	width

Dans l'exemple ci-dessous, nous allons modifier le style de la ligne TR du tableau en lui appliquant une couleur de fond rouge.

Code JavaScript 20-29 :

```
function test(){
  elementTR=document.getElementById("ligne1");
  elementTR.style.backgroundColor = "red";
}
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
  <tr id="ligne1">
   <td id="celluleA1">A1</td>
   <td id="celluleA2">A2</td>
   <td id="celluleA3">A3</td>
  </tr>
</table>
<input type="button" value="CHANGER LE STYLE DE TR" onclick="test();">
```

À noter que pour modifier plusieurs styles d'un même élément, il devient avantageux d'utiliser la structure `with(element)` qui vous évitera de répéter le nom de l'élément devant chaque déclaration de style (voir le code 20-30).

Code JavaScript 20-30 :

```
function test(){
  elementTR=document.getElementById("ligne1");
  with(elementTR) {
    style.backgroundColor = "red";
    style.color = "blue";
    style.fontSize = "30px";
  }
}
```

Enfin, si vous désirez supprimer un style afin que l'élément reprenne son état initial, il suffit d'effacer les précédentes valeurs affectées comme l'illustre le code 20-31.

Code JavaScript 20-31 :

```
function test(){
  elementTR=document.getElementById("ligne1");
  with(elementTR) {
    style.backgroundColor = "";
    style.color = "";
    style.fontSize = "";
  }
}
```

innerHTML : lecture ou écriture du contenu d'un élément

Contraintes d'usage :

Classe : attribut de la classe Element (applicable au nœud élément exclusivement).

Document : HTML.

L'attribut `innerHTML` permet de remplacer le contenu d'un élément d'une manière très simple. Il permet aussi de lire le contenu d'un élément très facilement même s'il contient

d'autres balises. Cependant, cet attribut, initialement développé par Microsoft, ne fait pas partie des spécifications du W3C DOM et ses implémentations peuvent être assez différentes d'un navigateur à l'autre. Il est donc conseillé, dans la mesure du possible, d'utiliser des techniques alternatives à l'usage de l'attribut `innerHTML` utilisant les méthodes standards du DOM, même si leur utilisation est beaucoup moins simple.

Pour vous faciliter la tâche, nous vous proposons dans le code 20-33 deux fonctions (réalisées avec les méthodes DOM présentées dans cette partie) qui vous permettront de rester conforme aux spécifications tout en étant presque aussi simple d'utilisation que l'attribut `innerHTML`.

Syntaxe :

```
noeudElement.innerHTML ;
```

Pour démontrer l'utilisation de cet attribut, nous allons mettre en place deux fonctions. La première (`test1()`) utilise `innerHTML` en écriture afin de remplacer le texte de la cellule A1 par un nouveau texte. La seconde (`test2()`) utilise `innerHTML` pour lire le contenu de la cellule A2 et pour l'afficher ensuite dans une fenêtre d'alerte (voir fichier `exempleInnerHTML1.html`).

Code JavaScript 20-32 :

```
function test1(){
  var elementA2=document.getElementById("celluleA1");
  //Ecriture avec innerHTML
  var elementA2.innerHTML="Nouveau contenu de A1";
}
function test2(){
  var elementA2=document.getElementById("celluleA2");
  //Lecture avec innerHTML
  var contenuA2=elementA2.innerHTML;
  alert("le contenu de la cellule 2 est : "+contenuA2);
}
```

Balises de la page HTML :

```
<table width="400" border="1" cellspacing="0">
 <tr id="ligne1">
  <td id="celluleA1">A1</td>
  <td id="celluleA2">A2</td>
  <td id="celluleA3">A3</td>
 </tr>
</table>
<input type="button" value="ECRITURE DANS CONTENU DE LA CELLULE A1" onclick
➥="test1();">
<input type="button" value="LECTURE DU CONTENU DE LA CELLULE A2" onclick="test2();">
```

Pour illustrer les solutions alternatives à `innerHTML`, nous proposons maintenant de réaliser les deux mêmes fonctions que précédemment mais avec des fonctions DOM standards (voir fichier `exempleInnerHTML2.html`).

Par la suite les deux fonctions `remplacerContenu()` et `supprimerContenu()` pourront être avantageusement placées dans un fichiers `JS` externe afin d'éviter de surcharger inutilement le code de la page (voir pour exemple les ateliers du chapitre 10).

Code JavaScript 20-33 :

```
function test1(){
  remplacerContenu("celluleA1","Nouveau contenu de A1");
}
function test2(){
  var elementA2=document.getElementById("celluleA2");
  var contenuA2=elementA2.firstChild.nodeValue;
  alert("le contenu de la cellule 2 est : "+contenuA2);
}
//fonctions DOM en alternative à innerHTML
function remplacerContenu(id, texte) {
  var element = document.getElementById(id);
  if (element != null) {
    supprimerContenu(element);
    var nouveauContenu = document.createTextNode(texte);
    element.appendChild(nouveauContenu);
  }
}
function supprimerContenu(element) {
  if (element != null) {
   while(element.firstChild)
     element.removeChild(element.firstChild);
  }
}
```

Gérer les événements avec DOM Events

Jusqu'à présent, nous avons utilisé les méthodes et attributs des parties DOM Core (XML), DOM HTML et DOM Style pour connaître les informations d'un nœud, accéder à un nœud particulier de l'arbre ou encore se déplacer dans l'arbre. Nous vous proposons maintenant d'exploiter les propriétés de DOM Events pour contrôler les événements du navigateur, du clavier et de la souris.

Les événements et leur gestionnaire

Le principe d'une application interactive est de réagir à un événement pour changer le contenu ou la forme de la page Web affichée dans le navigateur. Il existe de nombreux événements ; certains seront initiés par l'utilisateur par le biais d'une action sur le clavier (`keypress` ou `keyup` par exemple) ou à l'aide de la souris (`mouseover` ou `mouseout` par exemple) alors que d'autres peuvent être déclenchés par le navigateur lui-même pour signaler qu'une page est complètement chargée, par exemple (`load`).

Quelle que soit l'origine de l'événement, le processus de son traitement est semblable. Dès que l'événement survient, il est détecté et un programme développé préalablement pour le traiter est exécuté.

Le système qui détecte un événement se nomme un gestionnaire d'événement. Avec DOM Events, le gestionnaire d'événement est matérialisé par une propriété de l'objet qu'il doit contrôler et dont l'appellation est fabriquée par la concaténation du nom de l'événement et du préfixe "on" (voir le tableau 20-6). Par exemple, le gestionnaire de l'événement `click` (correspondant à un clic de souris) se nommera `onclick`.

Tableau 20-6 Correspondances entre quelques événements et leurs gestionnaires

Evénéments	Gestionnaires d'événement (propriétés du nœud)	Type de nœud contrôlable	Actions détectées
`blur`	`onblur`	`Element`	Perte de focus de l'élément
change	onchange	`Element`	Modification de la valeur de l'élément
click	onclick	`Element`	Clic de souris sur l'élément
dblclick	ondblclick	`Element`	Double clic de souris sur l'élément
focus	onfocus	`Element`	Obtention du focus de l'élément
keydown	onkeydown	`Element`	Touche enfoncée
keypress	onkeypress	`Element`	Touche pressée
keyup	onkeyup	`Element`	Touche relâchée
load	onload	`Document`, `Element`	Chargement de la page
mousedown	onmousedown	`Element`	Bouton de la souris enfoncé
mousemove	onmousemove	`Element`	Déplacement de la souris
mouseout	onmouseout	`Element`	La pointeur de la souris est sorti de l'élément
mouseover	onmouseover	`Element`	Passage du pointeur de la souris au dessus de l'élément
mouseup	onmouseup	`Element`	Relâchement du bouton de la souris
resize	onresize	`Document`, `Element`	Redimensionnement de la page ou de l'élément
scroll	onscroll	`Document`, `Element`	Utilisation du défilement de la page
unload	onunload	`Document`, `Element`	Lorsqu'on quitte la page

L'objet que doit contrôler le gestionnaire d'événement peut être un élément HTML (`div` ou `input`…) s'il s'agit de détecter un clic de souris sur une zone ou encore l'élément racine de la page (Document), ou s'il s'agit de détecter le chargement de la page dans le navigateur par exemple (voir le tableau 20-6).

Une fois l'objet défini, le gestionnaire d'événement lui sera appliqué en utilisant la syntaxe pointée (`objet.onclick` avec `objet` correspondant à une référence d'un élément `div` de la page par exemple). Le but de cette propriété, correspondant au gestionnaire d'événement concerné, est de mémoriser le nom de la fonction qui prendra en charge le traitement lorsque l'événement surviendra. Pour cela, il suffit simplement de lui affecter le nom de la fonction (attention, le nom de la fonction ne doit pas avoir de parenthèses sinon c'est le résultat de la fonction qui serait affecté et non son nom) comme dans l'exemple du code 20-34.

Il ne reste plus ensuite qu'à déclarer la fonction de traitement dont nous venons d'enregistrer le nom dans la propriété de l'objet pour que le gestionnaire soit opérationnel.

Syntaxe :

```
noeud.propriete = fonction ;
```

avec :

`noeud` pouvant être le document ou une référence à un élément du document (à l'aide de `getElementById()` par exemple) selon le type d'événement (voir tableau 20-6).

`propriete` est une des propriétés DOM Events de l'objet `noeud` (soit l'un des gestionnaires d'événements du tableau 20-6).

`fonction` est le nom de la fonction appelée pour traiter l'événement et qui a été créée préalablement par le développeur.

Code 20-34 : voir fichier `gestionnaire1.html` :

```
<div id="zoneA" >Cliquez ici </div>
…
<script type="text/javascript">
function afficheClic(){
  alert("CLIC");
}
objetCliquable=document.getElementById("zoneA");
objetCliquable.onclick=afficheClic;
</script>
```

Dans l'exemple ci-dessus, la page contient une zone `div` dont l'identifiant est `"zoneA"` et dans laquelle on a placé le texte `"cliquez ici"`. Dans un script placé après cette zone, nous avons déclaré la fonction qui prendra en charge le traitement de l'événement (`afficheClic()`) puis nous avons configuré le gestionnaire d'événement en le liant avec l'objet à contrôler et en lui affectant le nom de la fonction de traitement. Si nous appelons cette page et que nous cliquons sur le texte de la balise `div`, une boîte d'alerte doit alors apparaître contenant le texte `"CLIC"` (voir figure 20-6).

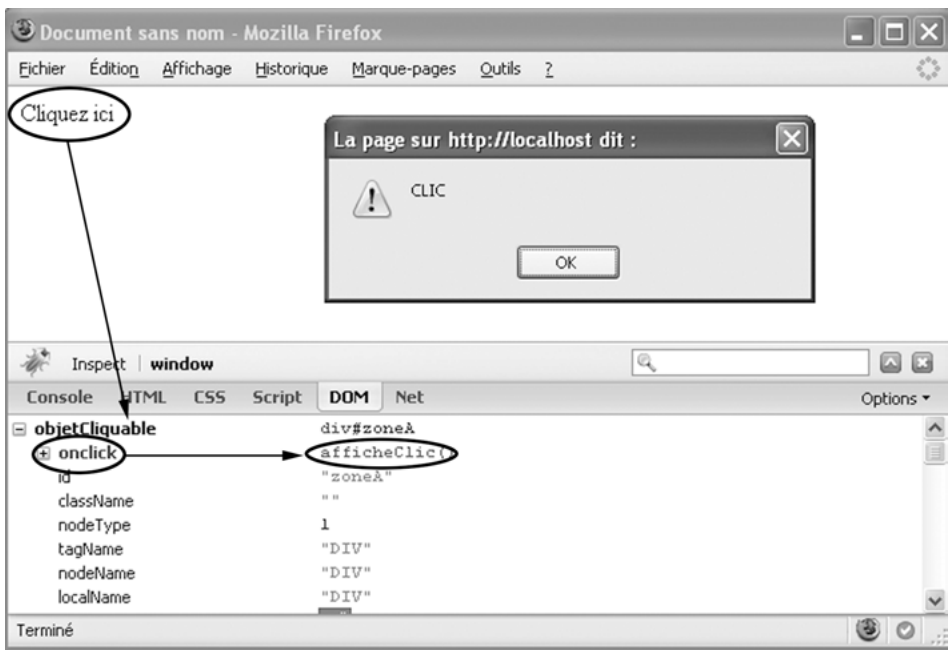


Figure 20-6

Test du gestionnaire d'événement du code 20-34

Si nous observons le nœud de l'objet `objetCliquable` dans l'arbre DOM avec Firebug, nous pouvons constater que la propriété `onclick` (aussi appelée gestionnaire d'événement) de l'objet contient la fonction de traitement `afficheClic()` que nous lui avons affecté dans le code 20-34 (voir figure 20-6).

Ancienne façon de configurer un gestionnaire d'événement

Avant l'apparition du DOM, on avait coutume d'utiliser des gestionnaires d'événements insérés directement dans les balises de la page HTML (voir exemples du code 20-35). Désormais, avec les événements DOM Events, il convient d'utiliser des gestionnaires d'événements qui ne seront pas déclarés dans les balises HTML mais intégrés dans le code JavaScript (voir code 20-34). Cette nouvelle technique permet d'améliorer la lisibilité et la maintenance de l'application mais elle permet surtout de séparer parfaitement la structure et le contenu HTML du code JavaScript.

Code 20-35 : voir fichier `gestionnaire0.html` :

```
<head>
<script type="text/javascript">
function afficheClic(){
  alert("CLIC");
}
</script>
</head>
<body>
<div onclick="afficheClic()">Cliquez ici </div>
</body>
```

L'objet Event et ses propriétés

Lorsqu'un événement se produit, un objet `Event` est créé. Cet objet dispose de plusieurs propriétés qui permettent de connaître des informations complémentaire à l'événement.

Ainsi, si vous utilisez un gestionnaire d'événement lié à la souris (`onmouseover`, `onmousedown`…) vous pourrez connaître la position du curseur au moment de l'événement, ou encore le bouton qui a été utilisé. De même que pour les événements du clavier (`keydown`, `keyup`…), il vous sera possible d'identifier la touche qui a été pressée ou si des touches sélecteur (`Alt`, `Ctrl` ou `Maj`) ont été actionnées en même temps (voir les exemples de propriétés de l'objet `Event` du tableau 20-7).

Tableau 20-7 Exemples de quelques propriétés de l'objet Event

Propriétés de l'objet Event	Description
`altKey`	Renvoie `true` si la touche Alt a été actionnée
`ctrlKey`	Renvoie `true` si la touche Ctrl a été actionnée
`shiftKey`	Renvoie `true` si la touche Shift a été actionnée
`keyCode`	Renvoie le code Unicode de la touche actionnée
`clientX`	Coordonnée X de la souris
`clientY`	Coordonnée Y de la souris
`screenX`	Coordonnée X de l'événement
`screenY`	Coordonnée Y de l'événement

Récupération de Event dans la fonction de traitement

Pour exploiter les propriétés de l'objet `Event`, il faut en disposer dans la fonction de traitement de l'événement. Pour cela, la méthode de récupération sera différente selon le navigateur utilisé.

Pour les navigateurs conformes aux spécifications du W3C, comme Firefox, la récupération de l'objet est automatique. En effet, lors de l'événement, le gestionnaire transmet l'objet `event` en paramètre dans la fonction de manière implicite sans arguments à configurer dans l'appel de la fonction. Ainsi, dans le code ci-dessous (code 20-36) l'objet `event` et ses propriétés sont disponibles et pourront être utilisés dans le traitement de l'événement.

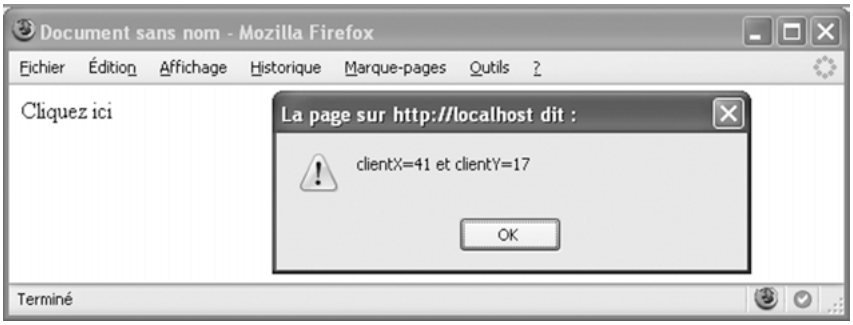
Code 20-36 : voir fichier `exempleEvent1.html` :

```
<div id="zoneA" >Cliquez ici </div>

<script type="text/javascript">
 function afficheClic(event){//valable uniquement avec nav. W3C
  alert("clientX="+event.clientX+" et clientY="+event.clientY);
 }
 objetCliquable=document.getElementById("zoneA");
 objetCliquable.onclick=afficheClic;
</script>
```

Figure 20-7

Récupération de l'objet Event et de ses propriétés



Si nous testons la page du code 20-36 dans Firefox, une fenêtre d'alerte doit s'ouvrir lorsque l'on clique sur le texte `"cliquez ici"` et afficher les coordonnées X et Y de la souris au moment de l'action (voir figure 20-7). Par contre, si nous basculons dans IE (en cliquant sur le bouton de l'extension `IE Tab` par exemple) et que nous réalisons le même test, nous constatons que le système ne fonctionne plus.

En effet, dans le cas d'Internet Explorer l'objet n'est pas passé automatiquement en paramètre à la fonction et il faut utiliser l'objet `window.event` directement dans le corps de la fonction pour disposer de ses propriétés comme l'illustre le code 20-37.

Code 20-37 : voir fichier `exempleEvent2.html` :

```
<div id="zoneA" >Cliquez ici </div>

<script type="text/javascript">
 function afficheClic(){ //valable uniquement avec IE
  alert("clientX="+window.event.clientX+" et clientY="+window.event.clientY);
}
```

```
 objetCliquable=document.getElementById("zoneA");
 objetCliquable.onclick=afficheClic;
</script>
```

Si nous testons ce code 20-37 dans IE, nous constatons que cela fonctionne de la même manière qu'avec Firefox précédemment (revoir figure 20-7) mais par contre si nous testons de nouveau avec le navigateur Firefox, le système ne fonctionne plus. Il faut donc trouver une solution pour que nous puissions disposer des propriétés de l'objet `Event` avec les deux navigateurs.

La solution consiste à utiliser l'opérateur OU (symbole `||`) entre l'objet `window.event` de IE et l'objet `event` récupéré dans le paramètre de la fonction avec les navigateurs conformes au W3C (comme Firefox). Dans le cas d'une utilisation de Firefox, le résultat du test sera `event` car `window.event` n'est pas défini dans Firefox et, à l'inverse, avec IE le résultat sera `window.event`. Il suffit ensuite d'affecter ce résultat à une variable objet (nommée aussi `event` dans notre exemple) qui sera utilisée dans la suite du traitement pour obtenir un système compatible avec les deux navigateurs (voir code 20-38).

Code 20-38 : voir fichier `exempleEvent2.html` :

```
<div id="zoneA" >Cliquez ici </div>

<script type="text/javascript">
function afficheClic(event){ //valable avec FF et IE
 event=window.event||event;//test des navigateurs
 alert("clientX="+event.clientX+" et clientY="+event.clientY);
}
 objetCliquable=document.getElementById("zoneA");
 objetCliquable.onclick=afficheClic;
</script>
```

21

PHP

PHP est un langage de script exécuté coté serveur (contrairement à JavaScript qui est exécuté côté client). À la fois simple, performant et disposant d'une grande diversité de bibliothèques, il permet aussi de se connecter à une base de données (comme MySQL).

Avertissement

L'objectif de ce chapitre est de vous donner les bases du langage PHP afin que vous puissiez comprendre les instructions utilisées dans les différentes ateliers. Il n'a donc pas la prétention d'être un cours exhaustif sur ce langage.

La syntaxe de PHP

Extension de fichier PHP

Le code PHP doit être interprété avant d'être envoyé vers le navigateur du client. Pour cela, le code source d'un script est toujours enregistré dans un fichier portant l'extension `.php`, pour que l'interpréteur PHP l'identifie et l'analyse (exemple : `mapage.php`).

Balises de code PHP

Le script intégré dans la page doit en outre être encadré par deux balises `<?php` et `?>`, afin que l'interpréteur PHP puisse évaluer le code ainsi délimité.

Lorsque l'interpréteur PHP détecte la balise de début (`<?php`), il traite le code qui suit comme des instructions PHP et les interprète jusqu'à ce que la balise de fin (`?>`) soit rencontrée. Après celle-ci, il considère que le code est en HTML et l'envoie tel quel au navigateur du destinataire.

Code 21-1 : exemple de script PHP :

```
<?php
echo "Bonjour";
?>
```

Dans l'exemple ci-dessus, la première ligne du programme est composée de la fonction `echo` qui affiche `Bonjour` à l'écran. La page PHP peut comprendre uniquement ce script ou du code HTML dans lequel on a intégré le script PHP. Dans ce cas, seul le script PHP délimité par ses balises est interprété par l'interpréteur PHP ; le code HTML est quant à lui retranscrit à l'identique dans la page finale envoyée au navigateur.

La fonction d'affichage echo

Nous abordons l'étude des fonctions intégrées PHP un peu plus loin dans ce même chapitre. Cependant, nous allons utiliser la fonction `echo` dès maintenant, afin d'afficher facilement les valeurs des variables ou d'émuler du code HTML dans un script PHP. Voici quelques informations sur l'utilisation de cette première fonction. Sachez pour commencer que la fonction `echo` est la seule fonction qui ne nécessite pas l'usage de parenthèses pour encadrer ses arguments. Elle permet d'afficher des textes ou des balises HTML s'ils sont encadrés par des guillemets simples ou doubles. Par exemple, `echo "bonjour";` ou `echo 'bonjour';` affichent le mot « `bonjour` » dans la page Web. `echo "<br>";` émule la balise HTML `<br>` et provoque un retour à la ligne. La commande `echo` permet également d'afficher la valeur d'une variable si elle est encadrée par des doubles guillemets ou ne comporte pas de guillemets. Par exemple, `echo "$var1";` ou `echo $var1;` affichent la valeur de la variable `$var1` dans la page Web. Attention ! Si vous passez dans l'argument le nom d'une variable encadrée par des guillemets simples, vous affichez son nom et non sa valeur.

Les commentaires

Dans un script PHP, il est possible de commenter le code PHP en utilisant deux syntaxes différentes selon le type de commentaire.

Commentaires de simple ligne //

Si on désire insérer un commentaire sur une ligne ou à la fin d'une instruction, il faut saisir deux barres obliques `//` devant celle-ci.

Code 21-2 : commentaires d'une simple ligne :

```
echo "Bonjour"; //Ici c'est un commentaire en bout de ligne
// Ici c'est un commentaire sur une ligne complète
```

Commentaires multilignes /* et */

Si on désire insérer plusieurs lignes de commentaire, il faut employer le signe `/*` au début de la zone de commentaire et `*/` à la fin de celle-ci.

Code 21-3 : commentaire multilignes :

```
/*
ceci est un commentaire
sur plusieurs
lignes
*/
```

Utiliser les commentaires pour déboguer

Dans le cadre du dépannage d'un script PHP, vous pouvez utiliser les commentaires pour neutraliser une ligne ou un bloc de code. Cela permet de tester la page sans interpréter la partie neutralisée et d'identifier le script qui produit l'erreur. Pour plus d'information sur le débogage d'un programme, reportez-vous au chapitre dédié à la mise en œuvre des programmes.

Les variables

La variable et sa valeur

Les variables sont des symboles auxquels on affecte des valeurs. Après leur affectation, vous pouvez modifier les variables à tout moment au cours du déroulement du programme. La déclaration d'une variable n'est pas obligatoire en PHP car la variable peut être créée lors de sa première affectation. De même, les variables prennent le type correspondant à la valeur qui leur est affectée.

Noms des variables

Les noms des variables sont toujours précédés du caractère `$` et suivis du nom choisi pour identifier la variable, qui ne doit comporter que des caractères alphanumériques (sauf le premier caractère, qui ne doit pas être un chiffre) ou le caractère souligné _. En pratique, il est judicieux de choisir un nom explicite et de se fixer une règle de nommage.

Types des variables

Les variables peuvent être de plusieurs types.

