Recommandations W3C et ACube

Normes de réalisation Internet d'un client léger

Version 3.1 du 22/02/2010

Etat : Validé

SUIVI DES MODIFICATIONS

Version	Rédaction	Description	Vérification	Date
0.1	S. Péguet	Création du document		20/12/02
1.0	S. Péguet	Version validée		31/01/03
1.1	S. Péguet	Version reprise pour standardisation (plan)		16/01/04
2.0	M. Aldana S. Péguet	Ajout du XHTML et reprise de l'ensemble des normes		02/02/04
2.1	M. Aldana S. Péguet	Ajout du HTML, CSS, Javascript et reprise de l'ensemble des normes		26/02/04
2.2	M. Aldana S. Péguet	Reprise des normes pour cohérence d'ensemble Mise en valeur des préconisations Elargissement des références XHTML		01/03/04
2.3	M. Aldana S. Péguet	Amélioration de la présentation des annexes et intégration de correctifs mineurs Reprise des normes pour cohérence vis à vis des normes Internet d'un client riche		12/03/04
2.4	G.Pasquereau	Evolution du cadre d'utilisation des balises liées aux applets		07/09/05
3.0	S.Pitoiset	Ajout de JSLint et Javascript Lint		24/10/06
3.1	G.Pasquereau	Actualisation		22/02/10

LISTE DE DIFFUSION

Organisation	Nom	Info	Commentaire	Validation
		☐	☐	☐

SOMMAIRE

1 OBJET DU DOCUMENT 8
1.1 Introduction 8
1.2 Limitations 9
1.3 Contenu du document 9
2 NORME DE DEVELOPPEMENT HTML 11
2.1 Introduction 11
2.2 Doctype 11
2.3 Gestion des fenêtres et des cadres 12
2.3.1 Les cadres (Frames) 12
2.3.2 Les dimensions 14
2.3.3 Les bordures 17
2.3.4 Les marges 18
2.3.5 Les cadres uniques 19
2.4 Structure générale d'une page HTML 21
2.4.1 Zone d'entête 21
2.4.2 Corps de la page 25
2.5 Gestion du texte et des images 27
2.5.1 Gestion du texte 27
2.5.2 Gestion des images 29
2.6 Gestion des tableaux 30
2.6.1 La balise <table> 30
2.6.2 La structure 30
2.6.3 Les lignes du corps 33
2.6.4 Les colonnes et les cellules d'une ligne 34
2.6.5 L'en-tête et pied 34
2.6.6 Les légendes 35
2.6.7 Les regroupements de colonnes 36
2.6.8 Les attributs 38
2.7 Gestion des formulaires 44
2.7.1 La balise <form> 44
2.7.2 La saisie de l'utilisateur 47
2.8 Gestion du dynamisme des pages 53
2.9 Icônes de pages Web 53
2.9.1 Le format des icônes 53
2.9.2 Le code HTML dans la page 53
3 NORME DE DEVELOPPEMENT CSS 55
3.1 Introduction 55
3.2 Syntaxe 56
3.2.1 Définition 56
3.2.2 Syntaxe Générale 56
3.2.3 L'instruction style en tant qu'élément (Embedding) 57
3.2.4 L'instruction style en tant qu'attribut (Adding in line) 57
3.2.5 Méthode Externe (Linking) 57
3.2.6 Méthode Externe par Import 57
3.3 Sélecteurs 57
3.3.1 Les sélecteurs simples 58
3.3.2 Les combinateurs 58