Tableau 21-1 Types de variable PHP

Type de variable	Description	Exemples

Chaîne de caractères `(string)`	Leurs valeurs sont des lettres, chiffres ou symboles. Pour affecter une valeur alphanumérique à une variable, vous devez l'encadrer par des guillemets.
	À noter : les guillemets peuvent être doubles (") ou simples (’). Une chaîne encadrée par des guillemets simples peut contenir des guillemets doubles et vice versa. Si vous devez intégrer des guillemets de même type dans la chaîne, il faut les faire précéder du caractère d'échappement (\).
		`$var1="Dupond";`
		`$var2="bonjour M. $var1";`
		`$var3="254";`
		`$var4="total=150";`
		`$var5= "d'en face" ;`
		`$var5= 'd\'en face';`
		`$var6= 'height="20" ' ;`
		`$var7= "height=\"20\" " ;`

Numériques entiers `(integer)`	Leurs valeurs sont uniquement des nombres entiers. Pour affecter un entier à une variable, il ne doit pas être encadré par des guillemets.	`$var1=152;`
		`$var2=5;`
		`$var3=45+$var1;`

Numériques décimales `(double)`	Leurs valeurs sont uniquement des nombres décimaux. Pour affecter un décimal à une variable, il ne doit pas être encadré par des guillemets.
	À noter : le séparateur des valeurs décimales utilisé en PHP est le point (`.`) et non la virgule (`,`).
		`$var1=152.20;`
		`$var2=124.50;`
		`$var3=45.85+$var1;`

Booléens `(boolean)`	Leurs valeurs sont soit `true` (vrai), soit `false` (faux). Ces valeurs sont affectées à la variable en utilisant une expression de condition (exemple : `$var1==$var2`).
	La valeur `false` peut être le 0, une chaîne vide `""` ou le caractère `"0"` et la valeur `true` toutes les autres valeurs.
		`$var1=5 ; //var num`
		`$var2=2 ; //var num`
		`$var3=($var1==$var2);`
		`/* $var3 est une variable booléenne et sa valeur est FALSE dans ce cas */`

Tableaux `(array)`	Un tableau est une série de valeurs ayant en commun le même nom de variable. Il peut être indicé ou associatif. Le premier indice d'un tableau est toujours zéro.	`$mois[0]="janvier";`
		`$mois[1]="février";`
		`$mois[2]="mars";`
		…

Objets `(object)`	Les objets (ou classes) sont des ensembles de variables et de fonctions définies par l'utilisateur.	`class chrono {`
		`var $var1=2;`
		`function debut()`
		`{this->$var1=time()}`
		`}`

Comment connaître le type d'une variable ?

En cours de développement, vous aurez peut-être besoin de connaître le type d'une variable avant de l'exploiter. Dans ce cas, il suffit d'utiliser la fonction `gettype($var1)`, qui retourne le type de la variable `$var1`. En phase de mise au point d'un programme, vous pouvez intégrer provisoirement dans votre page l'instruction suivante : `echo gettype($var1);`, qui affiche le type de la variable directement dans la page Web (`string`, `integer`, `double`, `boolean`, `array`, `object`).

Variables simples

La façon la plus simple de manipuler une variable est d'utiliser son nom précédé d'un caractère `$` (exemple : `$var1`). Cette syntaxe est utilisée aussi bien pour son affectation que lors de son utilisation au sein d'une page Web. Dans l'exemple ci-dessous, le type de la variable `$var1` est numérique comme le type de la valeur qui lui est affectée.

Code 21-4 : affectation d'une variable :

```
//affectation d'une variable simple
$var1=100;
//utilisation d'une variable
echo $var1; // ici la valeur de la variable sera affichée dans la page
```

Comment tester si une variable existe ?

Pour tester si une variable a déjà été affectée, on peut utiliser la fonction `isset($var1)`, qui retourne `true` (vrai) si la variable existe (si elle est déjà affectée, même avec une chaîne vide ou un 0) et `false` dans les autres cas (si elle n'existe pas encore ou si elle n'a pas été affectée). À l'inverse, la fonction `empty($var1)` permet de tester si une variable est vide (une variable vide vaut 0 dans le cas d'une variable numérique ou est une chaîne vide dans le cas d'une chaîne de caractères). Si elle est vide, la fonction retourne `true` ; elle retourne `false` dans tous les autres cas. Attention, si la variable testée contient une chaîne vide ou un 0, les deux fonctions retournent `true`.

À titre d'exemple, voici un cas pratique : afin d'éviter de multiples messages d'erreur du genre `Undefined variable` lors de l'affichage de la page, vous serez certainement amené à vérifier si vos variables sont bien initialisées. Le code suivant (placez-le en début de page) initialise la variable en question uniquement si elle n'existe pas encore (l'instruction `if()` utilisée dans ce test est présentée dans le chapitre).

Code 21-5 : test d'existence d'une variable :

```
if(!isset($variable)) $variable=" ";
```

Variables en tableaux indicés

Les tableaux sont des séries de valeurs regroupées sous le même identifiant. On distingue chaque élément de la série par un indice entre crochets (exemple : `$tab1[0]`). Les tableaux qui utilisent des indices numériques pour distinguer leurs éléments s'appellent des tableaux indicés. Les indices des tableaux commencent toujours à `0`.

Code 21-6 : affichage du contenu d'un tableau :

```
echo "<pre>";
print_r($agence);
echo "</pre>";
```

Le contenu du tableau est alors affiché avec la mise en forme suivante :

```
Array
(
 [0] => "Paris"
 [1] => "Lille"
 [2] => "Marseille"
)
```

Comment connaître les valeurs contenues dans un tableau ?

En cours de développement, vous aurez peut-être besoin d'afficher rapidement le contenu d'un tableau afin de tester votre script. Dans ce cas, il suffit d'utiliser la fonction `print_r($tab1)`, qui affiche directement à l'écran les différentes valeurs du tableau `$tab1`. Par exemple, si vous désirez connaître les valeurs du tableau `$agence` après son affectation (voir le premier exemple de tableau ci-dessous), vous pouvez intégrer provisoirement dans votre page l'instruction suivante :

```
print_r($agence)
```

qui affiche les informations suivantes dans la page Web :

```
Array([0]=>Paris [1]=>Lille [2]=>Marseille)
```

Si vous désirez afficher des tableaux plus importants, il est préférable de formater la mise en forme des résultats à l'aide de la balise HTML `<pre>` comme dans l'exemple du code 21-6.

Il existe plusieurs manières d'affecter des valeurs à un tableau. Chaque manière a ses avantages et ses inconvénients et il faut donc choisir la méthode d'affectation selon le contexte du programme.

La première consiste à affecter sa valeur à chaque élément en précisant de manière explicite l'indice de l'élément. Cette méthode est bien adaptée à l'affectation isolée d'une variable d'un tableau. L'exemple ci-dessous initialise un tableau qui mémorise des noms d'agences.

Code 21-7 : affectation individuelle de valeurs à un tableau indicé :

```
$agence[0]="Paris";
$agence[1]="Lille";
$agence[2]="Marseille";
```

La deuxième consiste à ne pas préciser l'indice de l'élément du tableau lors de son affectation. Dans ce cas, le premier élément affecté est d'indice `0` et les autres sont incrémentés au fur et à mesure des affectations. Cette méthode est bien adaptée à l'affection d'une série de variables d'un tableau intégrée dans une structure de boucle. L'exemple ci-dessous réalise exactement la même affectation que l'exemple précédent utilisant des indices.

Code 21-8 : affectation auto-incrémentée de valeurs à un tableau indicé :

```
$agence[]="Paris";
$agence[]="Lille";
$agence[]="Marseille";
```

La troisième manière consiste à utiliser la fonction `array()`. Il convient alors de préciser les différentes valeurs du tableau dans les arguments de la fonction (séparés par des virgules). Cette dernière méthode est bien adaptée à l'affection initiale et complète d'un tableau de variables. L'exemple ci-dessous réalise exactement la même affectation que l'exemple précédent utilisant des indices.

Code 21-9 : affectation groupée de valeurs à un tableau indicé lors de la déclaration :

```
//affectation des valeurs
$agence=array(”Paris”,”Lille”,”Marseille”);
//utilisation des variables
echo $agence[0];//affiche ”Paris”
```

Variables en tableaux associatifs

Il est également possible de remplacer les indices par un nom explicite (la clé). Dans ce cas, le tableau est dit associatif (exemple `$tab1[”nom”]`).

Nous avons décliné ci-dessous les exemples expliqués précédemment, mais en utilisant cette fois des tableaux associatifs.

Code 21-10 : affectation individuelle de valeurs à un tableau associatif :

```
$agence["centre"] = "Paris";
$agence["nord"] = "Lille";
$agence["sud"] = "Marseille";
```

ou bien :

Code 21-11 : affectation groupée de valeurs à un tableau associatif lors de la déclaration :

```
//affectation des valeurs
$agence=array(
"centre"=>"Paris",
"nord"=>"Lille",
"sud"=>"Marseille");
//utilisation des variables
echo $agence["centre"]; //affiche "Paris"
```

Variables en tableaux multidimensionnels

PHP ne gère pas les tableaux à deux dimensions. Il est toutefois possible de créer une telle matrice en déclarant une autre structure de tableau à la place des différentes variables du tableau principal. Le tableau principal se comporte alors comme un tableau à deux dimensions (voir le tableau 21-2 et l’exemple ci-dessous).

Tableau 21-2 Matrice équivalant à l’exemple ci-dessous (`$tableauprincipal`)

[x][y]	[y]=[0]	[y]=[1]
[x]=[0]	A1	A2
[x]=[1]	B1	B2

Code 21-12 : déclaration, affectation et utilisation de valeurs dans un tableau à deux dimensions :

```
//initialisation d’un tableau à 2 dimensions
$ligneA=array(”A1”,”A2”);
$ligneB=array(”B1”,”B2”);
$tableauprincipal=array ($ligneA,$ligneB);
//utilisation de ses éléments
echo $tableauprincipal[0][0]; //affiche A1
echo $tableauprincipal[0][1]; //affiche A2
echo $tableauprincipal[1][0]; //affiche B1
echo $tableauprincipal[1][1]; //affiche B2
```

Variables HTTP

Il existe plusieurs familles de variables HTTP (variables de formulaire, d'URL, de requête, de fichier, de session, de cookie et de serveur). Selon leur famille, les variables HTTP seront stockées dans des tableaux différents (`$_POST[]`, `$_GET[]`, `$_REQUEST[]`, `$_FILES[][]`, `$_SESSION[]`, `$_COOKIE[]`, `$_SERVER[]`).

Ainsi, pour récupérer une variable envoyée depuis un formulaire, il faut utiliser le tableau `$_POST[]` des variables HTTP. Par exemple, si le contenu d'un champ envoyé par un formulaire en méthode POST est nommé `nomVar` , on doit utiliser la variable HTTP `$_POST['nomVar']` pour récupérer sa valeur côté serveur.

De même les variables de fichier (lors du téléchargement d'un fichier depuis un formulaire en méthode POST) suivent les mêmes règles. Ainsi, si le champ fichier de votre formulaire se nomme `photo`, la variable HTTP `$_POST['photo']` contient un tableau comportant les différentes informations concernant le fichier téléchargé (voir le détail de ces information dans le tableau 21-3).

Tableau 21-3 Les variables HTTP de PHP

Famille de variable	Syntaxe du tableau de variables	Description
variable de formulaire utilisant la méthode POST.	`$_POST['var1']`	Tableau des variables de formulaire stockant les informations récupérées lors d'une requête d'un formulaire utilisant la méthode POST.
variable passée dans l'URL ou issue d'un formulaire utilisant la méthode GET.	`$_GET['var1']`	Tableau des variables d'URL qui stocke les valeurs transmises par l'URL (exemple : `bonjour.php?var1=100`) ou saisies par l'internaute dans un formulaire utilisant la méthode GET.
variable passée indifféremment avec la méthode GET ou avec la méthode POST.	`$_REQUEST['var1']`	Tableau regroupant les variables de formulaire et d'URL.
variable de session (voir le détail dans la partie ci-après consacrée aux sessions)	`$_SESSION['var1']`	Tableau des variables de session qui mémorise des informations enregistrées pendant toute la durée de la visite de l'internaute (la session).
variable de cookie	`$_COOKIE['var1']`	Tableau des variables de cookie qui permet de récupérer une information stockée au préalable sur le poste client.

variable de fichier
Lors du chargement d'un fichier depuis un formulaire en méthode POST, il est stocké dans un répertoire temporaire. Il conviendra donc de le copier ensuite dans le répertoire désiré (à l'aide de la fonction `copy` ou `move_uploaded_file` par exemple)
	`$_FILES['photo'] ['name']`
	correspond au nom du fichier
	`$_FILES['photo'] ['type']`
	correspond au type MIME du fichier
	`$ _FILES['photo'] ['size']`
	correspond à la taille en octets du fichier
	`$ _FILES['photo'] ['tmp_name']`
	correspond au nom temporaire du fichier
		Tableau des différentes informations correspondant à un fichier téléchargé par la méthode POST.
		Dans les exemples ci-contre, on suppose que le nom du champ du formulaire (de type fichier) est `photo`.
		Attention ! Dans ce cas, le formulaire dans lequel est intégré le champ de téléchargement de fichier (balise input de `type="file"`) doit être configuré avec l'attribut `enctype="multipart/form-data"` et en méthode POST.

Variables de session

Depuis PHP 4, la gestion des sessions est plus facile. Dès qu'une nouvelle session est demandée, un identifiant de session est automatiquement créé, auquel on associe les différentes variables mémorisées durant la session dans un tableau nommé `$_SESSION`.

Pour pouvoir stocker ou lire des variables dans la session de l'utilisateur, il faut que l'instruction d'initialisation de la session `session_start()` soit présente en début de la page (il faut placer impérativement cette fonction avant tout code HTML ou fonction `echo()`). L'ajout, la modification ou la lecture d'une variable de session peut ensuite être effectué.

Tableau 21-4 Fonctions de gestion des sessions

Fonction	Définition
`session_start()`	Initialise la session. Si la session n'existe pas encore, un identifiant de session est créé puis transmis dans un cookie. Si la session existe déjà, la fonction actualise toutes les variables mémorisées dans la session existante. Attention ! Vous devez impérativement appeler cette fonction avant toute interprétation de balise HTML.
`session_destroy()`	Détruit toutes les données associées à une session.
`session_id()`	Renvoie l'identifiant de la session en cours.
`$_SESSION['var1']="Bonjour"`	Mémorise la valeur `"Bonjour"` dans la variable `$var1` de la session en cours.

Voici un exemple d'enregistrement et d'utilisation d'une variable de session dans deux pages différentes pendant une même session :

```
/* à placer en haut d'une première page : enregistrement de la variable "$var1"
➥dans la session en cours */
session_start();
$_SESSION['var1'] = "bonjour";
/* à placer en haut d'une autre page : récupération de la variable "$var1"
➥depuis la session en cours */
session_start();
echo $_SESSION['var1']; //affiche la valeur récupérée
```

Variables de serveur

D'autres variables prédéfinies fournissent des renseignements précieux, comme les noms du navigateur client et du système d'exploitation ou encore l'adresse IP de l'internaute qui consulte votre site. Vous pouvez facilement exploiter ces variables à l'aide du tableau `$_SERVER['nom_var']`, qui regroupe toutes ces variables HTTP. La plupart de ces valeurs sont attribuées par le serveur : vous pouvez les exploiter dans de multiples applications, mais vous ne devez pas les modifier. Le tableau 21-5 propose une liste non exhaustive de ces variables.

Tableau 21-5 Liste non exhaustive de quelques variables de serveur prédéfinies disponibles sur le serveur Web

Nom de la variable	Définition
`$_SERVER['HTTP_USER_AGENT']`	Contient le nom et la version du navigateur utilisé par l'internaute.
`$_SERVER[HTTP_ACCEPT_LANGUAGE']`	Contient le code de la langue paramétrée dans le navigateur de l'internaute (par exemple `fr`, si le navigateur est en version française).
`$_SERVER['REMOTE_ADDR']`	Contient l'adresse `IP` de l'internaute qui consulte la page.

Tableau 21-5 Liste non exhaustive de quelques variables de serveur prédéfinies disponibles sur le serveur Web *(suite)*

Nom de la variable	Définition
`$_SERVER['DOCUMENT_ROOT']`	Contient le nom du répertoire de la page affichée.
`$_SERVER['QUERY_STRING']`	Contient les informations passées dans l'URL après le caractère ?
`$_SERVER['PHP_SELF']`	Contient le chemin et le nom de la page Web en cours de traitement.

Exemple :

```
echo $_SERVER['PHP_SELF'];
//affiche le chemin de la page en cours
echo $_SERVER['QUERY_STRING'];
//affiche les informations passées dans l'URL après le point d'interrogation
//si l'URL est page.php?var1=20, alors var1=20 est affiché à l'écran
```

Les constantes

Il est possible de définir des constantes à l'aide de la fonction `define()`. Contrairement aux variables, les valeurs affectées initialement à une constante ne peuvent plus être modifiées. Les deux principaux avantages des constantes sont les suivants : non seulement elles rendent le code plus explicite, mais leur valeur peut être modifiée en un seul point alors qu'elles sont accessibles globalement en tout endroit du code (contrairement aux variables, qui ont une durée de vie limitée à l'exécution du script de la page). Il est d'usage (mais ce n'est pas obligatoire) de nommer les constantes avec des majuscules. Vous pouvez accéder à une constante en indiquant son nom (attention à ne pas mettre de `$` devant ce nom).

Tableau 21-6 Définition d'une constante

Syntaxe
define (nom_constante, valeur)
Exemple : define("REP_IMAGES", "/monsite/images/");

À noter

Il existe des constantes prédéfinies par PHP comme `_FILE_`, qui contient le nom du fichier en cours d'utilisation, `_LINE_`, qui contient le numéro de ligne actuellement exécuté ou encore `PHP_VERSION`, qui contient la version de PHP installée sur le serveur (`_FILE_` et `_LINE_` sont des exceptions et sont appelées constantes magiques, car leur valeur n'est pas figée comme dans le cas d'une véritable constante mais peut évoluer au cours du programme).

Voici un exemple d'utilisation de constantes :

Code 21-13 : déclaration et utilisation de constantes :

```
define("ANNEENAISSANCE",1960);
define("TEXTE","Je suis né ");
echo TEXTE."en".ANNEENAISSANCE;
//affiche alors "Je suis né en 1960";
```

Expressions et instructions

Les expressions

On appelle « expression » tout regroupement d'éléments du langage de script qui peut produire une valeur (une simple variable `$a` peut donc être considérée comme une expression puisqu'elle produit une valeur lors de son évaluation). Les expressions sont constituées de variables, d'opérateurs ou de fonctions. On les utilise pour construire des instructions ; dans ce cas, elles sont terminées par un point-virgule et elles peuvent être exécutées. Les expressions servent également pour élaborer des conditions de tests, que nous allons étudier dans le chapitre dédié aux structures de programme ; dans ce cas, elles sont converties en booléen `true` ou `false` lors de leur évaluation par l'interpréteur PHP.

Code 21-14 : exemples d'expressions :

```
$a=100
$resultat=fonction()
if ($a>5)
```

Les instructions

On appelle « instruction » une expression terminée par un point-virgule. Les instructions sont traitées ligne par ligne lors de l'interprétation du code PHP.

Code 21-15 : exemples d'instructions :

```
$a=100; //cette première instruction affecte la valeur 100 à la variable $a
echo $a; //cette deuxième instruction affiche la valeur de $a (soit 100)
```

Les opérateurs

Les opérateurs permettent de lier des variables ou des expressions. Il existe différentes familles d'opérateurs selon les fonctions réalisées ou les expressions avec lesquelles on peut les employer (affectation, arithmétique, comparaison, logique ou de chaîne).

Opérateurs d'affectation

L'opérateur d'affectation est le plus courant. On peut aussi l'utiliser sous une forme compacte intégrant une opération arithmétique puis une affectation.

Tableau 21-7 Opérateurs d'affectation

Symbole	Définition
=	Affectation de base
+=	Addition puis affectation
-=	Soustraction puis affectation
*=	Multiplication puis affectation
/=	Division puis affectation
%=	Modulo puis affectation

L'opérateur d'affectation de base permet d'attribuer une valeur issue d'une expression. Les autres opérateurs d'affectation permettent en outre de réaliser des opérations arithmétiques.

Code 21-16 : exemples d'affectations :

```
$var1=0; //affectation de base (initialisation de $var1 à 0)
echo $var1;
$var1+=2; //ici $var1 vaut 2 (0+2)
echo $var1;
$var1+=14; //et maintenant $var1 vaut 16 (2+14)
echo $var1;
```

Opérateurs arithmétiques

Lorsqu'on gère des variables de type numérique, on dispose d'opérateurs arithmétiques qui peuvent réaliser toutes les opérations mathématiques standards.

À noter

Si l'on désire forcer la priorité d'exécution d'une opération, il est possible d'utiliser les parenthèses pour encadrer l'expression à exécuter en premier.

Enfin, l'incrémentation et la décrémentation (addition ou soustraction d'une unité) sont souvent utilisées en programmation et PHP fournit des opérateurs spécifiques pour ce faire (++ et --).

Tableau 21-8 Les opérateurs arithmétiques permettent d'appliquer des opérations mathématiques à des variables de type numérique

Symbole	Définition
+	Addition
-	Soustraction
/	Division
*	Multiplication
%	Modulo : l'expression $a % $b retourne le reste de la division de $a par $b
++	Incrément ($a++ ou ++$a)
--	Décrément ($a-- ou --$a)

Code 21-17 : exemples d'opérateurs arithmétiques :

```
$var1=5+2; //addition de deux valeurs numériques
echo $var1;
$var2=2+$var1; // addition d'une valeur numérique et d'une variable
echo $var2;
//----------------------------------
$var5=14;
$var4=$var5%5; // 14 modulo 5 est égal à 4 (14/5=2 et reste 4)
echo $var4 ;
//----------------------------------
$var3=($var2+$var1)/2;
//utilisation des parenthèses pour forcer les priorités des opérateurs
echo $var3;
++$var3; //après cette incrémentation, la variable est égale à "$var3+1"
echo $var3;
```

Opérateurs de comparaison

Les opérateurs de comparaison sont utilisés dans les expressions de condition des structures de programme (voir le chapitre sur les structures de programme) ou avec l'opérateur ternaire présenté ci-après. Ils permettent de comparer deux expressions. L'expression qui résulte de cette comparaison est égale à `true` (`vrai` ou `1`) lorsque la condition à contrôler est vérifiée et à `false` (`faux` ou `0`) dans le cas contraire.

Tableau 21-9 Opérateurs de comparaison

Symbole	Définition
==	Égal
<	Strictement inférieur
>	Strictement supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal
!=	Différent

L'opérateur de comparaison permet d'élaborer des expressions de condition que vous pouvez utiliser dans les instructions de structure du programme (`if`, `while`, `for`…).

Code 21-18 : exemple d'utilisation d'expressions de condition :

```
if($var1>2) //teste l'expression de condition
 echo "le test est positif";
```

Opérateur ternaire

L'opérateur ternaire permet de tester rapidement une expression de condition et d'effectuer une action spécifique selon le résultat du test. Nous verrons qu'il est possible d'utiliser les structures de choix (avec les instructions `if` et `else`) pour réaliser la même action. Dans ce cas, la syntaxe est moins concise.

Tableau 21-10 Opérateur ternaire

Syntaxe
[expression de condition]?[expression effectuée si vrai]:[expression effectuée si faux]
Exemple: `($var1>2)?($var3="oui"):($var3="non")` dans le cas où `$var1` est supérieure à 2, `oui` est affecté à `$var3`, sinon c'est `non`.

L'opérateur de choix ternaire permet de réaliser l'équivalent d'une petite structure de choix utilisant `if` et `else`.

Code 21-19 : exemple d'utilisation d'un opérateur ternaire :

```
$lg="fr";
echo ($lg=="fr")?"bonjour":"hello";
```

Opérateurs logiques

Les opérateurs logiques permettent de composer des expressions de condition complexes à partir de variables booléennes ou d'autres expressions de condition. Vous pouvez utiliser les parenthèses pour forcer les priorités entre les opérateurs ou pour améliorer la lisibilité du code en encadrant les expressions de condition.

Tableau 21-11 Opérateurs logiques

Symbole	Exemple	Fonction	Définition
&&	$var1 && $var2	ET	Renvoie true (vrai) si les deux variables $var1 ET $var2 sont true.
AND	$var1 AND $var2	ET	
||	$var1 || $var2	OU	Renvoie true (vrai) si au moins l'une des deux variables $var1 OU $var2 est true.
OR	$var1 OR $var2	OU	
XOR	$var1 XOR $var2	OU exclusif	Renvoie true (vrai) si l'une des deux variables $var1 OU $var2 est true, mais pas les deux.
!	!$var1	Négation	Renvoie la négation de $var1.

L'opérateur logique permet de relier logiquement deux expressions booléennes.

Code 21-20 : exemples d'opérateurs logiques :

```
if($var1>2) AND ($var1<5)
  echo "la variable ".$var1." est plus grande que 2 mais inférieure à 5";
//-------------------------------
if($var1=="jean") OR ($var1=="fred") OR ($var1=="marie")
  echo "la variable ".$var1." est jean, fred ou marie";
```

Opérateur de concaténation

L'opérateur de concaténation est souvent utilisé pour former des expressions à partir d'éléments de différents types (variable avec du texte par exemple) ou à partir d'autres expressions. L'opérateur utilisé pour relier ces expressions est le point. Comme pour les opérateurs arithmétiques, il existe une forme compacte d'affectation d'une concaténation. Elle est fréquemment utilisée pour regrouper différentes expressions dans une même variable (voir exemples ci-dessous).

Tableau 21-12 Opérateur de concaténation

Syntaxe		
Forme normale : `expression3 = expression1.expression2 ;`		
Exemple : `$var3=$var1."euros" ; // si $var1 est égale à 50, alors $var3 est égale à "50 euros"`		
Forme compacte : `expression3 .= expression2 ;`		
Exemple	`$var1="Monsieur";` `$var1.="Dupond";`	Après ces deux instructions, $var1 est égale à « Monsieur Dupond »

L'opérateur de concaténation permet de regrouper deux expressions et d'affecter ce résultat à l'expression de gauche.

Code 21-21 : exemples d'opérateurs de concaténation :

```
$var1="Jean";
$var2="Fred";
$var3=$var1." et ".$var2;
echo $var3; //on affiche "Jean et Fred" à l'écran
//------------------------------
$var3="Bonjour";
$var3.=$var1;
$var3.=" et ";
$var3.=$var2;
echo $var3; //on affiche "Bonjour Jean et Fred" à l'écran.
```

Bibliothèques de fonctions intégrées à PHP

On appelle bibliothèque un ensemble de fonctions qui permettent de réaliser des tâches relatives à un même domaine. On distingue les bibliothèques de fonctions utilisateurs et les bibliothèques natives PHP. Dans le deuxième cas, les fonctions intégrées sont facilement identifiables par leur nom, qui contient souvent un préfixe commun à chaque bibliothèque (exemple : `mysql_connect()`, `strpos()`…). La liste des fonctions intégrées est longue. Nous en avons sélectionné quelques-unes que nous présentons ci-dessous. Nous vous invitons à consulter la documentation PHP (`www.php.net`) si vous désirez exploiter une fonction spécifique.

Fonctions PHP générales

Tableau 21-13 Les principales fonctions PHP générales

Syntaxe et exemples	Description
`empty($var)`	Retourne le booléen `false` si la variable `$var` est définie ou bien prend une valeur. Retourne `true` dans le cas contraire.
`define(nom_cst,valeur,option)`	Définit une constante `nom_cst` qui a pour valeur « valeur ». Par défaut, le nom de la constante est sensible à la casse, mais si l'argument option est égal à 1, le nom est insensible à la casse.
`header("chaîne")`	Produit un en-tête HTTP comme `Content-type`, `Location`, `Expires`…
`isset($var)`	Retourne `true` si la variable `$var` est définie et prend une valeur. Retourne `false` dans le cas contraire.
`md5("chaîne")`	Fonction de codage qui retourne une chaîne de 32 octets associée à l'argument chaîne.
`phpinfo()`	Affiche les informations sur l'interpréteur PHP installé sur le serveur.
`rand()`	Retourne une valeur aléatoire.

Fonctions PHP dédiées aux tableaux

Les tableaux de variables sont fréquemment employés en programmation. PHP propose de nombreuses fonctions pour gérer leur contenu (tri, navigation dans le tableau, accès aux éléments…). Le tableau 21-14 récapitule les fonctions les plus couramment employées.

Tableau 21-14 Principales fonctions PHP dédiées aux tableaux

Syntaxe et exemples	Descriptions
`array_walk($tab1,nom_fonction)`	Exécute la fonction `nom_fonction` sur chaque élément du tableau `$tab1`.
`arsort($tab1)`	Trie le tableau `$tab1` par ordre décroissant.
`asort($tab1)`	Trie le tableau `$tab1` par ordre croissant.
`count($tab1)`	Retourne le nombre d'éléments contenus dans le tableau `$tab1`.
`current($tab1)`	Retourne l'élément courant du tableau `$tab1` mais n'avance pas le pointeur interne du tableau.
`each($tab1)`	Retourne chaque paire clé/valeur du tableau `$tab1` et avance le pointeur interne du tableau. Si le pointeur arrive à la fin du tableau, retourne `false`.

Fonctions PHP dédiées aux dates

Les fonctions de date et d'heure de PHP se réfèrent au tampon horaire UNIX. Ce tampon contient le nombre de secondes qui se sont écoulées depuis le début d'UNIX, soit le premier janvier 1970 à 0 h 00 (heure Greenwich GMT).

Tableau 21-15 Principales fonctions PHP dédiées aux dates

Syntaxe et exemples	Description

`checkdate($month,$day,$year)`
`checkdate(03,20,2002);`
`//retourne true car la date du 20 mars 2002 existe`
	Vérifie la validité d'une date et retourne `true` si elle est valide et `false` dans le cas contraire.

`date("format",$date)`
`$aujourdhui=date("d/m/y");`
`//retourne la date du jour au format suivant : 30/01/03`
	Formate une date. Retourne une chaîne de caractères au format imposé par format (il existe de nombreux formats possibles). La date est fournie par `$date`, mais si ce paramètre n'est pas mentionné, c'est par défaut la date courante qui est retournée.

`mktime($h,$min,$sec,$mon,$day,$year[,int is_dst])`
`mktime(10,0,0,1,25,2003,1);`
`//retourne en secondes : 1043481600`
	Retourne le tampon UNIX (temps référence en secondes depuis le 1er janvier 1970) correspondant à la date indiquée en paramètre.
	À noter : l'argument optionnel `is_dst` permet de tenir compte de l'heure d'hiver (si l'heure d'hiver est appliquée, `is_dst=1` et 0 sinon).

`getdate(tampon_UNIX)`
`//ce script affiche l'heure actuelle`
`//par exemple : il est 18 h 45`
`$datejour=getdate();`
`$heure=$datejour[hours];`
`$min=$datejour[minutes];`
`echo "il est ".$heure." h ".$min;`
	Appelée sans paramètre, cette fonction retourne un tableau associatif qui contient la date et l'heure actuelle. Si une valeur de tampon UNIX est indiquée, le tableau contient les informations en rapport avec ce tampon. Pour récupérer une information de ce tableau, il faut utiliser la clé adaptée, par exemple :
	`$aujourdhui=getdate();`
	`echo $aujourdhui[mday]; //jour du mois (numérique de 1 à 31);`
	`echo $aujourdhui[mon]; // mois (numérique de 1 à 12);`
	`echo $aujourdhui[year]; //année (numérique, par ex : 2004);`

Fonctions PHP dédiées aux chaînes de caractères

Certaines fonctions de gestion des chaînes de caractères sont très importantes dans les sites dynamiques. En effet, lorsqu'on ajoute dans une base de données une chaîne comportant des caractères spéciaux (« ' » par exemple), le serveur MySQL signale une erreur. La solution consiste à faire précéder ces caractères spéciaux d'une barre oblique inverse grâce à la fonction `AddSlashes()`. La fonction `StripSlahes()` permet ensuite de les supprimer.

Tableau 21-16 Principales fonctions PHP dédiées aux chaînes de caractères

Syntaxe et exemples	Description
`trim($chaine)`	Supprime les espaces initiaux et finaux d'une chaîne de caractères `$chaine`.
`str_replace($ch1,$ch2,$ch3)`	Substitue une chaîne `$ch2` à toutes les occurrences d'une chaîne de recherche `$ch1` dans une chaîne spécifique `$ch3`.

`StripSlashes($chaine)`
`$chaine1="aujourd\'hui";`
`$chaine2=StripSlashes($chaine1);`
`echo $chaine2;`
`//affiche "aujourd'hui"`
	Supprime les barres obliques inverses devant les caractères spéciaux de la chaîne `$chaine`.

`AddSlashes($chaine)`
`$chaine1="aujourd'hui";`
`$chaine2=AddSlashes($chaine1);`
`echo $chaine2;`
`//affiche "aujourd\'hui"`
	Ajoute une barre oblique inverse devant les caractères spéciaux d'une chaîne `$chaine`.

`strtolower($chaine)`	Renvoie la chaîne `$chaine` en minuscules.
`strtoupper($chaine)`	Renvoie la chaîne `$chaine` en majuscules.

`explode($separe,$chaine)`
`$chaine1="Paris:Rennes:Lille";`
`$tab1= explode(":",$chaine1);`
`//retourne le tableau suivant :`
`array("Paris","Rennes","Lille")`
	Scinde la chaîne `$chaine` en plusieurs parties selon le séparateur `$separe`. Retourne un tableau des valeurs ainsi séparées.

`implode($separe,$tab)`
`$tab=array("Paris","Rennes","Lille");`
`$chaine1= implode(":",$tab);`
`echo $chaine1;`
`/* affiche la chaîne suivante :`
`"Paris:Rennes:Lille" */`
	Retourne une chaîne de caractères constituée de tous les éléments du tableau `$tab` séparés par `$separe`.

Fonctions PHP dédiées aux fichiers

La plupart des opérations de base réalisables avec des fonctions PHP nécessitent le passage en argument d'un identifiant de fichier que nous noterons `$id` dans le tableau 21-17. Cet identifiant est obtenu lors de l'ouverture du fichier à l'aide de la fonction `fopen()`. Pour toutes les opérations d'écriture, il faut s'assurer au préalable que les droits du fichier sont configurés en conséquence (vous pouvez rapidement réaliser ce paramétrage à l'aide de la fonction `chmod` de votre outil de FTP).

Tableau 21-17 Principales fonctions PHP dédiées aux fichiers

Syntaxe et exemples	Description
`fopen("nom_fichier","mode")` `$id=fopen("monfichier.txt","r");`	Ouverture d'un fichier selon le mode « mode » (voir détail ci-dessous). Retourne un identifiant (`$id` par exemple) nécessaire dans d'autres fonctions de manipulation de fichiers.
Le paramètre mode de la fonction `fopen()` définit les modes d'ouverture de fichier suivants : - « r » : ouvre un fichier en lecture seule et place le pointeur en début du fichier. - « r+ » : ouvre un fichier en lecture et en écriture et place le pointeur en début du fichier. - « w » : ouvre un fichier en écriture seule et place le pointeur en début du fichier, mais en supprimant le contenu existant. Si le fichier n'existe pas, il est créé. - « w+ » : ouvre un fichier en lecture et écriture et place le pointeur en début du fichier, mais en supprimant le contenu existant. Si le fichier n'existe pas, il est créé. - « a » : ouvre un fichier en écriture seule et place le pointeur à la fin du fichier (ajout). Si le fichier n'existe pas, il est créé. - « a+ » : ouvre un fichier en lecture et écriture et place le pointeur à la fin du fichier (ajout). Si le fichier n'existe pas, il est créé.	

`copy($source,$destination)`
`$fichier="index.php";`
`copy($fichier,$fichier.'.bak');`
`//ici, on réalise une copie de sauvegarde "index.php.bak"`
	Copie le fichier `$source` dans le fichier `$destination` (si `$destination` n'existe pas, il est créé). Si l'opération est effectuée, la fonction retourne `true` sinon `false`.

`fpassthru("nom_fichier")`	Lit complètement le fichier et l'affiche dans le navigateur. Le fichier peut être un fichier texte ou un fichier image (binaire).
`file_exists("nom_fichier")`	Retourne la valeur `true` si le fichier existe. Sinon retourne `false`.
`file("nom_fichier")`	Lit un fichier et retourne un tableau contenant une ligne du fichier dans chacun de ses éléments.
`fgetc($id)`	Lit un caractère du fichier ouvert avec `$id`.
`fgets($id, "nombre")` `fread($id,"nombre")`	Lit un nombre donné d'octets (nombre) du fichier ouvert avec `$id`.
`fputs($id,"chaine")` `fwrite($id,"chaine")`	Écrit une chaîne de caractères (chaîne) dans un fichier ouvert avec `$id`.
`rewind($id)`	Place le pointeur du fichier au début du fichier ouvert avec `$id`.
`feof($id)`	Retourne la valeur `true` si le pointeur de fichier se trouve à la fin du fichier ou s'il y a une erreur. Sinon retourne `false`.
`fclose($id)`	Ferme un fichier ouvert par la fonction `fopen()`.

Code 21-22 : dans l'exemple ci-dessous, la valeur du fichier est lue, incrémentée puis enregistrée dans le même fichier :

```
<?php
$fichier=fopen("monfichier.txt","r+");
//ouverture du fichier "monfichier.txt" en mode lecture/écriture
$sortie=fgets($fichier,10);
$sortie++; //incrémente la valeur
fseek($fichier,0); //positionne le pointeur en début de fichier
fputs($fichier, $sortie);
//écriture de la nouvelle valeur dans le fichier
fclose($fichier); // fermeture du fichier
?>
```

Attention aux droits des répertoires sur votre site distant

Si vous désirez utiliser des fonctions dédiées aux fichiers sur votre site distant, sachez qu'il est alors nécessaire de configurer les droits du répertoire contenant le fichier concerné. Pour changer les droits d'un répertoire, la méthode la plus simple est d'utiliser la fonctionnalité CHMOD de votre client FTP (par exemple avec le client FTP de Dreamweaver, il suffit de faire un clic droit sur le répertoire distant à modifier et de choisir l'option « gérer les autorisations »).

Fonctions PHP dédiées à MySQL

PHP dispose de nombreuses fonctions dédiées à la gestion de la base MySQL. Ces fonctions commencent toujours par `mysql_`. Nous vous présentons ci-dessous celles qui sont le plus couramment utilisées.

Tableau 21-18 Principales fonctions PHP dédiées à MySQL

Syntaxe et exemples	Description

`mysql_connect ("nom_hote","login","password");`
`//ci-dessous la même fonction avec l'option connexion persistante`
`mysql_pconnect ("nom_hote","login","password");`
`//exemple : création d'un identifiant de connexion "$id"`
`$id=mysql_connect ("locahost","sport","eyrolles");`
	Permet d'établir la connexion avec le serveur de base de données MySQL. Le nom de l'hôte, l'identifiant et le mot de passe sont communiqués comme arguments de cette fonction. Ils doivent évidemment être configurés au préalable sur le serveur MySQL. Dans le cas d'un hébergement mutualisé, ces informations vous sont communiquées par votre hébergeur.
	Cette fonction retourne un identifiant de connexion qu'il convient d'enregistrer dans une variable (exemple : `$id`), afin de pouvoir l'utiliser dans les autres fonctions MySQL.

`mysql_select_db ("nom_base_donnees");`
`//exemple : sélection de la base de données "sport_db"`
`mysql_select_db ("sport_db");`
	Permet de sélectionner une base de données dont le nom est communiqué en argument de la fonction.

`mysql_query ("requete_SQL","id_connexion");`
`//exemples :`
`$result=mysql_query ("SELECT * FROM adherents ", $id);`
`$result=mysql_query ($query,$id);`
	Permet d'envoyer une requête SQL au serveur. Vous pouvez saisir la requête directement dans la fonction ou la déterminer au préalable dans une variable (exemple : `$query`) et la passer en argument dans la fonction.

`mysql_fetch_array ("resultat_query" [,type_tableau]);`
`//argument optionnel : type_tableau :`
`//MYSQL_ASSOC : tableau associatif`
`//MYSQL_NUM : tableau indicé`
`//MYSQL_BOTH : tableau mixte (par défaut)`
`//exemple :`
`$membres=mysql_fetch_array ($result);`
	Permet de fournir le résultat de la requête dans un tableau associatif, indicé ou mixte (selon l'argument optionnel `type_tableau`). La variable contenant le résultat de la requête doit être passée en paramètre dans le premier argument (exemple : `$result`).

`mysql_free_result ("id_query");`
`//exemple :`
`mysql_free_result ($result);`
	Permet de vider les résultats obtenus à l'aide de la précédente requête avant de sortir de la procédure.

Code 21-23 : l'exemple ci-dessous est un script complet de connexion et d'affichage des noms des joueurs stockés dans une base nommée `machineasous` :

```
$id=mysql_connect("localhost","machine","1234");
//Le serveur est "localhost", l'identifiant "machine" et le mot de passe "1234"
(reportez vous au chapitre 22 pour créer des paramètres de connexion MySQL.
```

```
//-----------------------------------------------
mysql_select_db("machineasous");
//Sélection de la base de données "machineasous".
//-----------------------------------------------
$query="SELECT * FROM joueurs";
//Élaboration de la requête SQL et enregistrement dans la variable "$query".
//-----------------------------------------------
$result=mysql_query($query,$id);
//La requête "$query" est envoyée au serveur accompagnée de son identifiant "$id"
//-----------------------------------------------
while ($joueurs=mysql_fetch_array($result))
{
 echo "Le joueur ".$joueurs[nom]." est inscrit au club de jeu <br>";
}
//Récupération des données dans le tableau $joueurs[] et affichage ("nom" est un des
➥champs retournés par le serveur de base de données).
//-----------------------------------------------
mysql_free_result($result);
//Libère la mémoire des résultats obtenus.
```

Ne mélangez pas les langages !

Nous avons vu qu'une page dynamique était composée de scripts PHP intégrés dans une structure de page HTML. Le langage SQL prend place dans la même page dynamique. On le retrouve notamment en argument de la fonction PHP `mysql_query()`, qui permet de transmettre la requête SQL à la base de données. Ces trois langages sont donc intégrés dans le même code d'une page dynamique et il est très important de ne pas les mélanger : ce sont des langages différents dont il faut veiller à utiliser la syntaxe correspondante.

Fonctions utilisateur

Dans la partie précédente, nous avons déjà présenté de nombreuses fonctions intégrées par défaut dans PHP. Cependant, en pratique il est souvent nécessaire de créer des fonctions utilisateur adaptées aux besoins du programme.

Gestion des fonctions utilisateur

Déclaration et utilisation des fonctions

Une fonction permet d'exploiter une même partie de code à plusieurs reprises dans un programme, ce qui est très intéressant pour les routines standard souvent utilisées en programmation (affichage d'une valeur, calculs mathématiques courants, conversions…). PHP propose de nombreuses fonctions intégrées en standard, que nous étudierons dans la section suivante (`echo()`, par exemple). Vous pouvez également réaliser vos propres fonctions en les déclarant à l'aide du mot-clé `function()`. Dans une fonction, il est possible d'exploiter des variables sans risque de conflit avec celles du programme principal, car elles n'ont qu'une portée locale. La déclaration d'une fonction comporte une tête et un corps. Le mot-clé `function` est placé dans la tête de la fonction, suivi du nom de celle-ci et, entre parenthèses, de la liste des arguments attendus séparés par une virgule (lorsque la fonction ne comporte pas d'argument, les parenthèses sont vides). La tête de la fonction est suivie du corps encadré par des accolades (comme pour un bloc).

Les fonctions ont généralement une valeur de retour, désignée par le mot-clé `return`, suivi du résultat retourné dans le programme.

Important

La valeur retournée avec `return` (exemple : `return $res`) se substitue à l'appel de la fonction (exemple : `moyenne(4,6)`) dans le programme principal. Il est alors possible d'affecter ce résultat à une autre variable pour l'exploiter ultérieurement.

Par exemple, si la fonction est appelée de cette manière :

```
$moncalcul=moyenne(4,6);
```

et si le résultat retourné est `$res` (`return $res`), l'appel de la fonction est alors équivalent à l'affectation suivante :

```
$monCalcul=$res;
```

Tableau 21-19 Déclaration d'une fonction utilisateur

Syntaxe de la déclaration d'une fonction	

`function nom_de_fonction (arg1,arg2…)`
`{`
`  instruction1;`
`  instruction2;`
`  …`
`  [return $var0;]`
`}`	

Remarques	Le nom de la fonction ne doit comporter ni espace ni point. Les parenthèses permettent de passer des arguments séparés par des virgules. Les arguments peuvent être des variables ou des constantes. La durée de vie d'une variable de fonction est limitée au temps d'exécution de la fonction. `return` permet de retourner une variable résultat `$var0` dans le programme, mais sa présence n'est pas obligatoire. `[xxx]` : le code `xxx` est facultatif.

Tableau 21-20 Utilisation d'une fonction utilisateur

Syntaxe de l'utilisation d'une fonction
`nom_de_fonction (arg1,arg2…);`

Code 21-24 : création d'une fonction `moyenne()` pour le calcul de la moyenne de deux valeurs :

```
//Déclaration de la fonction -------------------------------------
function moyenne ($a,$b) //tête de la déclaration
{ //début du corps
 $res=($a+$b)/2; //instructions de la fonction
 return $res; //information retournée au programme
} //fin du corps
//Utilisation de la fonction dans le programme ------------------
$moncalcul=moyenne(4,6); //appel de la fonction
echo "la moyenne de 4 et de 6 est égale à $moncalcul";
```

Utilisation avancée des fonctions

Retour de plusieurs résultats

Avec le mot-clé `return`, il est possible de renvoyer une variable résultat dans le programme principal. Cependant, dans certaines applications, il est utile de récupérer plusieurs valeurs de résultat. Dans ce cas, la solution consiste à utiliser un tableau en guise de variable retournée par `return()`, comme le montre l'exemple ci-dessous.

Code 21-25 : retour de variables à l'aide d'un tableau :

```
//Déclaration de la fonction --------------------
function calcul($a,$b,$c)
{
  $res1=($a+$b);
  $res2=($a+$c);
  $res=array ($res1,$res2);
  //le résultat est un tableau de valeurs
  return $res;
}
//Utilisation de la fonction dans le programme --
$moncalcul=calcul(4,6,5);
echo $moncalcul[0];
echo $moncalcul[1];
/* appel de la fonction et récupération du résultat dans le tableau $moncalcul, les deux
➥résultats du calcul peuvent ainsi être exploités dans le programme principal. */
```

Arguments facultatifs

Les arguments d'une fonction peuvent devenir facultatifs s'ils sont paramétrés avec leur valeur par défaut dans la déclaration. La valeur par défaut d'un argument doit obligatoirement être une constante. De même, il convient de placer les arguments dotés d'une valeur par défaut à la fin de l'énumération des arguments dans la parenthèse. L'exemple ci-dessous reprend la même fonction `moyenne()` que précédemment, mais avec la possibilité de passer un troisième argument optionnel : si ce troisième argument n'est pas passé dans l'appel de la fonction, il prend sa valeur par défaut, soit la valeur `euros`.

Code 21-26 : exemple avec arguments facultatifs :

```
//Déclaration de la fonction --------------------
function moyenne($a,$b,$c="euros")
// $c est initialisée avec "euros" par défaut
{
  $res=($a+$b)/2;
  $res.=$c;
  //ajoute l'unité au résultat exemple : 5 euros
  return $res;
}
//Utilisation de la fonction dans le programme --
$moncalcul=moyenne(4,6);
/* appel de la fonction avec deux arguments uniquement */
echo "la moyenne de 4 et de 6 est égale à $moncalcul";
/* dans ce cas, la ligne ci-dessus affiche :
"la moyenne de 4 et de 6 est égale à 5 euros" */
```

Variables globales

Par défaut, les variables définies dans une fonction ont une portée limitée à celle-ci (variables locales). Ainsi, dans l'exemple précédent, vous pouvez très bien exploiter la variable `$res` à la fois dans la fonction et dans le programme principal sans risque de conflit. De même, il n'est pas possible de récupérer la valeur qui a été affectée à cette variable dans le programme principal sans la retourner avec le mot-clé `return`. Il existe toutefois une solution pour augmenter la portée d'une variable afin qu'elle puisse être exploitée dans la fonction et dans le programme principal (variable globale) : il faut la déclarer dans la fonction en utilisant le préfixe `global`, comme le montre l'exemple ci-dessous (selon la version de votre environnement PHP, la syntaxe pour la déclaration et l'utilisation des variables globales peut varier mais le principe reste le même).