3.3.3 Les pseudo-éléments 59
3.4 SPAN et DIV 59
3.4.1 Définition 59
3.4.2 Exemple SPAN 59
3.4.3 Exemple DIV 59
3.5 Arrière-plan 60
3.5.1 Définition 60
3.5.2 Propriétés 60
3.5.3 Propriétés 60
3.6 Texte 61
3.6.1 Définition 61
3.6.2 Propriétés 62
3.7 Polices 62
3.7.1 Définition 62
3.7.2 Propriétés 63
3.8 Bordures 64
3.8.1 Définition 64
3.8.2 Propriétés 64
3.9 Marges 65
3.9.1 Définition 65
3.9.2 Propriétés 65
3.10 Espacement 66
3.10.1 Définition 66
3.10.2 Propriétés 66
3.11 Listes 66
3.11.1 Définition 66
3.11.2 Propriétés 67
3.12 Dimensions 67
3.12.1 Définition 67
3.12.2 Propriétés 68
3.13 Positionnement 68
3.13.1 Définition 68
3.13.2 Propriétés 69
3.14 Liens 69
3.14.1 Définition 69
3.14.2 Propriétés 70
3.15 Barre de défilement : Overflow 70
3.15.1 Définition 70
3.15.2 Propriétés 70
3.16 Affichage : Display 71
3.16.1 Définition 71
3.16.2 Propriétés 71
3.17 Pointeur : Cursor 71
3.17.1 Définition 71
3.17.2 Propriétés 71

4 NORME DE DEVELOPPEMENT XHTML 73
4.1 Introduction 73
4.2 Différences entre le HTML et le XHTML 73
4.2.1 Formulation des tags 73
4.2.2 Format du document 75
4.2.3 Validation du document par DTD 76
4.3 XHTML support du CSS 77
4.3.1 Illustration sur un cas d'utilisation : navigation par onglets 77

4.3.2 Choix du type de rendu....78

5 NORME DE DEVELOPPEMENT JAVASCRIPT....80
5.1 Références et spécifications du Javascript....80
5.1.1 JavaScript/JScript/ECMAScript....80
5.1.2 DOM Level 0....82
5.1.3 DOM W3C....83
5.2 Gestion d'événements....83
5.2.1 Les événements....83
5.2.2 Accès à un événement....84
5.2.3 Les propriétés liées à un événement....85
5.2.4 Gestionnaire d'événements....86
5.2.5 Ordonnancement des événements....88
5.3 Convention d'utilisation JS....89
5.3.1 Emplacement du code JavaScript....89
5.3.2 Appel de fonction JavaScript dans un fichier en HTML....90
5.4 Gestion des objets JS....91
5.4.1 Interrogation d'objet DOM....91
5.4.2 Vérification de l'existence d'un objet ou d'une propriété....91
5.4.3 Utilisation d'objet JS en variable....92
5.5 Détection de compatibilité....93
5.5.1 Vis à vis du DOM W3C....93
5.5.2 Vis à vis d'une version d'un navigateur ou d'un OS....93
5.5.3 Vis à vis des cookies....94
5.5.4 Vis à vis d'un plugin....95
5.6 Gestion de la navigation....96
5.6.1 Construction des URLs....96
5.6.2 Appel d'une page lors de la sélection d'un item d'une boîte de sélection....97
5.6.3 Gestion de l'historique des navigateurs (boutons BACK et FORWARD du navigateur)....97
5.7 Gestion des formulaires....99
5.7.1 Validation d'un formulaire....99
5.7.2 Gestion d'appel multiple pour la validation d'un formulaire....103
5.7.3 Désactivation des actions utilisateur en attente de l'affichage de la page suivante....103
5.7.4 Contrôles de surface pour valider un formulaire....103
5.8 Gestion des effets dynamiques....105
5.8.1 Changement d'images....105
5.8.2 Affichage/Désaffichage d'une zone dynamique suivant une action utilisateur....106
5.8.3 Modification du contenu d'une zone dynamique suivant une action utilisateur....107
5.8.4 Affichage d'une information dans la barre de statut du navigateur....108
5.8.5 Modification du style d'un élément du document....108
5.9 Gestion de fenêtres....109
5.9.1 Gestion du multi-fenêtrage (FRAME et IFRAME)....109
5.9.2 Création d'une fenêtre fille....109
5.9.3 Impression....111
5.9.4 Fermeture....111
5.10 Gestion particulière pour les IFRAME....111
5.10.1 Affichage du résultat d'une recherche selon critères pré-saisis....111
5.10.2 Gestion de la pagination....112
5.10.3 Affichage d'un détail par sélection d'un item présent dans l'IFRAME....113
5.11 Gestion des cookies session par le client....114
5.12 Génération de contenu par le client....115
5.12.1 Génération d'une partie du contenu d'un document HTML....115
5.12.2 Génération du contenu d'une fenêtre surgissante....116
5.13 Gestion d'envoi d'un message électronique....117