Code 21-27 : exemple d'utilisation d'une variable globale :

```
//Déclaration de la fonction --------------------
function moyenne($a,$b)
{
  global $res; //la variable $res est maintenant de type global
  $res=($a+$b)/2;
}
//Utilisation de la fonction dans le programme --
moyenne(4,6);
/* La variable $res peut maintenant être exploitée dans le programme principal */
echo $res; //affiche la valeur 5
```

Utilisation de fonctions externes

Inclusion de fichiers avec require()

Pour éviter de déclarer dans chaque programme vos fonctions utilisateur, vous pouvez les regrouper dans un même fichier, `mesfonctions.php` par exemple, inséré dans le script grâce à la commande `require()`. Si le fichier appelé ne se trouve pas dans le même répertoire que le fichier appelant, il faut préciser le chemin pour accéder au fichier bibliothèque (exemple : `require("bibliotheques/mesfonctions.php")`). En outre, la variante `require_once()` gère l'inclusion multiple sans générer d'erreur (l'inclusion d'un même fichier avec `require()` ne pouvant être effectué qu'une seule fois).

Code 21-28 : exemple d'inclusion de fichier avec `require()` :

```
//-----------------------------------------------
//contenu du fichier "mesfonctions.php"
<?php
function moyenne ($a,$b)

  $res=($a+$b)/2;
  return $res;
}
?>
//-----------------------------------------------
/* Contenu d'un autre fichier utilisant les fonctions du fichier "mesfonctions.php" */
<?php
require("mesfonctions.php");
$moncalcul=moyenne(4,6);
// appel de la fonction
echo "la moyenne de 4 et de 6 est $moncalcul";
?>
```

Inclusion de fichiers avec include()

La commande `require()` est bien adaptée à l'inclusion d'un fichier regroupant des fonctions, car il ne doit être appelé qu'une seule fois au début de la page. Cependant, cette commande n'est pas une fonction et reste indépendante des structures de programme. Si vous désirez ajouter un fichier géré par une structure de programme, il faut utiliser la fonction `include()`. Cette fonction est bien adaptée lorsque vous voulez inclure des blocs de code plusieurs fois dans la même page ou les conditionner par une instruction de choix.

Code 21-29 : exemple avec `include()` :

```
if ($condition=="oui")
 include('fichier.php');
```

Structures de programme

Les structures des programmes permettent de créer des scripts qui réagissent différemment selon des événements (structures de choix) ou exécutent d'une manière répétitive le même bloc d'instructions (structure de boucle).

Structures de choix

Structures de choix avec if

Les structures de choix sont utilisées pour traiter les alternatives logiques au cours de l'exécution du script, afin d'orienter le déroulement du programme en fonction du résultat de l'alternative. Elles comprennent en général une expression de condition. Les expressions de condition sont constituées de variables ou de constantes reliées par des opérateurs logiques. Si l'expression de condition entre parenthèses est vraie, l'instruction qui suit est exécutée, sinon il ne se passe rien et le programme continue de se dérouler après le bloc du `if`.

Tableau 21-21 Instruction conditionnelle if

Syntaxe

`if (expression_de_condition)`
`{`
`  instruction1;`
`  instruction2;`
`  …`
`}`

Forme simplifiée sans accolades (s'il n'y a qu'une seule instruction à traiter) : `if (expression_de_condition) instruction1;`

Code 21-30 : exemple de structure `if` :

```
if ($var1>4)
 echo "la valeur est supérieure à 4";
```

Structures de choix avec if et else

La structure de choix utilisant l'instruction `if` ne traite que les structures de programme pour lesquelles la condition est vraie ; dans le cas contraire, aucune instruction n'est exécutée. Avec l'instruction `else`, vous pouvez définir les instructions à exécuter dans le cas où la condition testée serait fausse. Ces instructions sont regroupées dans un autre bloc qui suit l'instruction `else`.

Tableau 21-22 Instructions conditionnelles `if` et `else`

Syntaxe

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

Code 21-31 : exemple de structure `if else` :

```
if ($var1>4)
{
 echo "la valeur est supérieure à 4";
 echo "<br> elle est exactement égale à $var1 ";
}
else
{ //début du bloc else
 echo "la valeur est inférieure ou égale à 4";
 echo "<br> elle est exactement égale à $var1";
} //fin du bloc else
```

Structures de choix avec if et elseif

En pratique, lors d'un choix, plusieurs conditions doivent être testées. Dans ce cas, il faut utiliser l'instruction `elseif`, qui est en quelque sorte une combinaison du `else` et du `if` suivant ; elle se place à la suite d'une instruction `if` pour introduire le bloc à exécuter au cas où sa condition serait fausse (comme le `else`) et introduit une nouvelle condition (comme le `if`). Vous pouvez ainsi créer autant de conditions imbriquées que vous le souhaitez selon le nombre d'instructions `elseif` utilisées.

Tableau 21-23 Instructions conditionnelles `if`, `elseif` et `else`

Syntaxe

```
if (expression_de_condition)
{
  instruction1;
  instruction2;
  …
}
elseif (expression_de_condition)
{
  instruction3;
  instruction4;
  …
}
else
{
  instruction5;
  instruction6;
  …
}
```

Code 21-32 : exemple de structure `if elseif else` :

```
if ($var1>4)
{
 echo "la valeur est supérieure à 4";
 echo "<br> elle est exactement égale à $var1";
}
elseif ($var1>2)
{//début du bloc elseif
 echo "la valeur est supérieure à 2 mais inférieure ou égale à 4";
 echo "<br> elle est exactement égale à $var1";
}//fin du bloc elseif
else
{
 echo "la valeur est inférieure ou égale à 2";
 echo "<br> elle est exactement égale à $var1";
}
```

Structures de choix avec switch … case

L'instruction `switch` permet de tester l'égalité d'une valeur passée en paramètre (`valeur_testée`) avec une série de valeurs possibles (`valeur1`, `valeur2`…). Si l'une des valeurs correspond à la valeur testée, le bloc d'instructions correspondant est exécuté. L'exécution des instructions doit se terminer par une instruction `break`, afin que le programme puisse sortir de la structure de choix. On peut ajouter une branche `default` à la fin d'un bloc, afin de traiter tous les cas non prévus dans la structure.

Tableau 21-24 Instruction `switch … case`

Syntaxe

`switch (valeur_testée)`
`{`
`  case valeur1:`
`  instruction1;`
`  break;`
`  case valeur2:`
`  instruction2;`
`  break;`
`  case valeur3:`
`  instruction3;`
`  break;`
`  …`
`  …`
`  default:`
`  instructionD;`
`}`

Code 21-33 : programme permettant d'afficher « bonjour » dans la langue de l'internaute réalisé avec une structure `switch case` :

```
$var1="fr";
//cette variable mémorise la langue choisie par l'internaute
switch($var1)
{
case "fr":
```

```
 echo "Bonjour";
 break;
case "en":
 echo "Hello";
 break;
case "es":
 echo "Hola";
 break;
case "de":
 echo "Guten Tag";
 break;
}
```

Structures de boucle

Structures de boucle avec while

Lorsqu'un ensemble d'instructions doit être exécuté plusieurs fois en fonction d'une condition, il faut utiliser les structures de boucle. En PHP, il existe plusieurs types de boucles. Avec l'instruction `while`, le bloc d'instructions est exécuté et répété tant que l'expression de condition retourne `true`. Lorsque la condition est ou devient fausse (`false`), le programme sort de la boucle pour exécuter les instructions qui se trouvent après la fin du bloc.

Dans cette structure, il est fréquent d'utiliser une variable dédiée pour le compteur de boucle (exemple : `$i`). Vous devez initialiser cette variable avant la boucle. Elle est ensuite testée dans l'expression de condition, puis incrémentée (ou décrémentée selon les cas) dans le corps de boucle. La valeur de l'expression de condition étant évaluée avant chaque début de boucle, les instructions du bloc peuvent ne jamais être exécutées si la condition est évaluée à `false` dès le début. Pour faire évoluer le compteur de boucle, on utilise généralement un opérateur d'incrémentation ou de décrémentation (`$i++` ou `$i--`). Choisissez le bon type d'opérateur, en fonction de la valeur de l'initialisation du compteur et de l'expression de condition choisie, sinon vous risquez d'obtenir une boucle infinie.

Tableau 21-25 Instruction de boucle `while`

Syntaxe

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

Code 21-34 : exemple de boucle `while` :

```
$i=5; //initialisation du compteur de boucle à 5
while($i>0)
{
  echo "Encore $i tour(s) à faire <br>";
  $i--; //décrémentation du compteur de boucle
}
echo "Voilà, c'est enfin terminé";
/* ci-dessus un exemple qui affiche cinq fois le même texte (tant que $i est supérieur
➥à 0) avant d'afficher le texte final. */
```

Structures de boucle avec for

L'instruction `for` est une seconde solution pour traiter les boucles. Sa syntaxe est cependant différente de celle de la structure précédente, car les parenthèses de l'instruction regroupent trois expressions déjà présentes dans la boucle `while`, séparées par des points-virgules. La première expression correspond à l'instruction d'initialisation de la boucle `while`, la seconde à son expression de condition et la troisième à l'instruction d'incrémentation ou de décrémentation située auparavant dans le corps de la boucle `while`. Cette syntaxe très compacte est particulièrement appréciable quant à la lisibilité du code.

Tableau 21-26 Instruction de boucle for

Syntaxe	

`for (expression1;expression2;expression3)`
`{`
`  instruction1;`
`  instruction2;`
`  …`
`}`	

Légende	`expression1` : expression évaluée en début de boucle. Fréquemment utilisée pour initialiser le compteur de boucle à l'aide de l'opérateur d'affectation (exemple : « `$i = 5` »). `expression2` : expression évaluée au début de chaque passage de boucle. Si le résultat de l'évaluation est `true` (vrai), le bloc d'instructions de la boucle est de nouveau exécuté. Dans le cas contraire, le programme sort de la boucle pour exécuter les instructions qui suivent le bloc. Cette expression est fréquemment utilisée pour tester le compteur de boucle à l'aide d'un opérateur de comparaison (exemple : `$i > 0`). `expression3` : expression évaluée à la fin de chaque boucle. Fréquemment utilisée pour incrémenter ou décrémenter le compteur de boucle à l'aide d'un opérateur d'auto-incrémentation ou de décrémentation (exemple : `$i--`). À noter : pour chaque zone (délimitée par un point-virgule), il est possible d'exécuter plusieurs expressions, qui doivent être séparées par de simples virgules. Cela permet notamment de gérer plusieurs variables de compteur (exemple : `for ($i=5,$x=1;$i>0;$i--,$x++)`).

Code 21-35 : exemple d'une boucle `for` :

```
for ($i=5;$i>0;$i--)
{
  echo "Encore $i tour(s) à faire <br>";
}
echo "Voilà, c'est enfin terminé";
//ci-dessus un exemple qui réalise la même boucle que celle donnée en exemple pour
➥l'instruction "while".
```

Structures de boucle avec foreach

La boucle `foreach` est dédiée à la manipulation des tableaux de variables. Elle permet en effet de lire rapidement le contenu d'un tableau sans avoir à écrire beaucoup de code. Vous avez le choix entre deux syntaxes selon le type de tableau (indicé ou associatif). Le principe de cette instruction est semblable à celui d'une boucle pour laquelle un pointeur interne au tableau est placé sur le premier élément du tableau (`0` pour les tableaux indicés et la première clé pour les tableaux associatifs). À chaque tour de boucle, la variable `$var` (voir tableau ci-dessous) contient la valeur de l'élément pointé : vous pouvez ainsi l'exploiter dans les instructions du corps de la boucle. À la fin de chaque boucle, le pointeur se déplace sur l'élément suivant dans le tableau et ainsi de suite jusqu'à la fin du tableau.

À noter

Dans la syntaxe du tableau associatif, il est également possible d'exploiter la clé de chaque élément (`$cle`) dans les instructions du corps de la boucle.

Tableau 21-27 Instruction de boucle `foreach`

Syntaxe pour tableau indicé

`foreach($tableau as $var)`
`{`
`  instruction utilisant $var;`
`}`

Syntaxe pour tableau associatif

`foreach($tableau as $cle=>$var)`
`{`
`  instruction utilisant $cle et $var;`
`}`

Code 21-36 : exemple d'application de l'instruction `foreach` :

```
$agence=array("Paris","Lille","Marseille");
foreach ($agence as $ville)
{
  echo "Ville:".$ville."<br>";
}
/* ci-dessus un exemple qui affiche toutes les villes des agences contenues dans le tableau
➥indicé $agence. */
```

Instructions de contrôle

Instruction de contrôle avec break

Dans certaines applications, il peut s'avérer nécessaire de sortir de la boucle avant que l'expression de condition ne l'impose (c'est valable pour toutes les boucles : `while`, `do…while`, `for`, `switch…case` et `foreach`). L'instruction `break` permet de quitter la boucle pour que le programme passe à l'exécution des instructions qui se trouvent après celle-ci. Si plusieurs boucles sont imbriquées, précisez combien de boucles doivent être stoppées avec l'argument `n` de l'instruction : `break n`. L'exécution du programme passe alors directement à la boucle de niveau supérieur, si elle existe (par défaut, cet argument est égal à 1).

À noter

Cette instruction est obligatoire dans les structures `switch…case` afin d'éviter d'exécuter les instructions qui suivent la branche du `case` sélectionné.

Tableau 21-28 Instruction de contrôle de boucle `break`

Syntaxe	
`break [n]`	
Légende :	`n` : nombre de boucles imbriquées qui sont interrompues. Par défaut, `n` est égal à 1. `[xxx]` : le code `xxx` est facultatif.

Code 21-37 : exemple d'application de l'instruction `break` :

```
$i=5; //initialisation du compteur de boucle
while($i>0)
{
  if ($commande[$i]="arret")
    break; //arrête la boucle si cette variable est égale à "arret"
  echo "Encore ".$i." tour(s) à faire <br>";
  $i--; //décrémentation du compteur de boucle
  }
echo "Voilà, c'est enfin terminé";
/* Voir ci-dessus pour un exemple qui reprend le script de la première boucle
"while", dans lequel on a ajouté une instruction "break" conditionnée par la
variable "$commande". Si l'expression de condition renvoie "true", le programme
sort prématurément de la boucle et le message de fin s'affiche. */
```

Instruction de contrôle avec continue

L'instruction `continue` est également une instruction de contrôle de boucle. Contrairement à l'action `break`, elle permet de se rendre au passage de boucle suivant. De même que pour `break`, on peut lui préciser, par le biais d'un argument optionnel, le nombre de passages de boucle qu'on désire court-circuiter.

Tableau 21-29 Instruction de contrôle de boucle continue

Syntaxe	
`continue [n]`	
Légende :	n : nombre de passages de boucle ignorés. `[xxx]` : le code `xxx` est facultatif. (Attention ! Vous ne devez surtout pas saisir les crochets `[` et `]` dans le code.)

Code 21-38 : exemple d'application de l'instruction `continue` :

```
for($i=5; $i>0; ++$i)
{
  if (!($i%2))
    continue; //court-circuite l'affichage des tours impairs
  echo "Encore $i tour(s) à faire <br>";
}
echo "Voilà, c'est enfin terminé";
/* Voir ci-dessus pour un exemple dans lequel on a ajouté une instruction "continue",
exécutée pour tous les tours impairs (grâce à l'utilisation de l'opérateur modulo
dans l'expression de condition !($i%2)). Au final, l'affichage du message de boucle
est réalisé uniquement sur les tours pairs. */
```

Redirection interpage

Nous venons d'étudier différentes structures qui permettent de gérer le cheminement du programme au sein d'une même page. Cependant, il faut fréquemment rediriger le programme automatiquement vers une autre page du site. Cette redirection peut être le résultat d'un test de condition (revoir les instructions de choix `if`, `else`, `elseif` et `switch`) ou bien se situer à la fin d'un script PHP, par exemple après un script d'ajout d'un nouvel enregistrement (pour rediriger l'internaute vers une page affichant la liste actualisée de tous les enregistrements présents dans la base).

La fonction `header()` permet de rediriger l'internaute vers une page ou une URL sans intervention de sa part. L'inconvénient de cette fonction est que vous devez toujours l'utiliser avant tout envoi vers le navigateur, qu'il s'agisse de codes HTML ou d'affichages provoqués par des fonctions PHP comme `echo()` ou `print()` voire d'un simple espace placé avant la balise d'ouverture PHP (`<?php`). Il faut donc veiller à ce que cette fonction soit appelée au début du script.

Tableau 21-30 Fonction de redirection `header()`

Syntaxe	
`header("Location:nom_cible")`	
Légende :	`nom_cible` : la cible vers laquelle on redirige l'internaute peut être un chemin relatif comme `monfichier.php` ou un chemin absolu comme `http://www.agencew.com`

Code 21-39 : exemple de redirection PHP par la fonction `header()` conditionnée par l'instruction `if` :

```
<?php
//ce script permet de rediriger l'internaute selon l'état de la variable
//$profil vers la page suite.php
if($profil=="admin")
  header("Location:suite.php");
?>
```

Gestion XML avec SimpleXML

Le module SimpleXML est une nouvelle extension disponible depuis PHP 5. Si vous désirez manipuler des documents XML avec PHP 4, vous devez utiliser l'analyseur syntaxique XML ou des technologies plus avancées comme SAX, DOM, XSLT… À noter que l'extension SimpleXML est installée par défaut sur toutes les distributions PHP 5, ce qui vous garantit une portabilité de vos scripts d'un serveur à l'autre.

SimpleXML se caractérise par sa simplicité d'utilisation contrairement à l'utilisation de l'extension XML beaucoup plus lourde à mettre en place en raison de la configuration de ses différents gestionnaires. Cependant, notons que si SimpleXML est très bien adapté pour analyser ou modifier des fichiers XML simples, son utilisation n'est pas recommandée pour gérer des fichiers XML complexes (il est préférable dans ce cas d'utiliser les technologies SAX, DOM ou XSLT).

Tableau 21-31 Syntaxe des principales fonctions et méthodes SimpleXML

`$SimpleXMLElement=simplexml_load_file(nomFichier)`

Fonction qui permet d'importer un document XML placé dans un fichier externe (`nomFichier`) en le transformant en élément SimpleXML. Si l'importation échoue, la fonction retourne `false`.

`$SimpleXMLElement=simplexml_load_string(nomChaine)`

Fonction qui permet de lire un document XML mémorisé dans une variable (`nomChaine`) en le transformant en élément SimpleXML. Si la lecture échoue, la fonction retourne `false`.

`$SimpleXMLElement->asXML(nomFichier)`

Cette méthode renvoie le contenu XML d'une partie (ou la totalité si vous l'appliquez sur l'élément racine) du document SimpleXML sous forme d'une chaîne de caractères qui peut être simplement affichée (s'il n'y a pas d'argument) ou enregistrée dans un fichier externe (dans ce cas le nom du fichier est celui de l'argument de la méthode : `nomFichier`).

Tableau 21-31 Syntaxe des principales fonctions et méthodes SimpleXML *(suite)*

`$SimpleXMLElement->children()` Cette méthode permet de récupérer tous les enfants d'un élément SimpleXML. La liste retournée par la méthode est structurée comme un tableau de variables et peut donc être facilement exploitée à l'aide de la fonction `foreach()`.
`$SimpleXMLElement->attributes()` Cette méthode permet de récupérer tous les attributs d'un élément SimpleXML. La liste retournée par la méthode sera structurée comme un tableau de variables et peut donc être facilement exploitée à l'aide de la fonction `foreach()`.
`$SimpleXMLElement->xpath(expressionXpath)` Cette méthode permet de récupérer une sélection de nœuds d'un élément SimpleXML selon une expression Xpath (non abordée dans cet ouvrage). La liste retournée par la méthode est structurée comme un tableau de variables et peut donc être facilement exploitée à l'aide de la fonction `foreach()`.

Pour lire un document XML contenu dans une variable interne, il suffit d'utiliser la fonction suivante : `simplexml_load_string(nomChaine)`. L'argument de cette fonction doit correspondre au nom de la variable. Si la lecture réussit, la fonction renvoie un objet SimpleXML qui peut être ensuite manipulé par les autres fonctions de la classe SimpleXML (voir l'exemple du code 21-40 ou encore le code 11-5 de l'atelier 11-2), sinon en cas d'erreur la fonction renvoie la valeur `false`.

Code 21-40 : exemple de lecture d'un élément spécifique d'un document XML :

```
<?php
$nomPeres='<ages>
 <pere >
  <nom>Dupond</nom>
  <prenom>Jean</prenom>
  <age>45</age>
 </pere>
 <pere >
  <nom>Durand</nom>
  <prenom>Claude</prenom>
  <age>35</age>
 </pere>
 <pere >
  <nom>Duval</nom>
  <prenom>Thierry</prenom>
  <age>32</age>
 </pere>
</ages>';
$xml = simplexml_load_string($nomPeres);
echo $xml->pere[0]->nom; // Affiche "Dupond"
?>
```

22

MySQL

Le langage SQL (Structured Query Language) est un langage normalisé d'interrogation de bases de données. Puisqu'il est normalisé, il est indépendant du type des bases de données : les mêmes commandes peuvent donc être exploitées quelle que soit la base utilisée (Access, MySQL…). Les commandes SQL peuvent ainsi gérer tout type d'action sur le serveur de bases de données MySQL, depuis la simple manipulation des enregistrements jusqu'à la création, la modification ou la suppression d'une base, d'une table ou d'un champ.

Méthodes d'exécution d'une commande SQL

Les commandes SQL peuvent être transmises au serveur MySQL de deux manières selon l'action qu'on désire réaliser (voir tableau 22-1).

Tableau 22-1 Tableau de choix d'une méthode pour intervenir sur une base de données

Commandes	Actions	Méthode conseillée

`CREATE`
`ALTER`
`DROP`
	Création ou modification d'une base de données ou d'une table	Gestionnaire phpMyAdmin :
		Interface graphique qui permet de générer tout type de commande SQL.
		En pratique, le gestionnaire est utilisé pour créer ou modifier la structure de la base (`CREATE`, `ALTER`, `DROP`) ou pour tester des requêtes (`SELECT`…) en phase de mise au point des scripts PHP (nécessite un compte administrateur).

		Client MySQL : Même utilisation que le gestionnaire phpMyAdmin mais en mode lignes de code. L'utilisation du client MySql nécessite de connaître toutes les syntaxes SQL, contrairement à l'interface graphique présentée ci-dessus (nécessite un compte administrateur).

`SELECT`
`INSERT`
`UPDATE`
`DELETE`
`REPLACE`
	Manipulations des enregistrements (données) des différentes tables d'une base de données	Script PHP d'interfaçage MySql :
		Scripts PHP créés dans un éditeur de code dont la fonction est d'assurer l'interfaçage avec la base de données.
		Ces scripts répondent exactement aux besoins de l'application et peuvent générer tout type de commande SQL mais, en pratique, ils sont surtout utilisés pour gérer les enregistrements des tables (`SELECT`, `INSERT`…).

La première manière consiste à utiliser un logiciel comme le gestionnaire phpMyAdmin ou encore le client mysql (qui permet la saisie de commandes en ligne de code directement sur le serveur). Cependant, ces outils nécessitent de disposer d'un compte administrateur sur le serveur de la base de données et demandent un apprentissage préalable. En pratique, nous allons les utiliser uniquement dans le cadre de la création ou de la modification de la structure d'une base de données.

La deuxième manière consiste à utiliser des scripts PHP dont la fonction est d'assurer l'interfaçage avec la base de données MySql. Ces derniers envoient des requêtes à la base selon les demandes qu'ils réceptionnent (à l'aide d'un formulaire en ligne, par exemple). Les résultats des requêtes sont ensuite mis en forme et transmis à une application spécifique. Cette deuxième méthode permet de créer des interfaces client adaptées aux besoins d'une application et utilisables en ligne sans authentification préalable de l'utilisateur.

Dans ce chapitre, nous nous intéresserons uniquement à la rédaction des requêtes destinées à la manipulation des données (`SELECT`, `INSERT`, `UPDATE`, `DELETE` et `REPLACE`). Nous n'allons pas créer de requêtes pour la création ou la modification de la structure de la base de données car il s'agit du rôle du gestionnaire de base de données phpMyAdmin.

Dans le chapitre 13, consacré aux applications Ajax-PHP-MySql, nous avons étudié les différentes fonctions PHP qui permettent d'intégrer une commande SQL dans un script PHP (revoir aussi la partie du chapitre 21 consacrée à la bibliothèque MySQL de PHP). Dans le présent chapitre, nous nous limiterons à l'étude des commandes SQL.

Le tableau 22-2 présente les différentes commandes SQL destinées à la manipulation de données, leur fonction et leur syntaxe simplifiée. Nous allons compléter plus loin ces informations par une syntaxe détaillée des commandes. Chaque commande est accompagnée de plusieurs exemples.

Tableau 22-2 Principales commandes SQL de manipulation d'enregistrements

Commande SQL	Fonction	Syntaxe simplifiée

`SELECT` (commande utilisée pour tous les jeux d'enregistrements)	Recherche et extraction de données. Différentes options peuvent être exploitées pour sélectionner les enregistrements, les champs retournés, ou l'ordre dans lequel sont retournés les enregistrements.	`SELECT champ1, champ2, …`
		`FROM table1, table2, …`
		`WHERE critère(s) de sélection`
		`ORDER BY information sur le tri`

`INSERT`	Ajout d'enregistrement(s) dans la base de données	`INSERT INTO table (champ1, champ2, …)` `VALUES (valeur1, valeur2, …)`

`UPDATE`	Modification d'enregistrement(s)	`UPDATE table`
		`SET champ = valeur`
		`WHERE critère(s) de sélection`

`DELETE`	Suppression d'enregistrement(s)	`DELETE FROM table` `WHERE critère(s) de sélection`
`REPLACE`	Remplacement d'enregistrement(s) (équivaut aux commandes `DELETE` et `INSERT` exécutées successivement)	`REPLACE FROM table` `WHERE critère(s) de sélection`

Conditions de test des exemples de commande SQL

Installez la base de données machineasous rapidement

Si vous n'avez pas encore réalisé les ateliers du chapitre 13, vous ne disposez pas encore de la base de donnée `machineasous` avec laquelle nous avons réalisé les différents exemples SQL de ce chapitre. Pour que vous puissiez l'installer rapidement sur votre serveur de développement (Wamp), nous avons placé une exportation de cette base dans le répertoire du chapitre 22 des codes source de cet ouvrage (disponibles sur le site `http://www.editions-eyrolles.com`). Pour restaurer cette base, il faut commencer par ouvrir le gestionnaire de base de données phpMyAdmin depuis le manager de Wamp5 et de créer une base portant le nom `machineasous`. Une fois la base créée, il vous suffit d'importer le fichier de sauvegarde `machineasous.sql` en suivant la procédure indiquée à la fin de ce chapitre pour disposer des deux tables `joueurs` et `gains` avec exactement les mêmes données que dans nos exemples.

Tous les exemples de ce chapitre ont été testés sur la base de données `machineasous` et à l'aide du gestionnaire phpMyAdmin. Afin que vous compreniez bien les spécificités de chaque clause, nous vous suggérons de réaliser chacun de ces exemples. Pour cela, ouvrez le gestionnaire phpMyAdmin et sélectionnez dans le menu déroulant du cadre de gauche la base `machineasous` (voir repère 1 de la figure 22-1) que nous avons créée précédemment (revoir si besoin le chapitre 13) puis cliquez sur le nom de la table `joueurs` située en dessous du menu déroulant (voir repère 2 de la figure 22-1). Cliquez ensuite sur l'onglet `SQL` (voir repère 3 de la figure 22-1), saisissez la requête à tester dans le champ `Exécuter une ou des requêtes` (voir repère 4 de la figure 22-1) et cliquez sur le bouton `Exécuter` (voir repère 5 de la figure 22-1). Vous pouvez ainsi vous aider de la liste des champs du tableau de droite en les insérant directement dans votre requête par un simple double clic (cette liste n'apparaît cependant que si vous avez sélectionné au préalable une des tables de la base).

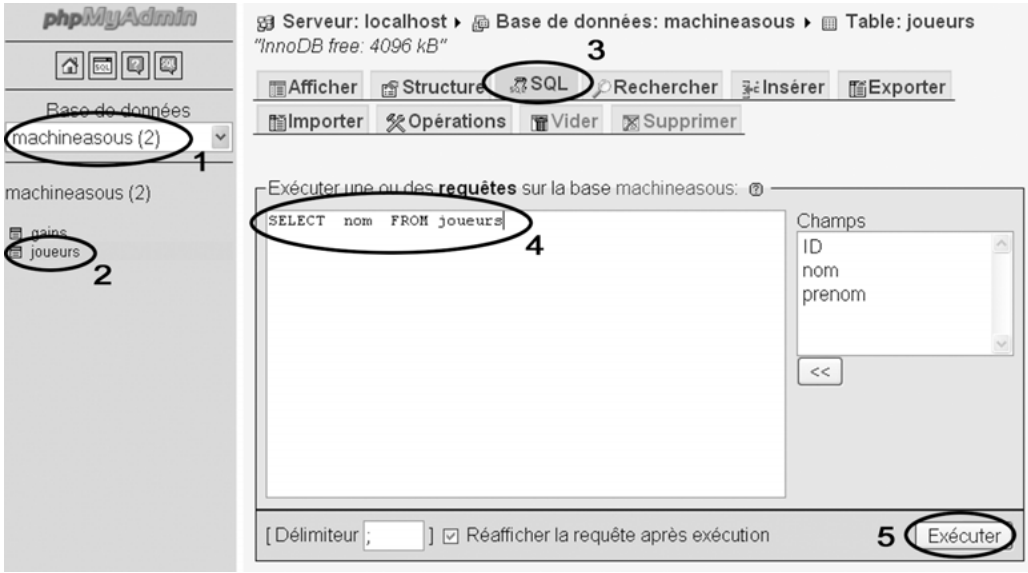


Figure 22-1

Procédure de test d'une requête SQL dans phpMyAdmin

> **À noter**
>
> phpMyAdmin ajoute automatiquement à la requête saisie une clause `LIMIT` (exemple `LIMIT 0,30`) pour limiter le nombre d'enregistrements affichés. Ne soyez donc pas surpris de voir apparaître cette clause dans les différentes illustrations présentées dans ce chapitre.

D'autre part, afin de partir d'une base de test commune et de pouvoir interpréter correctement les résultats des exemples que nous vous présentons dans ce chapitre, nous vous indiquons dans les figures 22-2 et 22-3 l'état des enregistrements des différentes tables de la base `machineasous` qui ont servi à nos essais.

Figure 22-2

Affichage des enregistrements de la table joueurs utilisés dans nos exemples

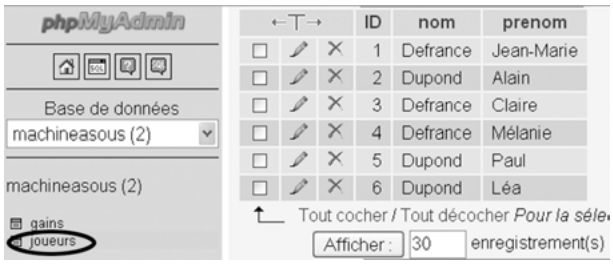
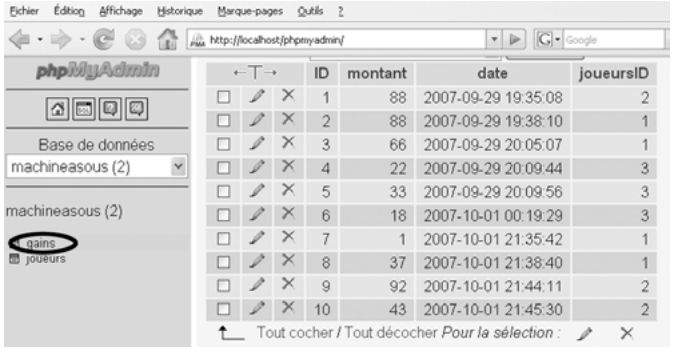


Figure 22-3

Affichage des enregistrements de la table gains utilisés dans nos exemples



Commande SELECT

La commande `SELECT` permet de rechercher puis d'extraire les champs demandés d'une ou plusieurs tables selon un ou plusieurs critères. Les résultats ainsi retournés forment des « jeux d'enregistrements ». Ceux-ci peuvent être triés ou groupés selon les options retenues dans la requête `SELECT`.

`SELECT` est une commande fréquemment utilisée pour des manipulations d'enregistrements courants ; il est donc très important de bien maîtriser toutes ses variantes si vous désirez concevoir des requêtes SQL avancées.

Différentes clauses peuvent compléter la commande `SELECT` afin de préciser l'opération à réaliser. Par exemple, la clause `WHERE` permet de sélectionner les enregistrements à extraire. La clause `ORDER BY` permet quant à elle de trier les résultats après leur sélection.

Nous vous proposons de détailler ces différentes clauses et de les illustrer par des exemples.

Tableau 22-3 Syntaxe de la commande SELET et exemples d'utilisation

	Sélection d'enregistrements

Syntaxe détaillée	`SELECT [DISTINCT | DISTINCTROW]`
	`[table.]champ, … | *`
	`[FROM table, …]`
	`[WHERE expression_de_sélection]`
	`[GROUP BY [table.]champ, …]`
	`[HAVING expression_de_sélection]`
	`[ORDER BY champ [ASC | DESC] ]`
	`[LIMIT [debut,] nb_lignes ]`

Légende	[xxx] : le code xxx est facultatif. xxx | yyy : le code « | » sépare des groupes de code alternatifs (il faut donc choisir de saisir soit xxx, soit yyy). xxx … : la suite du code peut être complétée par des groupes de code de même structure que xxx.

Exemples	Exemple 1 :
	`SELECT * FROM joueurs;`
	Exemple 2 :
	`SELECT nom FROM joueurs WHERE nom='Defrance';`
	Exemple 3 :
	`SELECT joueurs.nom, gains.montant`
	`FROM joueurs, gains`
	`WHERE gains.joueursID=joueurs.ID;`

Commande SELECT simple

Dans sa forme la plus simple, la commande `SELECT` peut être employée sans la clause `WHERE`. Dans ce cas, en l'absence d'expression de sélection, tous les enregistrements de la table sont retournés. Pour inclure uniquement certaines colonnes dans le résultat, il faut les énumérer après la commande `SELECT`. S'il y a plus d'une colonne à retourner, il faut séparer les différents noms des colonnes par une virgule. Enfin, si des champs de même nom issus de tables différentes peuvent être ambigus, il est indispensable de les faire précéder du nom de la table à laquelle ils appartiennent afin de lever l'ambiguïté. Enfin, il est possible de remplacer l'énumération des colonnes désirées par le caractère `*` : toutes les colonnes de la table sont alors retournées dans le résultat de la requête.

Voici deux exemples :

- pour obtenir tous les champs de tous les joueurs :

```
SELECT * FROM joueurs;
```

- pour obtenir uniquement les noms et prénoms de tous les joueurs :

```
SELECT nom, prenom FROM joueurs;
```

Commande SELECT avec des alias

Il est souvent pratique de définir des noms alias différents des noms des champs de la base. On peut s'en servir pour définir des expressions de sélection ou encore lorsqu'on utilise des fonctions, comme nous allons le voir. Pour définir un nom de champ alias, il

suffit de faire suivre l'expression qu'il représente par l'instruction `AS` et par le nom de l'alias désiré.

Par exemple, voici la commande à saisir pour obtenir tous les identifiants des joueurs sous l'alias `id1`, ainsi que leur nom :

```
SELECT joueurs.ID AS id1, nom FROM joueurs;
```

Commande SELECT avec des fonctions MySQL

De nombreuses fonctions MySQL peuvent être utilisées dans les requêtes. Elles se substituent au nom d'un champ juste après la commande `SELECT`. Ces fonctions permettent, entre autres, de réaliser des calculs mathématiques ou des concaténations. Elles servent aussi à préparer une date (enregistrée dans la base au format MySQL : `AAAA-MM-JJ`) afin qu'elle soit retournée et affichée au format français (`JJ-MM-AAAA`). Le tableau 22-4 propose une liste non exhaustive des fonctions MySQL disponibles.

Tableau 22-4 Liste non exhaustive des principales fonctions MySQL

Fonction	Description
`ABS(nbr1)`	Renvoie la valeur absolue du nombre `nbr1`.
`ASCII(car1)`	Renvoie le code ASCII du caractère `car1`.
`CONCAT(elem1, [elem2, …])`	Renvoie la concaténation de tous les éléments, un élément pouvant être un champ, un nombre ou un caractère. Si c'est un caractère, il convient de l'encadrer avec des guillemets simples.
`COUNT(ch1)`	Renvoie le nombre d'enregistrements non nuls du champ `ch1`. À noter : si on utilise `COUNT(*)`, on obtient le nombre total d'enregistrements de la table, quel que soit le champ.
`CURDATE()`	Renvoie la date courante au format `AAAA-MM-JJ`.
`CURTIME()`	Renvoie l'heure courante au format `HH:MM:SS`.
`HEX(nbr1)`	Renvoie la valeur hexadécimale du nombre `nbr1`.
`IFNULL(elem1,elem2)`	Renvoie elem1 s'il est NULL ; sinon renvoie `elem2`.
`LAST_INSERT_ID()`	Renvoie la dernière valeur créée pour un champ auto-incrémenté.
`MAX(ch1)`	Renvoie la plus grande des valeurs du champ `ch1`.
`MIN(ch1)`	Renvoie la plus petite des valeurs du champ `ch1`.
`NOW()`	Renvoie la date et l'heure courantes.
`SIGN(elem1)`	Renvoie le signe de `elem1`.
`SUM(ch1)`	Renvoie la somme des valeurs du champ `ch1`.

Lorsqu'on utilise des fonctions MySQL, il est très pratique de définir un nom de champ alias pour exploiter le résultat de la fonction à partir du jeu d'enregistrements.

Dans l'exemple qui suit, on désire obtenir les montants des gains suivis de « euros ». Pour manipuler plus facilement les données ainsi créées, on définit un alias nommé `montantEuro` pour représenter ce nouveau champ dans le jeu d'enregistrements :

```
SELECT CONCAT(montant,' euros') AS montantEuro FROM gains;
```

Commande SELECT avec la clause DISTINCT

Si, dans les résultats sélectionnés, deux enregistrements sont identiques, la clause `DISTINCT` permet de ne retourner dans le jeu d'enregistrements qu'un seul des enregistrements.

Dans les exemples ci-dessous, nous posons l'hypothèse que deux enregistrements identiques (Defrance Jean-Marie) ont été créés dans la table `joueurs` (pour faire un test, il suffit, par exemple, de modifier au préalable le prénom du joueur Defrance Claire).

Si on n'utilise pas la clause `DISTINCT`, voici ce qu'on obtient :

```
SELECT nom, prenom FROM joueurs;
```

Tableau 22-5 Jeu d'enregistrements obtenu sans la clause DISTINCT

nom	prenom
Defrance	Jean-Marie
Defrance	Jean-Marie
Defrance	Mélanie
Dupond	Alain
…	…

Avec la clause `DISTINCT`, le résultat devient le suivant :

```
SELECT DISTINCT nom, prenom FROM joueurs;
```

Tableau 22-6 Jeu d'enregistrements obtenu avec la clause DISTINCT

nom	prenom
Defrance	Jean-Marie
Defrance	Mélanie
Dupond	Alain
…	…

Commande SELECT avec la clause WHERE

La clause `WHERE` permet d'introduire l'expression de sélection à laquelle doit répondre le résultat retourné. Plusieurs types d'opérateurs peuvent être utilisés pour définir l'expression de sélection.

Expressions de sélection avec des opérateurs de comparaison

On utilise des opérateurs de comparaison pour définir la condition mathématique à vérifier pour que l'enregistrement soit sélectionné. Vous pouvez comparer deux champs de la base (cas des jointures), un champ et un nombre ou un champ et une chaîne de caractères (dans ce cas, la chaîne de caractères doit être encadrée par des guillemets simples « ' »)

Dans l'exemple qui suit, nous désirons obtenir une sélection des montants des gains supérieurs à 50 euros. Les colonnes renvoyées dans le jeu d'enregistrements sont les ID des joueurs et le montant de leur gain :

Tableau 22-7 Liste des opérateurs de comparaison qui peuvent être utilisés dans une requête SQL

Opérateur	Fonction
=	Égal
>	Supérieur
>=	Supérieur ou égal
<	Inférieur
<=	Inférieur ou égal
<>	Différent

```
SELECT joueursID, montant FROM gains WHERE montant>50;
```

Dans ce deuxième exemple, nous désirons obtenir toutes les colonnes de l'enregistrement correspondant au joueur dont l'identifiant est égal à `2` (attention, si la valeur utilisée pour la sélection est de type texte, il faut alors l'encadrer par des guillemets (`'`)) :

```
SELECT * FROM joueurs WHERE ID=2;
```

Dans ce troisième exemple, nous désirons obtenir le nom et prénom du joueur dont l'identifiant est égal à 2 (dans notre exemple, il s'agit du joueur `Dupond Alain`), ainsi que ses différents gains. Comme les informations sont placées dans deux tables différentes, il faut faire une jointure entre les deux tables (voir la section consacrée à la commande `SELECT` avec jointure dans ce même chapitre) pour récupérer les informations correspondantes (`gains.joueursID=joueurs.ID`). Si des champs de tables différentes sont utilisés dans la requête, les noms des tables concernées doivent être ajoutés après l'instruction `FROM` :

```
SELECT joueurs.nom, joueurs.prenom, gains.montant FROM joueurs, gains
WHERE gains.joueursID=joueurs.ID AND joueurs.ID=2;
```

Expressions de sélection avec des opérateurs logiques

Lorsque les critères de sélection sont multiples, l'expression de sélection finale doit être composée des différentes expressions de sélection, reliées entre elles par des opérateurs logiques. Plusieurs opérateurs logiques peuvent être utilisés selon le lien désiré entre les expressions. Vous pouvez trouver ci-après la liste des opérateurs logiques utilisables dans une requête MySQL.

Tableau 22-8 Liste des opérateurs logiques qui peuvent être utilisés dans une requête SQL

Opérateur	Fonction
`AND`	Les expressions reliées entre elles par `AND` doivent toutes être vérifiées (`VRAIES` ou `TRUE`).
`OR`	Au moins l'une des expressions reliées entre elles par `OR` doit être vérifiée (`VRAIE` ou `TRUE`).
`NOT`	L'expression précédée par `NOT` ne doit pas être vérifiée.
Si vous utilisez plusieurs opérateurs logiques, il faut utiliser des parenthèses pour définir la structure de l'expression.	

Dans l'exemple suivant, nous désirons obtenir les noms et prénoms des joueurs de la famille Defrance ayant gagné plus de 50 euros en un seul jeu. Comme il s'agit ici encore d'une requête multi-tables, nous avons ajouté dans la clause `WHERE` une troisième

condition pour effectuer la jointure entre les deux tables concernées (`gains.joueursID=joueurs.ID`) :

```
SELECT DISTINCT joueurs.nom, joueurs.prenom FROM joueurs, gains WHERE joueurs
➥.nom='Defrance' AND gains.montant>20 AND gains.joueursID=joueurs.ID;
```

Expressions de sélection avec des opérateurs de recherche

On utilise des opérateurs de recherche pour définir une condition spécifique (qui n'est ni logique ni comparative) à vérifier pour que l'enregistrement soit sélectionné. Il existe plusieurs types d'opérateurs de recherche selon la condition désirée. Le tableau 22-9 liste les opérateurs de recherche les plus fréquents.

Tableau 22-9 Liste des opérateurs de recherche qui peuvent être utilisés dans une requête SQL

Opérateur	Fonction
`LIKE`	Permet de sélectionner un champ dont la valeur commence par, finit par, ou contient une chaîne de caractères (`%` et _ accompagnent souvent `LIKE` pour définir les caractères de substitution dans la chaîne).
`BETWEEN`	Permet de sélectionner un champ dont la valeur est comprise dans une plage de valeurs.
`IN`	Permet de sélectionner un champ dont la valeur appartient à une liste de valeurs.
`IS NULL`	Permet de sélectionner un champ dont la valeur est `NULL`.
`IS NOT NULL`	Permet de sélectionner un champ dont la valeur n'est pas `NULL`.

Dans l'exemple suivant, nous désirons obtenir la liste des gains dont le montant est compris entre 20 et 50 euros :

```
SELECT montant FROM gains WHERE montant BETWEEN 20 AND 50;
```

Tableau 22-10 Caractères de substitution qui peuvent être utilisés dans une requête SQL

Caractère	Utilisation
`_` (caractère de soulignement)	Remplace un caractère quelconque.
`%`	Remplace aucun ou plusieurs caractère(s) quelconque(s).

Dans ce deuxième exemple, nous désirons obtenir tous les enregistrements des joueurs dont le prénom contient un « `n` » :

```
SELECT * FROM joueurs WHERE prenom LIKE '%n%';
```

Commande SELECT avec la clause ORDER BY

Dans les différents exemples que nous vous avons proposés jusqu'à présent, les enregistrements étaient retournés dans l'ordre de la saisie initiale (en général, l'ordre de la clé primaire `ID` auto-incrémentée). Si vous souhaitez présenter les enregistrements dans un ordre différent, utilisez la clause `ORDER BY` en précisant le ou les champs suivant le(s)quel(s) le tri s'effectue. Avec `ASC`, vous triez par ordre croissant, avec `DESC`, par ordre décroissant. Par défaut, l'ordre est croissant.

Dans l'exemple suivant, nous désirons trier par ordre alphabétique tous les prénoms des joueurs de la famille Defrance :

```
SELECT prenom FROM joueurs WHERE nom='Defrance' ORDER BY prenom;
```

Dans ce deuxième exemple, nous désirons trier tous les joueurs selon l'ordre alphabétique de leur nom, puis de leur prénom.

```
SELECT nom, prenom FROM joueurs ORDER BY nom, prenom;
```

Commande SELECT avec la clause LIMIT

La clause `LIMIT` indique le nombre maximal d'enregistrements pour le résultat retourné. Elle est toujours placée à la fin de la requête. Si vous faites suivre cette clause d'un seul chiffre, la limite s'applique à partir de la première ligne ; si vous indiquez deux chiffres séparés par une virgule, le premier indique le numéro de ligne à partir de laquelle la limite précisée par le deuxième chiffre s'applique.

> **À noter**
> Cette clause est spécifique à MySQL et ne fait pas partie du standard SQL.

Dans l'exemple ci-dessous, nous désirons obtenir une sélection des joueurs identique à celle de l'exercice précédent, mais limitée aux deux premiers enregistrements :

```
SELECT nom, prenom FROM joueurs ORDER BY nom, prenom LIMIT 2;
```

Dans ce deuxième exemple, nous désirons obtenir une sélection des joueurs identique à celle de l'exercice précédent, mais limitée aux deux enregistrements situés après la troisième ligne des résultats de la sélection initiale :

```
SELECT nom, prenom FROM joueurs ORDER BY nom, prenom LIMIT 3,2;
```

Commande SELECT avec jointure

La jointure permet de créer des requêtes portant sur des données réparties dans plusieurs tables. Pour réaliser une jointure, on utilise la même syntaxe que pour une requête traditionnelle, mais en indiquant dans la clause `FROM` la liste des tables concernées, séparées par des virgules et en ajoutant dans la clause `WHERE` l'expression de sélection qui permet le rapprochement entre les tables. En général, l'expression de sélection qui permet le rapprochement entre tables s'exprime par une égalité entre la clé primaire d'une table et la clé étrangère de l'autre (par exemple : entre `ID` de la table `joueurs` et `joueursID` de la table `gains`). Il faut mentionner le nom de la table en préfixe des noms de champs pour éviter toute ambiguïté, notamment entre deux champs portant le même nom mais intégrés dans des tables différentes. Afin d'éviter de rappeler dans son intégralité le nom de chaque table en préfixe des noms de champs, il est intéressant de créer un alias (exemple : dans la clause `FROM`, si on déclare `joueurs AS j`, on peut utiliser l'appellation `j.nom` dans la clause `WHERE`).

Dans l'exemple ci-dessous, nous désirons obtenir les différents gains classés du plus important au plus faible (issus de la table `gains`), avec les noms et prénoms des joueurs correspondants (issus de la table `joueurs`) :

```
SELECT gains.montant, joueurs.nom, joueurs.prenom FROM joueurs, gains WHERE gains
➥.joueursID=joueurs.ID ORDER BY gains.montant DESC;
```

Commande INSERT

La commande `INSERT` permet d'insérer de nouveaux enregistrements dans la base et peut être utilisée de différentes manières. Pour ajouter des valeurs dans une table à partir de variables ou de constantes récupérées par un script, par exemple, il existe deux méthodes différentes : `INTO VALUES` et `INTO SET`. Nous verrons aussi comment utiliser la commande `INSERT` pour insérer des enregistrements à partir d'une requête (`INTO SELECT`). Cette dernière méthode est très intéressante pour réaliser une copie d'enregistrements d'une table vers une autre.