5.13.1 Envoi d'un message électronique à un destinataire 117
5.13.2 Formatage du sujet et du corps d'un message électronique 118
5.14 Validation du code avec JSLint et JavaScript Lint 119
5.14.1 Présentation 119
5.14.2 Fonctionnalités de JSLint 119
5.14.3 Fonctionnalités de JavaScript Lint 122

6 NORME DE DEVELOPPEMENT PLUGINS 123
6.1 Applet 123
6.1.1 Introduction 123
6.1.2 Restriction de sécurité 123
6.1.3 Tag Applet 124
6.1.4 Tags object et embed 124
6.1.5 Recommandations 128
6.2 Flash 128
6.2.1 Introduction 128
6.2.2 Tags object et embed 128
6.2.3 Recommandations 131
6.3 Préconisations pour l'utilisation de flash ou d'une applet 131

ANNEXE A : CARACTERES SPECIFIQUES EN HTML 132
ANNEXE B : REFERENCES CSS 134
ANNEXE C : REFERENCES XHTML 144
ANNEXE D : REFERENCES JSCRIPT 171
ANNEXE E : REFERENCES JAVASCRIPT 182
ANNEXE F : REFERENCES DOM LEVEL 0 207
ANNEXE G : RÉFÉRENCES DOM W3C 228
ANNEXE H : REFERENCES DES EVENEMENTS JS 253
ANNEXE I : EXEMPLES D'APPLICATION DU FRAMEWORK POUR GERER UN FORMULAIRE 257
ANNEXE J : COMPARATIF INNERHTML VS DOM W3C 261
ANNEXE K : GESTION DU CACHE 266
ANNEXE L : BIBLIOGRAPHIE 268

TABLEAUX

Tableau 1 : types de déclaration de document 12
Tableau 2 : exemples de dimensionnement pour la balise Frameset 16
Tableau 3 : valeurs de l'attribut scrolling 16
Tableau 4 : rendus du cadre pour les Frames 17
Tableau 5 : exemples de dimensionnement pour la bordure des Frames 18
Tableau 6 : balises autorisées dans l'en-tête 22
Tableau 7 : attributs de la balise link 23
Tableau 8 : attributs de la balise style 25
Tableau 9 : attributs de la balise DIV 28
Tableau 10 : attributs de la balise IMG 29
Tableau 11 : règles d'application du dimensionnement 40

Tableau 12 : types de bordure (attribut frame) 42
Tableau 13 : types de bordure (attribut rules) 42
Tableau 14 : attributs de la balise form 45
Tableau 15 : protocoles d'envoi d'un formulaire 47
Tableau 16 : valeurs de l'attribut type de la commande input 47
Tableau 17 : types de bouton 50
Tableau 18 : valeurs de l'attribut wrap de la commande textarea 52
Tableau 19 : valeurs de l'attribut size de la commande select 52
Tableau 20 : propriétés CSS concernant l'arrière-plan 61
Tableau 21 : propriétés CSS pour le texte 62
Tableau 22 : propriétés CSS pour les polices 64
Tableau 23 : propriétés CSS pour les bordures 65
Tableau 24 : propriétés CSS pour les marges 66
Tableau 25 : propriétés CSS concernant l'espacement 66
Tableau 26 : propriétés CSS concernant les listes 67
Tableau 27 : propriétés CSS concernant les dimensions d'un élément 68
Tableau 28 : propriétés CSS concernant le positionnement d'un élément 69
Tableau 29 : propriétés CSS pour les liens 70
Tableau 30 : propriétés CSS de la barre de défilement overflow 70
Tableau 31 : propriétés CSS pour l'affichage des éléments (display) 71
Tableau 32 : propriétés CSS pour les pointeurs (cursor) 72
Tableau 33 : équivalences entre JavaScript et ECMAScript 80
Tableau 34 : versions de JavaScript vis à vis des navigateurs Netscape 81
Tableau 35 : versions de JScript vis à vis des logiciels Microsoft 81
Tableau 36 : événements sur un click de l'utilisateur 99
Tableau 37 : événements sur l'utilisation de la touche « Entrée » dans un élément Text du formulaire 100
Tableau 38 : événements sur l'utilisation de la touche « Entrée » dans un élément Check Box du formulaire 101
Tableau 39 : options possibles lors de la création d'une fenêtre fille 110
Tableau 40 : description des attributs falcultatifs d'un cookie session 114