Lors de l'insertion directe à partir de valeurs, les textes doivent être encadrés entre guillemets simples alors que les nombres peuvent s'en passer (exemple : `'Chapelier'` ou `1980`).

> **À noter**
>
> Dans les scripts PHP, il est également possible d'indiquer des expressions dans une requête (des variables, par exemple : `'$var'`). Dans ce cas, elles sont évaluées avant d'être insérées dans la base.

Tableau 22-11 Syntaxe de la commande INSERT et exemples d'utilisation

Fonction	Insertion d'enregistrements

Syntaxe de la commande d'insertion à partir de valeurs (première méthode)	`INSERT INTO`
	`table`
	`[ (champ1, champ2, …, champN) ]`
	`VALUES`
	`(valeur1, valeur2, …, valeurN);`

Syntaxe de la commande d'insertion à partir de valeurs (deuxième méthode)	`INSERT INTO`
	`table`
	`SET`
	`champ1=valeur1, champ2=valeur2, …, champN=valeurN;`

Syntaxe de la commande d'insertion à partir d'une autre table.	`INSERT INTO`
	`table_cible`
	`SELECT`
	`* | valeur1, valeur2, …, valeurN`
	`FROM table_source`
	`[WHERE expression_de_sélection]`

Légende	`table` : table dans laquelle sont insérées les données. `champN` : nom du champ de la table dans lequel est insérée `valeurN`. `valeurN` : la valeur doit respecter le format standard de son type (exemple : `'bonjour'` ou `4562` ou `'2003-03-24'`). À noter : si la requête est intégrée dans un script PHP, la valeur peut être remplacée par une variable qui est évaluée au moment de l'insertion (exemple : `'$var'`). `xxx | yyy` : le code `|` sépare des groupes alternatifs, il faut donc choisir de saisir soit `xxx`, soit `yyy`. `table_source` : dans le cas du transfert d'une table source vers une table cible, le symbole * peut être utilisé, mais il faut veiller à ce que le nombre de champs des deux tables soit identique.

Commande INSERT à partir de valeurs : méthode 1

Dans sa première syntaxe, la commande `INSERT` permet d'énumérer partiellement les champs à insérer dans la base. Cela peut être intéressant dans le cas d'un premier enregistrement partiel, mais il faut alors s'assurer que les champs omis sont paramétrés pour être facultatifs (attribut `null=null`) ou auto-incrémentés. Dans le cas où l'énumération des champs n'est pas indiquée, il faut s'assurer que l'ordre des valeurs respecte celui des champs de la table. Sachez cependant que, même si l'énumération est facultative, il est fortement conseillé de toujours spécifier la liste des champs dans lesquels les valeurs doivent s'insérer. En effet, si la structure de la table change, votre requête risque de ne plus fonctionner.

Dans l'exemple ci-dessous, nous désirons ajouter un joueur supplémentaire à la table `joueurs`. Pour cela, nous allons utiliser la syntaxe complète de la commande `INSERT`. Dans ce cas, mentionnez uniquement les champs qui reçoivent une valeur (`nom, prenom`). La clé `ID` n'étant mentionnée ni dans les champs ni dans les valeurs, MySQL lui attribue une valeur par incrémentation du dernier `ID` saisi.

```
INSERT INTO joueurs (nom,prenom) VALUES ('Audoux','Thierry');
```

Commande INSERT à partir de valeurs : méthode 2

Dans la deuxième syntaxe de `INSERT`, les couples champ/valeur doivent être spécifiés et séparés par un signe égal (`champ1='valeur1'`). La clause utilisée n'est plus `VALUES` mais `SET`.

Dans l'exemple qui suit, nous allons ajouter un nouveau joueur, comme dans l'exemple précédent :

```
INSERT INTO joueurs SET nom='Audoux', prenom='Thierry';
```

Commande INSERT à partir d'une requête

Avec ce type de syntaxe, il est possible d'insérer le résultat d'une requête dans une table. Dans ce cas, la clause `VALUES` est remplacée par la requête à insérer. Cette syntaxe est particulièrement intéressante pour copier des données d'une table à l'autre ou encore pour réaliser la projection d'une table : on prélève certains de ses champs pour créer une autre table de plus petite taille.

Dans ce premier exemple, on désire transférer le contenu de la table `joueurs` dans une table de sauvegarde `joueursbackup`. Il faut évidemment que le nombre et les types des champs soient rigoureusement identiques dans les deux tables (pour créer une structure identique, utilisez le formulaire de copie d'une table disponible dans l'onglet Opérations de phpMyAdmin, voir figure 22-4) :

```
INSERT INTO joueursbackup SELECT * FROM joueurs;
```

Dans ce deuxième exemple, on désire faire une projection de la table `joueurs` dans une autre table `tabledesnoms`. Cette table contient uniquement les noms des différents joueurs dans un champ `nom` avec une clé primaire auto-incrémentée `ID` et doit être créée au préalable :

```
INSERT INTO tabledesnoms (nom) SELECT nom FROM joueurs;
```

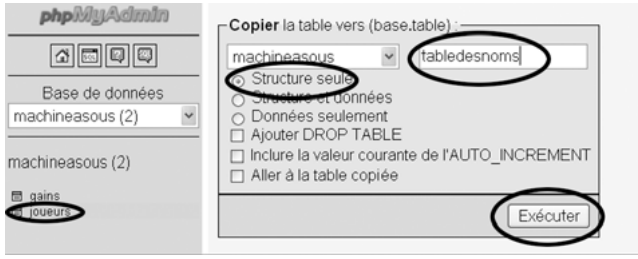


Figure 22-4

Avant de réaliser la sauvegarde d'une table à l'aide de la commande INSERT, il faut créer une structure de table identique à celle de la table à transférer. Pour cela, cliquez sur le nom de la table joueurs dans la partie gauche de phpMyAdmin, puis sur l'onglet Opérations en haut de l'écran. Dans le champ « Copier la table vers », saisissez le nom de la nouvelle table, cochez l'option « Structure seule » et cliquez sur le bouton Exécuter.

Commande DELETE

La commande `DELETE` permet de supprimer un enregistrement d'une table. Cette opération est irréversible et il vaut mieux prévoir l'affichage d'un message d'avertissement avant validation définitive de la requête. La commande `DELETE` doit être accompagnée de la clause `WHERE` (sinon vous supprimez tous les enregistrements de la table), suivie d'une expression de sélection d'enregistrements à supprimer. Cette même expression peut d'ailleurs être utilisée au préalable dans une requête `SELECT` pour sélectionner et afficher les enregistrements à supprimer avec la requête `DELETE` lors de l'étape suivante :

```
SELECT * FROM joueurs WHERE ID=2;
DELETE FROM joueurs WHERE ID=2;
```

Tableau 22-12 Syntaxe de la commande DELETE et exemple d'utilisation

Fonction	Suppression d'enregistrements

Syntaxe de la commande de suppression	`DELETE FROM`
	`table`
	`[WHERE expression_de_sélection]`
	`[LIMIT [debut,] nb_lignes ]`

Légende	`table` : table où se trouvent les enregistrements sélectionnés. [xxx] : le code xxx est facultatif. Attention ! Les crochets [et] ne doivent surtout pas être saisis dans le code.

Dans l'exemple ci-dessous, on désire supprimer tous les enregistrements correspondant aux gains inférieurs à 10 euros :

```
DELETE FROM gains WHERE montant<10;
```

Commande UPDATE

La commande `UPDATE` permet de modifier la valeur de certains champs si l'expression de sélection est validée. L'affectation des valeurs est introduite par la clause `SET`, comme pour la commande `INSERT`.

Tableau 22-13 Syntaxe de la commande UPDATE et exemple d'utilisation

Fonction	Mise à jour d'enregistrements

Syntaxe de la commande de modification	`UPDATE`
	`table`
	`SET`
	`chap1=valeur1, champ2=valeur2, …, champN=valeurN`
	`[WHERE expression_de_sélection]`
	`[LIMIT [debut,] nb_lignes ]`

Légende	`table` : table où se trouvent les enregistrements sélectionnés. [xxx] : le code `xxx` est facultatif. Attention ! Les crochets `[` et `]` ne doivent surtout pas être saisis dans le code.

Dans l'exemple ci-dessous, nous désirons le prénom du joueur dont l'identifiant est 2 :

```
UPDATE joueurs SET prenom='Stéphane' WHERE ID=2;
```

Commande REPLACE

La commande `REPLACE` permet de remplacer un enregistrement existant et donc de modifier les valeurs de ses différents champs. Cette commande est différente de la commande `UPDATE` car elle supprime d'abord l'enregistrement sélectionné pour ensuite le réinsérer avec de nouvelles valeurs. C'est en quelque sorte une combinaison des commandes `DELETE` et `INSERT`. Comme `INSERT`, cette commande peut être réalisée selon trois variantes. Les deux premières variantes permettent de remplacer les champs d'un enregistrement selon les informations transmises par des valeurs, alors que la troisième permet d'exploiter une requête SQL pour fournir les nouvelles données. La clause `WHERE` sélectionne le jeu d'enregistrements à utiliser pour cette commande.

Tableau 22-14 Syntaxe de la commande UPDATE et exemples d'utilisation

Fonction	Remplacement d'enregistrements

Syntaxe de la commande de remplacement à partir de valeurs (première méthode)	`REPLACE INTO`
	`table`
	`[ (champ1, champ2, …, champN) ]`
	`VALUES`
	`(valeur1, valeur2, …, valeurN)`

Syntaxe de la commande de remplacement à partir de valeurs (deuxième méthode)	`REPLACE INTO`
	`table`
	`SET`
	`chap1=valeur1, champ2=valeur2, …, champN=valeurN`

Syntaxe de la commande de remplacement à partir d'une requête	`REPLACE INTO`
	`table_cible`
	`SELECT`
	`* | valeur1, valeur2, …, valeurN`
	`FROM table_source`
	`[WHERE expression_de_sélection]`

Tableau 22-14 Syntaxe de la commande UPDATE et exemples d'utilisation *(suite)*

Fonction	Remplacement d'enregistrements
Légende	`table` : table dans laquelle les données sont modifiées. `champN` : nom du champ de la table dans lequel est insérée la `valeurN`. `valeurN` : la valeur doit respecter le format standard de son type (exemple : `'bonjour'` ou `4562` ou `'2003-03-24'`). À noter : si la requête est intégrée dans un script PHP, la valeur peut être remplacée par une variable qui sera est évaluée au moment de l'insertion (exemple : `'$var'`). `xxx | yyy` : le caractère | sépare des groupes alternatifs (il faut donc choisir de saisir soit `xxx`, soit `yyy`). `table_source` : lorsqu'un jeu d'enregistrements issu d'une table source est utilisé pour mettre à jour les champs d'une table cible, le symbole * peut être employé, mais il faut veiller à ce que le nombre de champs soit identique dans les deux tables.

Dans l'exemple ci-dessous, nous désirons remplacer le joueur dont l'identifiant est 5 par Thierry Audoux :

```
REPLACE INTO joueurs VALUES(5,'Audoux','Thierry');
```

Configuration des droits d'un utilisateur

Dans les ateliers du chapitre 13, le fichier de connexion à la base de données MySQL a été configuré avec l'utilisateur `root`. Par défaut, cet utilisateur `root` n'a pas de mot de passe et a tous les droits sur toutes les bases.

Évidemment, si vous désirez mettre en ligne votre application, il ne faut pas utiliser cet utilisateur par défaut pour des raisons de sécurité évidente et créer un utilisateur spécifique à votre application avec un mot de passe sécurisé et des droits restreints à la base de votre application.

En général, les paramètres de cet utilisateur sont imposés par votre hébergeur mais il est judicieux dans ce cas de configurer les mêmes paramètres sur votre base de développement afin d'éviter de modifier le fichier de connexion MySQL à chaque mise à jour. Pour vous accompagner dans la création de ce nouvel utilisateur, nous vous proposons de créer un compte utilisateur `machine` pour accéder exclusivement à la base `machineasous`.

Le compte root par défaut

Si vous ne créez pas de compte utilisateur, vous pouvez quand même configurer une connexion à la base en utilisant le compte `root` préconfiguré par défaut dans MySQL (dans ce cas, il faut remplacer dans le fichier de connexion le nom de l'utilisateur par `root` et ne pas indiquer de mot de passe). Attention ! L'usage de ce compte root sans mot de passe est évidemment limité à un usage local. Vous devez impérativement vous assurer que tous les comptes mysql possèdent bien un mot de passe si votre base de données doit être reliée à Internet.

Placez-vous dans la page d'accueil du gestionnaire (cliquez sur l'icône représentant une petite maison correspondant au lien Accueil en haut de la partie gauche de l'interface) puis cliquez sur le lien Privilèges dans la liste des actions de la partie droite de l'interface. Dans la nouvelle fenêtre cliquez sur le lien Ajouter un utilisateur en bas du tableau des utilisateurs (voir figure 22-5).

Figure 22-5

Cet écran affiche les utilisateurs existants de votre base de données. En cliquant sur le lien Créer un utilisateur, vous pouvez accéder au formulaire d'ajout d'un nouvel utilisateur.

Saisissez le nom d'utilisateur `machine` et sélectionnez `Local` (`localhost`) dans le menu déroulant Serveur (voir figure 22-6). Saisissez ensuite deux fois le mot de passe correspondant à cet utilisateur (vous pouvez aussi utiliser le générateur de mot de passe si celui-ci n'est pas imposé par votre hébergeur) et cliquez sur le bouton Exécuter (situé tout en bas de cet écran) sans valider d'autres options de cette page. À noter que si vous validez un droit à ce niveau (privilèges globaux), cela permettrait à l'utilisateur de l'exploiter sur toutes les bases du serveur MySQL et non exclusivement sur la base `machineasous`.

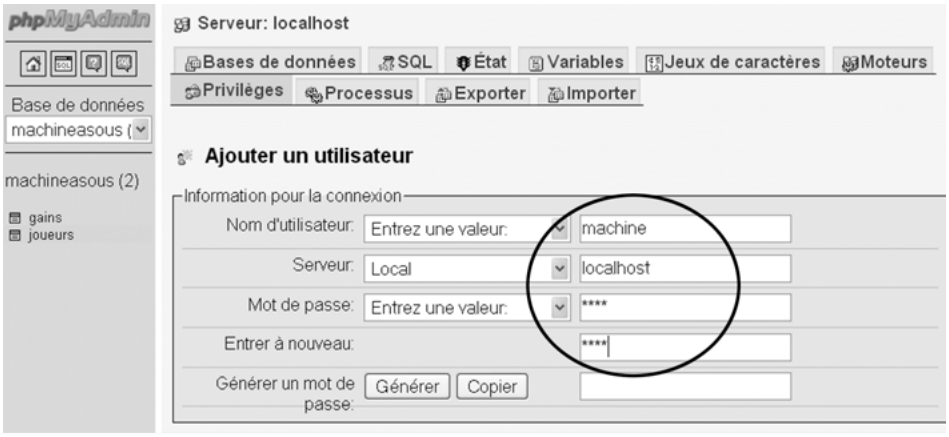


Figure 22-6

Pour ajouter un nouvel utilisateur, il suffit d'indiquer son nom, son serveur (en général localhost) et son mot de passe.

Après validation, un écran vous informe que le nouvel utilisateur `machine@localhost` (c'est-à-dire l'utilisateur `machine` depuis un accès `localhost`) a bien été créé et vous propose de modifier éventuellement ses attributions. Un peu plus bas, dans le même

écran de confirmation, se trouve une rubrique intitulée Privilèges spécifiques à une base de données. Sélectionnez la base `machineasous` dans le menu déroulant (voir figure 22-7) pour accéder au formulaire d'ajout d'un privilège d'accès à la base `machineasous`.

Figure 22-7

Après l'enregistrement du nouvel utilisateur, le gestionnaire affiche un écran de confirmation qui indique que l'utilisateur machine@localhost a bien été ajouté à la table users. En bas de ce même écran, un menu déroulant permet de sélectionner une base de données existante pour attribuer des privilèges spécifiques à l'utilisateur machine.

Cochez les droits que vous désirez attribuer à l'utilisateur pour la base concernée (par exemple, autorisez tous les droits relatifs aux données et à la structure mais pas à l'administration) puis validez en cliquant sur le bouton Exécuter (voir figure 22-8).

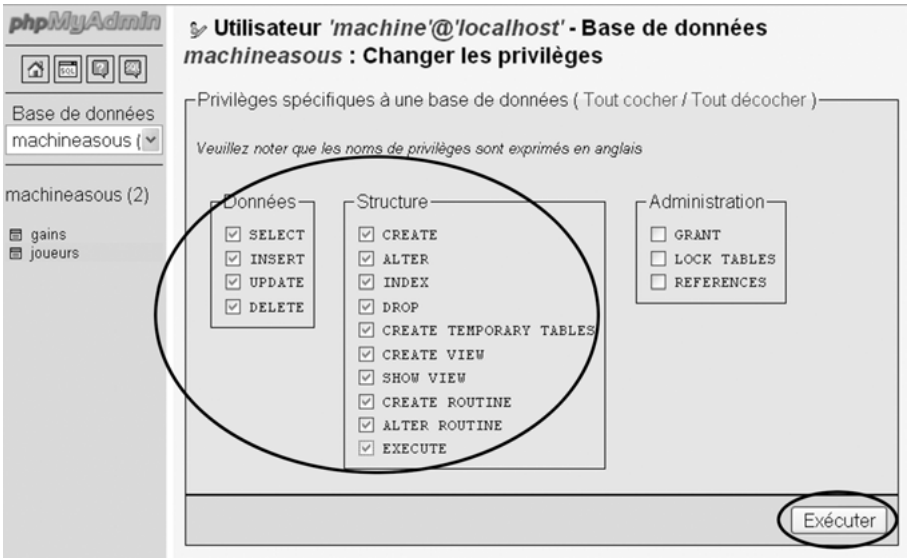


Figure 22-8

Le tableau des privilèges spécifiques à une base de données permet de définir un droit d'accès à une base spécifique.

Les droits de l'utilisateur `machine` sont désormais configurés pour accéder exclusivement à la base `machineasous`. Si vous consultez ensuite la vue d'ensemble des utilisateurs (cliquez sur l'onglet Privilèges en haut de l'écran), vous constatez qu'un nouvel utilisateur `machine` s'affiche (voir figure 22-9).

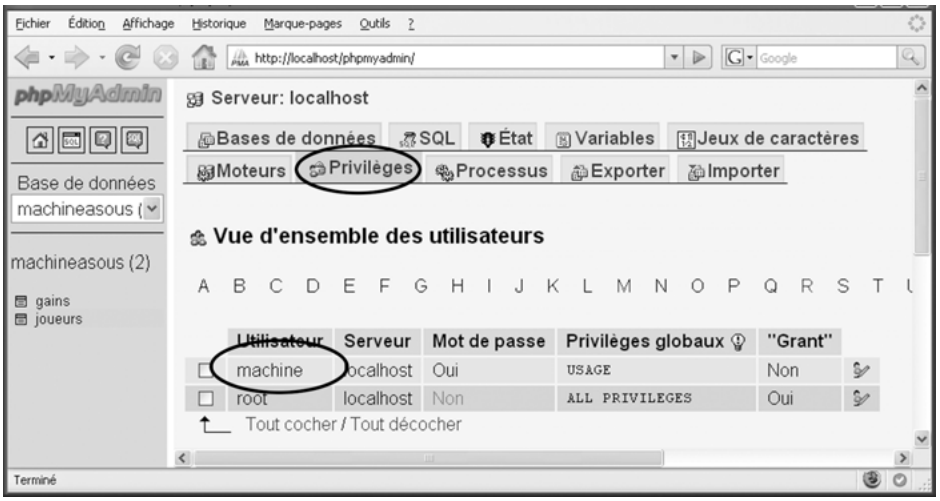


Figure 22-9

Dans la vue d'ensemble des utilisateurs, on constate que l'utilisateur machine est désormais configuré.

Sauvegarde et restauration d'une base de données

Il est hautement recommandé de faire une sauvegarde de secours de ses programmes et il en est de même pour les bases de données. Cependant, la démarche est quelque peu différente, car nous n'allons pas copier un simple fichier, mais enregistrer les requêtes MySQL qui ont été utilisées pour créer la structure de la base et éventuellement celles qui ont permis d'insérer des enregistrements dans les tables. Une fois enregistrées dans un fichier, ces requêtes peuvent ensuite être utilisées dans phpMyAdmin pour recréer à l'identique la base sauvegardée (voir la procédure de restauration).

Sauvegarde

Passons maintenant à la pratique. Pour cela, vous allez commencer par vous assurer que la base à sauvegarder est bien sélectionnée dans la liste déroulante de gauche (voir repère 1 de la figure 22-10). Dans la partie droite, cliquez sur l'onglet Exporter situé en haut de l'écran (voir repère 2 de figure 22-10). La nouvelle page de droite contient plusieurs cadres. Le premier, intitulé Exporter, permet de sélectionner les tables à exporter et le format d'exportation. Sélectionnez toutes les tables en cliquant sur le lien Tout sélectionner (voir repère 3 de la figure 22-10) et conservez le format SQL initialisé par défaut. Le second cadre, intitulé options SQL, vous permet d'ajouter un `DROP TABLE` (pour ce faire, cochez l'option portant le même nom dans le cadre secondaire nommé Structure : voir repère 4 de la figure 22-10) qui supprimera automatiquement les anciennes tables de la base avant d'y inclure les nouvelles, ce qui évite de générer un message d'erreur si une table de même nom existait déjà. C'est dans ce même cadre que l'on peut choisir d'exporter la structure, les données ou les deux ensembles (dans notre cas, nous validerons les deux). Le troisième cadre, intitulé Transmettre, vous permet d'indiquer que vous désirez générer un fichier (pour ce faire, cliquez dans la case à cocher intitulée Transmettre : voir repère 5 de la figure 22-10) et de choisir le type de compression à utiliser (choisissez l'option Aucune).

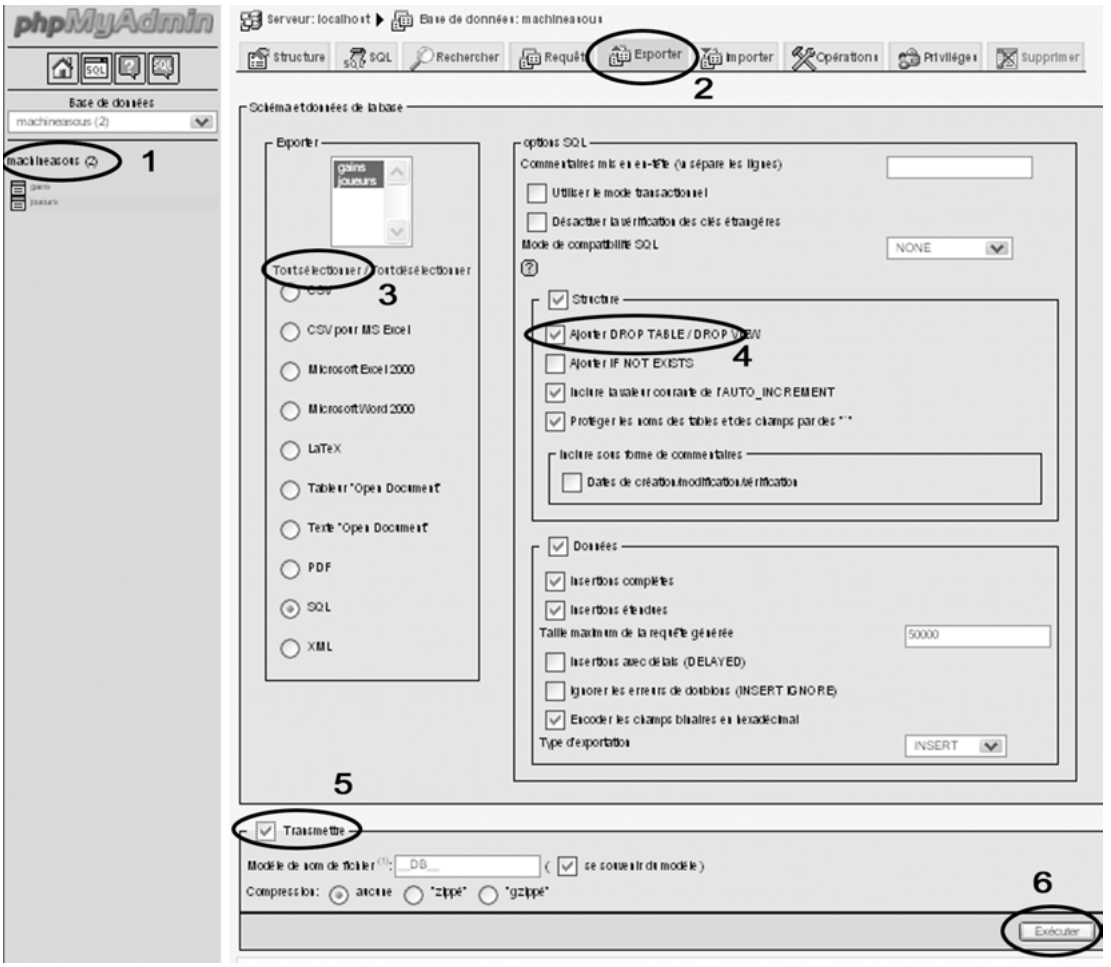


Figure 22-10

Pour sauvegarder une base, on utilise le formulaire de la rubrique Exporter.

Validez en cliquant sur le bouton Exécuter placé en bas de l'écran (voir repère 6 de la figure 22-10). Une première fenêtre vous demande de confirmer l'enregistrement (cliquez sur Enregistrer pour confirmer). Un deuxième écran vous demande de sélectionner le répertoire de sauvegarde. Vérifiez l'emplacement du répertoire qui vous est proposé par défaut ou choisissez un répertoire dédié aux archives de votre projet, afin de savoir où retrouver votre fichier lors de la restauration (vous pouvez par exemple créer un répertoire archives à la racine de votre site Web et créer dans celui-ci un répertoire « sql » qui regroupe toutes les sauvegardes de votre base de données). Après validation, l'enregistrement s'effectue. Nous vous suggérons d'utiliser l'explorateur Windows pour vous assurer que le fichier est bien enregistré à l'endroit indiqué. Si vous ouvrez ce fichier `machineasous.sql` avec un simple éditeur (Bloc-notes, par exemple), vous devez retrouver les requêtes destinées à recréer des structures de tables conformes à l'origine (`CREATE TABLE`), puis à provoquer des ajouts d'enregistrements dans les tables (`INSERT INTO`) selon les valeurs actuellement présentes dans la base.

Restauration

Pour restaurer une base, il faut que celle-ci soit déjà créée (revoir si besoin la procédure de création d'une base dans le chapitre 13). Ensuite, sélectionnez-la dans la liste de gauche et cliquez sur le nom d'une des tables de cette base.

> **À noter**
>
> La base `machineasous` doit être créée, mais peut être vide de toute table. Si ce n'est pas le cas, nous vous invitons à supprimer les tables pour bien comprendre la procédure de restauration (pour supprimer une table, cliquez sur le lien Supprimer correspondant).

Dans la partie droite, cliquez sur l'onglet Importer en haut de l'écran (voir repère 1 de la figure 22-11). La nouvelle page propose un formulaire vous invitant à sélectionner le fichier à importer. Cliquez sur le bouton Parcourir (voir repère 2 de la figure 22-11). Sélectionnez ensuite, dans l'explorateur, le fichier précédemment sauvegardé et cliquez sur Ouvrir pour valider votre choix. Le chemin du fichier est alors copié dans le champ à gauche du bouton Parcourir (voir repère 3 de la figure 22-11). Il ne vous reste plus qu'à cliquer sur le bouton Exécuter pour démarrer la restauration. Toutes les requêtes du fichier s'exécutent alors et s'affichent en haut de l'écran. Au terme de la restauration, les tables de la base sont de nouveau visibles dans le gestionnaire.

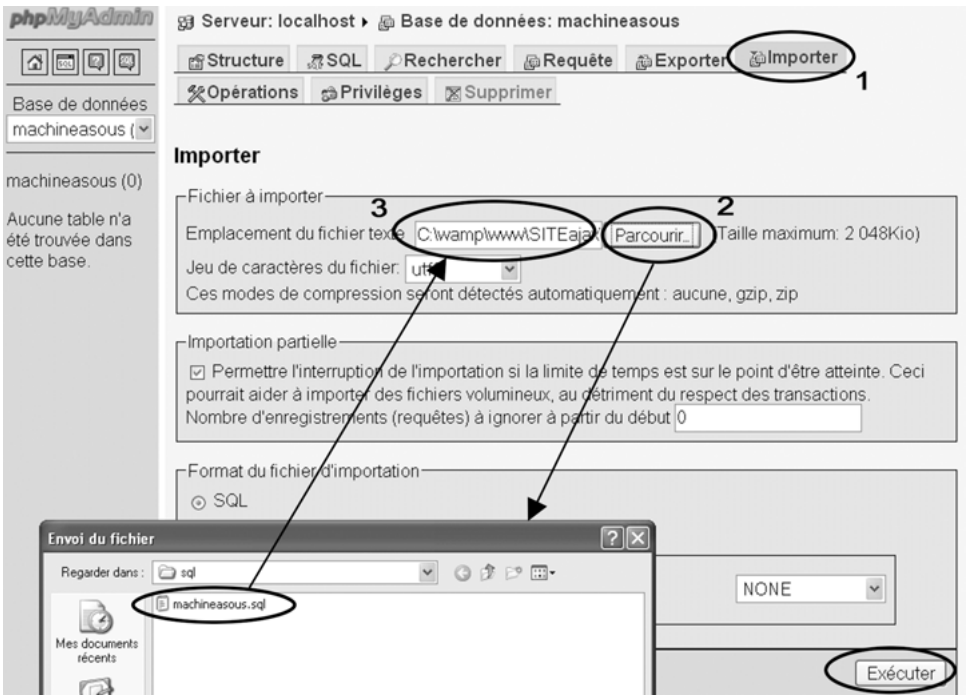


Figure 22-11

Pour la restauration d'une base de données, il faut utiliser le formulaire de la rubrique Importer en précisant l'emplacement du fichier de restauration machineasous.sql.

A

Configuration d'une infrastructure serveur locale pour Mac

Mamp, une infrastructure serveur pour Mac

Depuis les systèmes X, les machines Apple sont des ordinateurs Unix BSD qui intègrent un serveur Web Apache et le préprocesseur PHP par défaut. Le serveur Apache et les modules PHP sont pré-installés, mais ne sont pas activés par défaut. Par contre, la base de données MySQL et son gestionnaire phpMyAdmin ne sont pas pré-installés sur les Macintosh.

Il est donc nécessaire de configurer votre ordinateur Macintosh afin de disposer d'une infrastructure serveur locale et de pouvoir développer et tester des sites dynamiques en local.

Pour cela, vous disposez de deux alternatives :

- Activer le serveur Apache et PHP puis installer les applications manuellement.
- Installer automatiquement une suite de logiciel Mamp : Macintoch, Apache, MySQL, PHP (semblable à Wamp5 pour Windows).

La seconde solution étant beaucoup plus simple et rapide à mettre en place, nous vous proposons ci-après de vous guider pas à pas dans son installation.

Même si certains paramètres sont différents (numéro de port, etc), sachez que toutes les pages dynamiques présentées dans cet ouvrage pour l'environnement Windows peuvent être réalisées de la même manière avec votre Macintosh.

Installation de Mamp

Avant toute chose, vous devez commencer par télécharger le dernier package Mamp sur le site de son éditeur (voir figure A-1, *http://www.mamp.info*). Pour votre information, sachez que pour cette démonstration, nous avons utilisé la version Mamp 1.7, mais la démarche est semblable si vous utilisez une version plus récente.

Figure A-1

Saisissez l'adresse de l'éditeur de Mamp (http://www.mamp.info) dans votre navigateur puis cliquez sur le bouton Download now situé en bas à droite de l'écran.

Cliquez sur le bouton Download now de la page d'accueil du site (voir figure A-1). Dans la seconde fenêtre, choisissez l'option correspondante à votre ordinateur puis téléchargez le package sur votre ordinateur dans le dossier de votre choix et cliquez sur le package pour démarrer l'installation. Une fenêtre avec la procédure d'installation de Mamp doit ensuite apparaître. Ouvrez une fenêtre Finder et cliquez sur la catégorie Applications (voir figure A-2) puis glisser le dossier Mamp dans votre répertoire Applications. La suite Mamp est maintenant installée sur votre ordinateur.

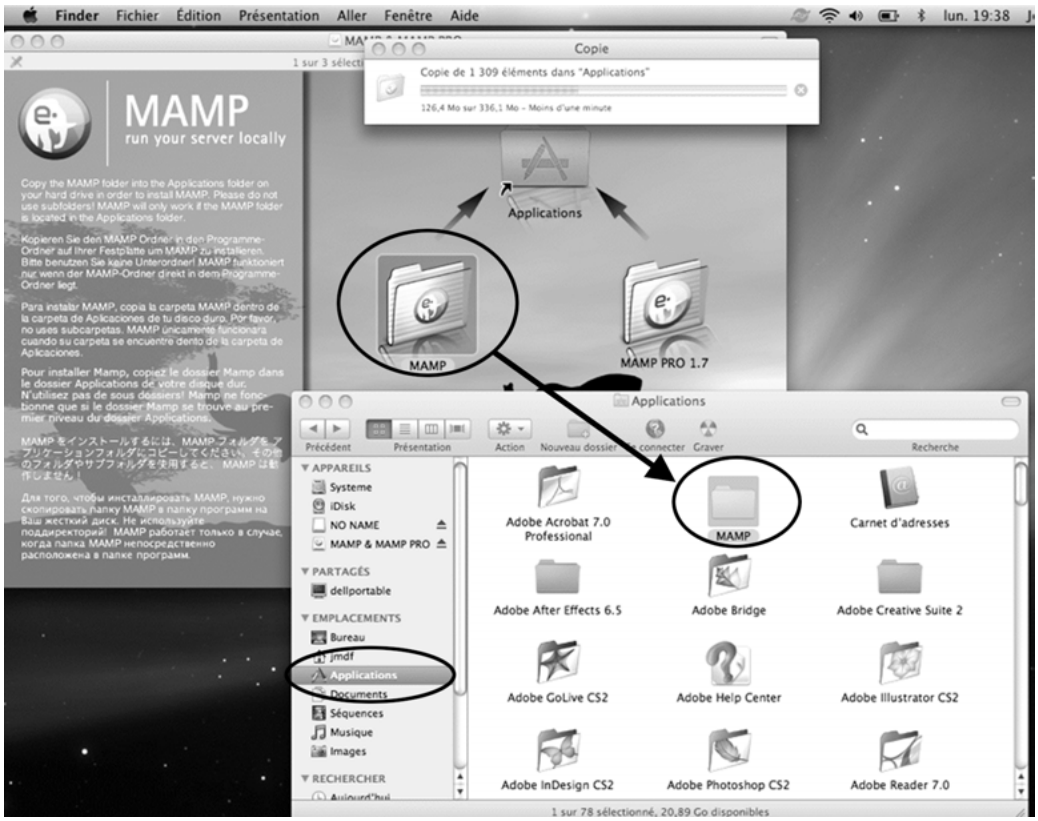


Figure A-2

Après avoir ouvert le répertoire Applications dans une fenêtre Finder, vous devez y copier le dossier complet de Mamp sans créer d'autre sous-répertoire.

Utilisation de Mamp

Une fois la suite Mamp installée sur votre ordinateur, nous vous suggérons de créer sur votre bureau un alias du fichier application situé dans le dossier Mamp (voir la figure A-3). Cliquez ensuite sur cet alias pour démarrer l'application. Une nouvelle fenêtre doit alors apparaître (voir la figure A-3) dans laquelle les deux voyants représentant les serveurs Apache et MySQL doivent progressivement passer au vert. Dans cette même fenêtre, vous pouvez configurer les différentes options de Mamp en cliquant sur le bouton `Préférences`. Cependant, nous vous conseillons de garder les paramétrages par défaut dans un premier temps. À noter que pour réduire cette fenêtre, vous devez cliquer sur le bouton orange situé en haut et à gauche de la fenêtre et non cliquer sur le bouton `Quitter` qui a pour incidence de fermer toutes les applications Mamp et, du même coup, les serveurs Apache et MySQL.

Le bouton « `Ouvrir la page d'accueil` » vous permet d'accéder à une page Web regroupant les différentes applications de la suite Mamp. La première page `Start` vous informe des paramètres à utiliser pour accéder au serveur MySQL (voir figure A-4). La seconde page `phpinfo` est très intéressante car elle affiche toutes les informations correspondant à la configuration actuelle de PHP. La page `phpMyAdmin` permet d'accéder au gestionnaire de

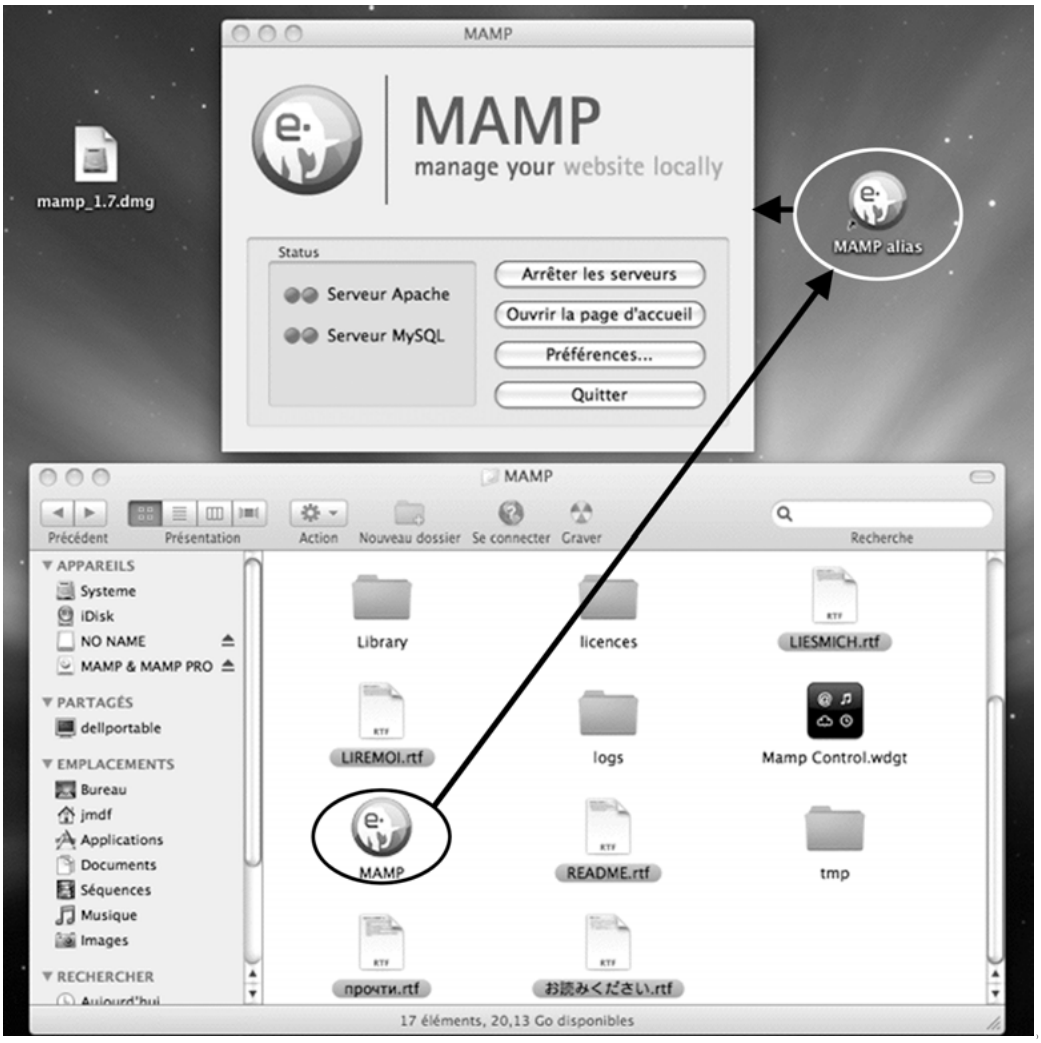


Figure A-3

Pour faciliter les futurs démarrages de Mamp, nous vous conseillons de créer un alias du fichier application sur votre bureau.

la base de données MySQL ou bien au gestionnaire du système de données intégré à PHP, SQLite (ce gestionnaire n'est pas utilisé dans le cadre de cet ouvrage). Enfin, la dernière page `FAQ` vous permet de connaître les différentes versions des applications de la suite Mamp et répond aux questions fréquemment posées.

Par la suite, l'utilisation de l'infrastructure serveur MAMP sera semblable à celle décrite dans les ateliers de cet ouvrage, hormis le fait que les adresses du Web Local et du serveur MySQL devront être accompagnées d'un numéro de port spécifique comme nous vous le rappelons ci-dessous :

Adresse du Web Local :

```
http://localhost:8888
```

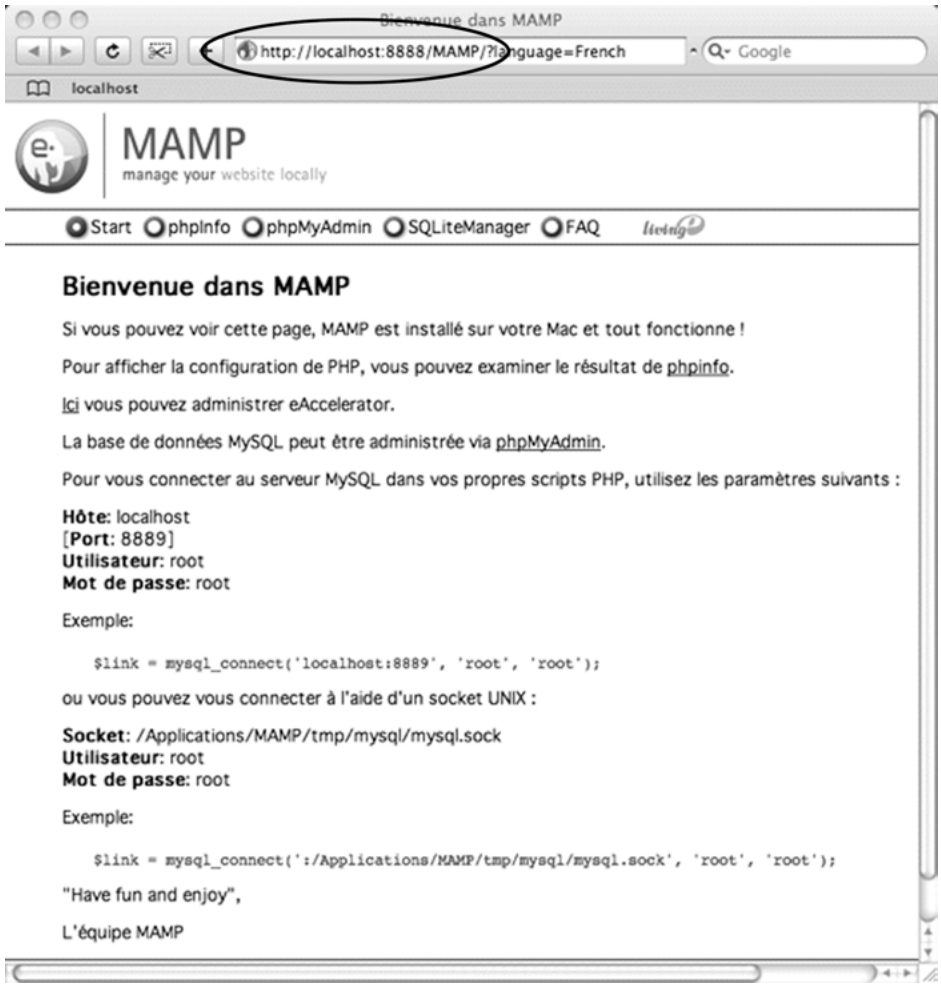


Figure A-4

La page Start vous informe des paramètres MySQL à utiliser dans vos scripts PHP et vous permet d'accéder au gestionnaire de la base de données phpMyAdmin.

Adresse du serveur MySQL :

```
http://localhost:8889
```

En ce qui concerne la configuration du site `SITEajax` avec Dreamweaver, vous devrez créer puis sélectionner un dossier `SITEajax` placé dans le répertoire `htdocs` lui-même situé dans `Applications/MAMP/`.

Le chemin complet du répertoire contenant les fichiers sera donc le suivant :

```
Systeme:Applications:MAMP:htdocs:SITEajax
```

De même, si vous désirez utiliser la fonctionnalité Aperçu dans le navigateur (attention, le raccourci sera Alt + F12 avec le Dreamweaver sous Mac), vous devrez alors configurer le préfixe d'url du serveur d'évalution avec l'adresse suivante :

```
http://localhost:8888/SITEajax/
```

B

Test et débogage (PHP et JavaScript)

Conseils pour bien programmer

Pour bien programmer et créer des scripts performants, quelques règles doivent être respectées.

Utilisez l'indentation

L'indentation correspond à la mise en forme des blocs de code grâce à des tabulations différentes selon le niveau d'imbrication des structures de boucle ou de choix dans lesquels ils se trouvent. Cela met en valeur le début et la fin des boucles et évite notamment d'oublier des accolades de fin ou de début.

Avec l'éditeur de Dreamweaver, vous pouvez décaler facilement le code à partir du menu Edition>Code de retrait (ou encore plus facilement avec la touche Tabulation). Toutes les nouvelles lignes seront décalées du même espace. Pour revenir d'un pas de décalage à gauche, utilisez l'option Décalage négatif du menu Edition (ou utilisez la touche Backspace).

Commentez votre code

Qu'il soit maintenu par le programmeur lui-même ou par une tierce personne, un programme doit toujours être parfaitement commenté (vous me remercierez peut-être dans quelques mois lorsque vous devrez modifier votre propre code…). Les commentaires d'un programme doivent porter sur des informations liées au contexte de l'application et non rappeler toutes les descriptions des fonctions issues du manuel du langage utilisé.

Les commentaires permettent aussi de préciser les liens qui existent entre plusieurs éléments d'un code, comme l'accolade d'ouverture et de fin d'un bloc. Une solution astucieuse consiste à commenter chaque accolade de fin de bloc avec un rappel de la condition de choix ou de boucle qui lui a donné naissance.

Avec l'éditeur de Dreamweaver, il est pratique de commenter un ensemble de lignes en utilisant les boutons latéraux (placés à gauche de l'éditeur de code) `Appliquer un commentaire` et `Supprimer un commentaire`.

Pour appliquer des commentaires, veillez bien à utiliser la syntaxe de PHP ou de JavaScript s'ils sont intégrés dans une zone de code, soit `//` pour un commentaire sur une simple ligne ou `/*` et `*/` pour un commentaire sur plusieurs lignes et la syntaxe des commentaires HTML, à savoir les balises `<!--` et `-->`, s'ils sont placés en dehors des balises PHP ou JavaScript.

Bien nommer les variables et les fichiers

Le choix du nom d'une variable est important. Dès qu'une variable concerne un élément fonctionnel du programme, nommez-la avec un nom explicite en rapport avec sa fonction. Vous pouvez par exemple utiliser la « convention du chameau » (appelée aussi « CamelCase minuscule ») qui consiste à concaténer une courte phrase qui caractérise la variable en ne conservant en majuscule que la première lettre de chaque mot (exemple en PHP : prix du produit deviendrait `$prixProduit`). En revanche, les variables d'utilisation ponctuelle, comme les compteurs de boucle ou les variables de fonction, peuvent conserver des appellations courtes et génériques (`$i` par exemple) car leur portée est limitée au bloc de la fonction et ne risque pas d'entrer en conflit avec une autre variable de même nom.

Pour les noms des fichiers, élaborez une convention de nommage afin de définir un préfixe commun pour tous les fichiers qui réalisent une action d'une même fonctionnalité. Cela permet d'identifier les fichiers, mais aussi de les regrouper facilement par fonctionnalité dans l'arborescence d'un répertoire à l'aide d'un simple tri alphabétique (par exemple en PHP `caddieModif.php` et `caddieSupp.php` ou en JavaScript `fonctionsAjax.js` et `fonctionsMachine.js`).

L'erreur du point-virgule

Parmi les erreurs habituelles, l'oubli du point-virgule pour terminer une instruction est certainement la plus fréquente. Aussi, nous vous recommandons de n'écrire qu'une seule instruction par ligne afin de vérifier facilement la présence du point virgule terminant chaque ligne.

Si le point virgule est indispensable à la fin de chaque instruction PHP, il n'est pas toujours obligatoire en JavaScript. Cependant, je vous conseille quand même de mettre un point-virgule à la fin de chaque instruction JavaScript, cela vous évite de l'oublier lorsqu'il est obligatoire et rend votre code plus lisible.

Utilisez les fonctions

Si vous découpez votre code en parties réutilisables, vous pouvez créer des fonctions faciles à exploiter dans toutes les pages du site. En outre, le code du script principal est plus léger et bien plus lisible. Rassemblez ces parties réutilisables dans un fichier commun (`nomFichier.inc.php` en PHP ou `nomFonction.js` en JavaScript) que vous pouvez appeler au début de chaque page grâce à la commande `require()` en PHP ou avec une balise `<script>` en JavaScript.

Utilisez les fragments de code

Pour être encore plus efficace dans le développement de vos applications et pour gagner en productivité, utilisez les fragments de code de Dreamweaver. Cela vous évite de ressaisir à chaque développement des parties de code fréquemment utilisées. Par défaut, Dreamweaver est livré avec des fragments de code standard classés par thème que vous pouvez trouver dans le panneau Fichier/Fragments de code. Cependant, la version standard comporte peu de fragments (en PHP, il faut d'ailleurs créer un répertoire PHP alors qu'avec JavaScript, ce répertoire existe déjà). Il vous faudra donc développer vos propres fragments avant de pouvoir les intégrer par la suite dans vos pages. Vous pouvez aussi importer des groupes de fragments de code afin de disposer rapidement d'un ensemble de ressources créées par des développeurs chevronnés.

Construisez brique par brique

De même qu'un maçon construit une maison brique par brique sur des fondations solides, le programmeur doit commencer par bien analyser le projet dans son ensemble pour définir sa structure et élaborer des modules correspondant à chaque fonctionnalité. Réalisez des essais pour chacun des modules dès qu'ils sont opérationnels et n'attendez surtout pas que tout le site soit créé pour les passer au banc de test.

Techniques de débogage PHP

Analyse d'un message d'erreur PHP

Avant d'être envoyée vers le navigateur, la syntaxe des scripts PHP est d'abord analysée puis exécutée. Lors de ces deux étapes peuvent apparaître des erreurs de syntaxe ou de sémantique. Des erreurs liées à la conception de votre programme (erreur logique) ou au contexte dans lequel il s'exécute (erreur d'environnement) peuvent également se produire. Selon le type d'erreur, des messages différents sont envoyés par PHP et s'affichent dans votre navigateur. Dans un premier temps, il est important de bien analyser ces messages d'erreur. Pour ce faire, il faut connaître la syntaxe du message d'erreur et les différents types d'erreurs qui peuvent se produire.

Tableau B-1 Syntaxe d'un message d'erreur

`Niveau_erreur : message_erreur in nom_fichier on line num_ligne`	
Légende	`Niveau_erreur` : les niveaux d'erreur peuvent être : – `Parse_error` : erreur de syntaxe lors de l'analyse ; – `Fatal_error` : erreur qui arrête le script ; – `Warning` : erreur qui se contente d'afficher un avertissement mais qui permet de poursuivre le script. `message_erreur` : message correspondant à l'erreur rencontrée (les messages sont souvent identiques au niveau). `nom_fichier` : nom du fichier dans lequel l'erreur a été détectée. `num_ligne` : numéro de la ligne où se trouve théoriquement l'erreur.

Voici un exemple d'erreur :

```
//ligne de script qui a produit l'erreur (point-virgule oublié en bout de ligne)
echo "bonjour"
//la version corrigée serait : echo "bonjour";
```

```
//message d'erreur affiché dans le navigateur :
Parse error: parse error, unexpected T_VARIABLE, expecting ',' or ';'
in c:\wamp\www\sitephp\debugphp\erreur1.php on line 4
```

Si on analyse l'erreur ci-dessus, on peut en déduire les informations suivantes :

```
niveau_erreur : Parse error ;
message_erreur : « parse error, unexpected T_VARIABLE, expecting ',' or ';' » ;
nom_fichier : c:\wamp\www\sitephp\debugphp\erreur1.php ;
num_ligne : line 4.
```

Utilisez l'équilibrage des accolades

Le non-respect de l'équilibrage des accolades d'ouverture et de fermeture est fréquemment à l'origine des erreurs de syntaxe. Utilisez l'équilibreur d'accolades que l'éditeur de Dreamweaver met à votre disposition pour vous assurer que votre programme respecte bien cette règle. Pour activer l'équilibrage d'accolades, placez votre pointeur à droite de la première accolade du bloc à tester, puis activez le testeur (sélectionnez Edition>Equilibrer les accolades ou utilisez le raccourci clavier `Ctrl + ,`). Toute la zone correspondant à un bloc équilibré (selon le niveau d'imbrication) est alors automatiquement sélectionnée. Si vous renouvelez le test, la zone sélectionnée s'étend au niveau d'imbrication supérieur et ainsi de suite jusqu'au dernier bloc. Un son signale que le test est terminé. Vérifiez qu'il ne reste aucune accolade en dehors de la zone sélectionnée. Dans l'affirmative, ajoutez l'accolade manquante et renouvelez le test. Si le testeur ne peut pas poursuivre sa recherche dès le début du test, il émet un son pour vous le signaler.

Détectez les erreurs de logique

Les erreurs de syntaxe ou de sémantique sont toujours accompagnées d'un message d'erreur qui précise le fichier et la ligne où se trouve l'erreur : il suffit en général de revoir le code situé à ce niveau pour résoudre le problème. En ce qui concerne les erreurs de logique, il est parfois difficile de localiser les lignes de code qui sont à l'origine du mauvais fonctionnement. Dans ce cas, le moyen le plus simple consiste à commenter les lignes (utilisez `//`) ou les blocs de code (utilisez `/*` et `*/`) susceptibles de provoquer l'erreur et à tester de nouveau le programme. Ainsi, par recoupements, vous pouvez circonscrire l'erreur de manière précise.

La fonction phpinfo()

La fonction `phpinfo()` affiche tous les paramètres de la configuration de PHP. Pour créer une page d'affichage de ces paramètres, saisissez cette fonction dans une page PHP (qui est nommée `phpinfo.php`, par exemple), comme indiqué ci-dessous, et appelez-la depuis le Web local ou depuis votre serveur distant. Vous pouvez également l'ajouter en bas de vos pages en cours de test, afin d'afficher les différentes variables actives et connaître leur valeur.

Voici le code à saisir dans le fichier `phpinfo.php` pour afficher la configuration de PHP :

```
<?php
phpinfo();
?>
```

Outre tous les paramètres PHP, qui peuvent s'avérer très utiles pour connaître la configuration de votre serveur (et la comparer avec celle de votre serveur local s'il s'agit d'une erreur contextuelle), la fonction `phpinfo()` affiche toutes les valeurs des variables créées avec des requêtes `GET` ou `POST` ainsi que le contenu des cookies.

Les pièges

L'examen des valeurs affectées à certaines variables pour différents endroits du script est souvent nécessaire au dépannage. Pour ce faire, vous pouvez ajouter dans votre page à tester des pièges qui affichent les noms des variables à tester suivis de leur valeur. Vous pourriez évidemment vous contenter d'ajouter dans votre code une simple fonction `echo $var1;` mais si vous désirez tester plusieurs variables dans la même page, cela devient vite illisible. Nous vous suggérons donc d'utiliser le code ci-dessous, qui présente l'avantage de rappeler le nom de la variable et d'insérer automatiquement un retour à la ligne après chaque test. Après son utilisation, vous pourrez localiser rapidement tous les endroits où vous avez inséré un piège. L'astuce consiste à lancer une recherche sur le mot `PIEGE` et à commenter la ligne ou à la supprimer complètement si tous les problèmes sont résolus.

```
echo '$var1='.$var1.'<br>'; //-----------PIEGE valeur
```

Dans certains cas, l'erreur peut provenir du type de la variable traitée. Pour tester cette information dans le programme, ajoutez l'affichage de son type, avec la fonction `gettype()` à la suite du piège précédent :

```
echo '$var1='.$var1.' de type '.gettype($var1).' <br>'; //-PIEGE type
```

Enfin, pour dépanner les scripts utilisant des variables de type tableau, employez la fonction `print_r()` que nous avons présentée lors de l'étude des tableaux dans le chapitre 6. Nous vous conseillons d'ailleurs de l'ajouter à vos fragments de code de Dreamweaver, ainsi par la suite, il suffira de saisir le nom du tableau à tester et à cliquer sur ce fragment pour que le code complet du piège soit ajouté autour du nom du tableau.

Exemple avec un simple tableau `$tab1` :

```
echo "<pre>";
print_r($tab1) ; //--------PIEGE tableau
echo "</pre>";
```

Enfin, vous aurez très fréquemment à tester les variables HTTP de votre page (lors de l'envoi d'informations issues d'un formulaire ou d'une requête Ajax, ou mémorisées dans des variables de session…), pour mémoire ces informations sont stockées, elles aussi, dans des tableaux et l'utilisation de la fonction `print_r()` vous sera d'une grande utilité pour connaître le contenu des ces variables.

Exemple avec un tableau `$_POST` :

```
echo "<pre>";
print_r($_POST); //--------PIEGE tableau
echo "</pre>";
```

Les fonctions de débogage

Une solution plus élaborée consiste à développer une fonction activée lors du débogage afin d'afficher ou d'enregistrer une trace de l'action réalisée. Pour activer ou désactiver cette fonction, vous pouvez la passer dans l'URL en lui affectant la valeur `1` pour activer

le débogage (exemple : `mapage.php?modedebug=1`). Vous pouvez ajouter localement cette fonction au code de la page, mais il est plus judicieux de l'intégrer dans un fichier de fonctions appelé par une commande `require()`. Vous pouvez trouver ci-dessous un exemple de fonctions à utiliser lors de la mise au point de vos programmes en local :

```
<?php
//fonction de débogage à insérer dans la page testée
//-----initialisation des variables
 if(!isset($_GET['modedebug'])) $_GET['modedebug']=0;
 else $modedebug=$_GET['modedebug'];
 //fonction de débogage à insérer dans la page testée
 function debugphp($var1)
 {
  if($GLOBALS["modedebug"]==1)
   echo 'PIEGE:$var1='.$var1.' de type '.gettype($var1).' <br>';
 }
//affectation d'une variable pour les besoins de l'exemple
$var2="bonjour";
//insertion du piège pour tester la variable $var2
debugphp($var2);
?>
```

Si vous testez l'exemple ci-dessus en passant un paramètre dans l'URL pour indiquer que vous désirez afficher le piège (exemple : `mapage.php?modedebug=1`), votre navigateur doit afficher le texte du piège ci-dessous :

```
PIEGE : $var1=bonjour de type string
```

Attention !

Si la variable que vous désirez tester n'est pas initialisée, des messages d'erreur `Undefined variable` peuvent apparaître à l'écran selon la configuration de votre serveur. Pour éviter cela, ajoutez la ligne de code suivante avant votre piége. La syntaxe `$var1` est valable pour une variable interne, mais s'il s'agit d'une variable HTTP issue d'un formulaire, d'une session, d'un cookie ou encore passée dans l'URL, utilisez le tableau de variable adapté (`$_XXX['var1']`) :

```
if(!isset($var1)) $var1="variable inconnue";
```

Suppression des messages d'erreur

Si, lors du développement, tous les messages doivent être affichés afin de mettre au point le programme, en production il est préférable que ces messages ne soient pas affichés dans la page Web visible par tous les internautes. Vous pouvez neutraliser l'apparition de ces messages à l'écran en ajoutant un `@` devant l'expression ou la fonction concernée. Considérez cette méthode comme une solution de dépannage en attendant de trouver la cause du problème ou pour empêcher l'affichage d'un simple `warning` qui n'a pas d'incidence sur le fonctionnement du script de la page.

Dans cet exemple, nous allons créer une erreur de division par 0 :

```
$var1=0; //simulation d'une erreur d'affectation pour les besoins du test
$testerreur=@(5/$var1); //ici il n'y aura pas de message d'erreur
```

Testez vos requêtes SQL dans phpMyAdmin

Si vous développez des pages intégrant des requêtes SQL, il est possible que l'erreur provienne d'un résultat erroné ou manquant remonté par la base, voire d'une erreur de syntaxe dans la requête SQL. Pour éviter ce genre de problème, testez la requête utilisée avant de l'intégrer dans le script PHP de la page dynamique.

Pour tester une requête, il faut d'abord ajouter un piège dans le code PHP afin d'afficher la requête SQL à l'écran (exemple : `echo $query`). Puis afficher la page dans le navigateur pour la copier puis la coller dans la zone `Exécuter une requête` du gestionnaire phpMyAdmin. Après avoir cliqué sur le bouton `Exécuter`, le résultat de la requête doit alors s'afficher dans l'écran du gestionnaire. Si une erreur est signalée ou si le résultat ne correspond pas à vos attentes, corrigez la requête directement dans le gestionnaire. Une fois que le résultat vous convient, reportez vos modifications dans le script PHP avant de la tester à nouveau dans la page dynamique.

Techniques de débogage JavaScript

Une bonne méthode pour développer en JavaScript consiste à commencer par mettre au point la structure XHTML, la mise en forme CSS et la programmation JavaScript avec Firefox puis terminer par l'adaptation aux autres navigateurs courants que l'on retrouve sur Internet, à savoir Internet Explorer bien sûr, mais aussi Safari et Opera.

En effet, si le navigateur Firefox a déjà le mérite de respecter les standards du W3C (contrairement à Internet Explorer…), il a aussi l'énorme avantage de disposer d'une multitude d'extensions, dont certaines vous facilitent la vie pour mettre au point vos programmes. Firebug fait partie de celles-ci et j'espère que vous avez déjà apprécié son usage lors de la réalisation des premiers ateliers de cet ouvrage (revoir si besoin le chapitre 8 dans lequel nous détaillons ses principales fonctionnalités).

La fonction alert()

La manière la plus courante d'afficher l'état d'une variable lors de l'exécution d'un programme JavaScript consiste à utiliser la fonction `alert()` qui affiche l'information dans une boîte de dialogue.

Par exemple, si vous désirez connaître la valeur de la variable `montant` à un moment particulier de votre programme, il suffit d'ajouter l'instruction suivante à l'endroit désiré :

```
alert('valeur de montant ='+montant);
```

Cependant, cette méthode n'est pas sans défaut : d'une part, elle bloque l'application tant que vous n'avez pas validé la boîte de dialogue et, d'autre part, elle devient vite envahissante en affichant de nombreuses données.

L'élément title

Pour éviter d'être bloqué par la fenêtre d'alerte, une technique très simple consiste à renvoyer les informations à observer dans la balise de titre de la fenêtre du navigateur. Le code de l'exemple ci-dessous permettra ainsi d'afficher la valeur prise par `parametres` (soit `nom=Defrance`) à l'endroit où le code à été ajouté.

```
parametres="nom=Defrance";
document.title=parametres;
```

La console de Firebug

Avec la console de Firebug il n'est plus utile de faire appel à une fonction `alert()` qui bloque votre application ou encore à l'élément `title` qui reste très limité en ce qui concerne la taille de la valeur à afficher.

En effet, la console de Firebug permet d'afficher un journal de différents messages qui ont été intégrés au préalable dans le code. Firebug propose différentes méthodes pour permettre de qualifier la nature du message par un affichage différent ; `debug()` pour les messages de débogage, `info()` pour une simple information, `warn()` pour un avertissement et `error()` pour signaler une erreur. Toutes ces méthodes fonctionnent de la même manière et vous être libre d'utiliser celle qui convient le mieux à la situation.

En guise d'exemple, voici le code que vous devez ajouter dans votre programme pour pouvoir afficher un message d'information avec l'état d'une variable (en l'occurrence `montant` dans l'exemple ci-dessous) dans la fenêtre de Firebug sans perturber le fonctionnement de l'application :

```
console.info('valeur de montant ='+montant);
```

L'inspecteur DOM de Firebug

Le réflexe du développeur Web est souvent d'afficher le code source de la page en cours de développement. Cependant, pour les programmes JavaScript, cette technique est d'un intérêt très limité. En effet, comme JavaScript modifie dynamiquement les éléments de la page HTML, le code source ne correspond plus au résultat affiché dans la page du navigateur car les modifications du JavaScript sont effectuées sur la hiérarchie du DOM placée dans la mémoire du navigateur et non sur le code source initialement chargé dans le navigateur.

Heureusement, avec l'outil de débogage Firebug, il est très facile de consulter le DOM de la page à un moment donné. Il suffit pour cela de dérouler l'arborescence DOM en cliquant successivement sur les petits + qui précédent chaque nœud élément de la hiérarchie de la page pour accéder ainsi à l'état d'un élément spécifique.

Mais la console Firebug n'est pas limitée à une simple consultation du DOM : elle vous permet aussi de modifier en temps réel les propriétés des différents éléments, voire d'en ajouter de nouvelles. De même, pour ceux qui ont tendance à se perdre dans l'arborescence du DOM (il est vrai qu'il n'est pas toujours simple de s'y retrouver…) sachez que vous pouvez accéder directement à l'élément du DOM désiré en faisant un clic droit sur l'objet concerné dans la fenêtre du navigateur et en sélectionnant `Inspect Element`. La fenêtre DOM de Firebug affiche alors automatiquement le nœud correspondant à l'objet sélectionné.

L'inspecteur HTML de Firebug

Si la fenêtre du DOM de Firebug vous effraie, vous devriez alors trouver un grand intérêt dans sa fenêtre HTML. En effet, cette fenêtre ne représente pas un simple code source statique de la page initialement chargée mais illustre l'évolution dynamique des différentes balises HTML. Pour s'en convaincre, il suffit d'ouvrir cette fenêtre et de développer un élément sur lequel une fonctionnalité de l'application doit intervenir (la balise `div` d'identifiant `info` par exemple). Si vous démarrez l'application (en cliquant sur le bouton JOUER, par exemple) vous pouvez alors voir en temps réel dans la fenêtre HTML

l'évolution de tous les contenus et attributs qui vont être modifiés au cours du traitement. Enfin, cette fenêtre permet aussi de forcer le contenu d'une balise ou la valeur d'un attribut afin d'observer immédiatement son incidence dans la fenêtre du navigateur (faites le test en changeant par exemple le nom du gagnant en le remplaçant par le votre…).

Les erreurs de syntaxe avec Firebug

Personne n'est à l'abri d'une erreur de syntaxe. Si cela se produit, une icône rouge vous le signale en bas à droite du navigateur Firefox lors de vos tests. Lorsque vous cliquez sur cette icône, Firebug s'ouvre en affichant dans la fenêtre de la console la nature de l'erreur rencontrée (soit `syntax error`). En dessous de ce message se trouve l'instruction suspectée en vert et, si cela ne suffit pas pour trouver l'erreur, vous pouvez alors cliquer sur ce lien afin que Firebug vous affiche dans la fenêtre script sa position dans le programme.

La fenêtre Script de Firebug

La fenêtre Script de Firebug permet de poser très facilement des points d'arrêt dans vos programmes. Pour cela, il suffit de cliquer dans la marge de gauche de la fenêtre Script au niveau de la ligne de script sur laquelle vous désirez marquer une pause. Un point rouge apparaît alors à cet endroit pour vous signaler que le point d'arrêt est enregistré (un second clic sur le même point le fait disparaître). À noter que si vous désirez accéder à d'autres programmes externes JavaScript que celui qui s'affiche actuellement dans la fenêtre Script, vous pouvez facilement passer de l'un à l'autre à l'aide du menu placé dans la barre Firebug au-dessus du bouton Script.

Une fois le point d'arrêt positionné, lancez l'application comme d'habitude (en cliquant sur le bouton JOUER, par exemple). Le programme démarre alors jusqu'à l'instruction repérée par le point d'arrêt (une petite flèche placée sur le point rouge indique que le programme est actuellement bloqué à ce niveau). Si maintenant vous déplacez le pointeur de votre souris sur les différentes variables de la fenêtre Script, une petite infobulle vous indique alors la valeur de l'information survolée. Évidement, vous pouvez aussi profiter de cette pause pour visiter les autres fenêtres de Firebug comme la fenêtre HTML ou CSS par exemple.

À ce stade, vous pouvez effectuer plusieurs actions à partir des boutons du panneau de contrôle situé à droite de la barre de menu de Firebug. Le premier bouton en forme de flèche bleue vous permet de relancer le programme jusqu'au prochain point d'arrêt (ou jusqu'à la fin du programme s'il n'y a qu'un seul point d'arrêt), alors que les autres boutons en forme de flèche vous permettent de continuer en marche pas à pas. À noter que si vous posez plusieurs points d'arrêt, vous avez la possibilité de les contrôler depuis la fenêtre de droite en cliquant sur l'onglet Breakpoints (activer ou désactiver un point, voire le supprimer complètement, aller directement à l'endroit du point…).

Observer les requêtes XMLHttpRequest

En ce qui concerne la mise au point des moteurs Ajax, l'observation des données échangées entre le navigateur et le serveur est certainement la fonctionnalité la plus intéressante pour mettre au point les programmes JavaScript et PHP qui gèrent ces échanges. Pour observer la requête envoyée ou la réponse du serveur, il suffit d'ouvrir la fenêtre Console après avoir déclenché une requête Ajax. Cliquez ensuite sur le petit + devant la requête concernée, vous avez alors la possibilité de consulter les informations émises par

la requête en cliquant sur l'onglet Params et la réponse du serveur en cliquant sur l'onglet Response. Le contenu des en-têtes des deux échanges peut aussi être consulté en cliquant cette fois sur l'onglet Header.

Index

Premières applications Web 2.0 avec Ajax et PHP

Jean-Marie Defrance

Diplômé d'un DEA de didactique de l'informatique, Jean-Marie Defrance enseigne le multimédia à Gobelins, l'école de l'image. Il est par ailleurs directeur technique de l'Agence W, société spécialisée dans le développement de sites dynamiques.

Ajax, la technologie phare des sites Web de nouvelle génération

Ajax s'est aujourd'hui imposé comme l'outil idéal pour créer des applications Web réactives, dotées d'interfaces utilisateur riches et ergonomiques. Couplée à PHP-MySQL, cette technologie permet en outre d'échanger des données avec le serveur, de manière souple et rapide, et de les afficher dans le navigateur sans nécessiter de rechargement.

Réaliser ses premiers moteurs Ajax-PHP

Grâce à 40 ateliers pratiques de difficulté croissante, cet ouvrage vous guidera pas à pas dans la construction d'un moteur Ajax-PHP performant, en résolvant progressivement les principaux problèmes rencontrés dans la création d'une application Ajax. Vous découvrirez en outre les multiples manières d'utiliser l'objet XMLHttpRequest pour échanger avec le serveur des flux de données dans différents formats (texte, HTML, XML, JSON ou RSS). Vous apprendrez également comment une application Ajax, côté client, peut gérer des informations stockées dans une base de données MySQL via un script PHP. Enfin, pour mettre au point ces réalisations, ce livre propose différentes techniques de débogage qui exploitent les fonctionnalités de l'extension Firebug de Firefox.

Concevoir une application performante Ajax-PHP avec jQuery

jQuery est une bibliothèque JavaScript qui permet en particulier de mettre en œuvre des applications Ajax de manière simple et rapide. Ce livre vous explique en détail comment exploiter cette bibliothèque sur votre site pour augmenter votre productivité et la fiabilité de vos développements. Il présente en outre une sélection de plug-ins issus de jQuery capables de créer des applications Ajax et autres widgets avec une facilité déconcertante. Pour compléter votre apprentissage, la dernière partie de l'ouvrage est consacrée aux différentes technologies associées à Ajax (XHTML, CSS, XML, JavaScript, DOM, PHP et MySQL), en exposant pour chacune d'elles les connaissances nécessaires à la compréhension du livre.

Au sommaire

À qui s'adresse cet ouvrage ?

– A ceux qui souhaitent intégrer aisément et rapidement des applications Web 2.0 dans leurs projets Internet
– À tous ceux qui désirent apprendre à concevoir des applications Ajax couplées à PHP/MySQL

Sur le site www.editions-eyrolles.com
– Téléchargez le code source des exemples et des 40 ateliers de l'ouvrage
– Dialoguez avec l'auteur

Code éditeur : G12090
ISBN : 978-2-212-12090-5

www.editions-eyrolles.com
Groupe Eyrolles | Diffusion Geodif | Distribution Sodis

Conception : Nord Compo