FIGURES

Figure 1 : représentation des parties constituant un tableau 30
Figure 2 : illustration de l'instance d'un tableau 33
Figure 3 : illustrations des attributs align et valign d'une cellule de tableau 39
Figure 4 : illustrations de fusions entre cellules d'un tableau 44
Figure 5 : concept de la feuille de style 55
Figure 6 : gestion de l'historique du navigateur 98

DOCUMENTS DE REFERENCE

Version	Titre	Document
3.1	Normes de réalisation Internet d'un client riche	A3_NOR_NormesW3C Client Riche

1 OBJET DU DOCUMENT

Ce document présente l'ensemble des normes de réalisation pour les applications Internet. Il a pour vocation d'aider l'équipe projet à déterminer rapidement et sans risque les solutions techniques pour la réalisation des interfaces graphiques sur client léger.

L'ensemble de ces normes a pour leitmotiv de suivre les recommandations du consortium W3C pour pouvoir garantir une conformité sur les standards appliqués par les éditeurs de navigateur.

1.1 INTRODUCTION

Le HTML (Hypertext Mark-up Language) est un langage décrivant la structure des pages Web à l'aide de balises et permettant la navigation hypertextuelle entre les documents.

Le HTML ne permet pas de décrire de façon stricte l'apparence d'une page, mais plutôt la structure logique du document.

L'aspect physique de la page dépend du logiciel de visualisation (le navigateur) utilisé qui interprète, parfois de façon différente, les balises.

Le HTML, très basique à l'origine, a évolué au fur et à mesure du succès d'Internet afin de répondre à des besoins de plus en plus complexes, tant au niveau des possibilités d'affichage qu'au niveau des traitements dynamiques exécutés par le navigateur.

Ainsi, le HTML a été progressivement enrichi et complété avec :

- Les feuilles de styles (CSS) qui permettent un contrôle plus important de l'affichage,
- Le JavaScript qui permet d'exécuter des instructions de programme dans le navigateur.

De plus, successeur du HTML, le XHTML apporte une rigueur en offrant d'une part, un langage de structuration de contenu à la fois compatible avec la structuration du HTML et avec le XML, et d'autre part, en laissant le soin de la présentation aux CSS. Ainsi, ce langage permet d'interdire l'utilisation de balises obsolètes et de respecter l'usage normalisé de celles qui avaient été détournées de leur fonction.

Un site WEB moderne résulte donc de l'association de trois langages différents (XHTML, CSS, JavaScript) tous normalisés mais encore trop permissifs avec ces langages susceptibles d'être interprétés par une grande variété de navigateurs au comportement plus au moins fidèle aux recommandations du W3C.

C'est pourquoi une norme de développement pour chacun de ces langages s'impose, afin de garantir à l'application :

- un comportement homogène et fiable en fonction du niveau de compatibilité des configurations clientes ciblées (OS, Navigateur),
- une bonne évolutivité,
- une bonne maintenabilité,
- une bonne exploitabilité.

Les normes présentées visent à permettre un développement industriel du site garantissant son bon fonctionnement sur l'ensemble des configurations clientes ciblées, tout en gardant une possible compatibilité avec d'autres navigateurs n'engageant cependant pas le support des projets applicatifs.

Ces normes visent aussi à définir un ensemble de bonnes pratiques de réalisation en définissant :

- les liaisons entre les différents langages afin d'éviter l'entrelacement dans un même document de quatre langages différents,
- le périmètre d'utilisation de chacun des langages,
- un framework permettant la réutilisation de composants logiciels.

Le respect de ces normes doit conduire à :

- assurer le bon comportement de l'application pour chaque navigateur,
- diminuer la complexité des programmes,
- améliorer la lisibilité des programmes,
- diminuer le volume de tests de validation nécessaire en fonction des différentes configurations clientes ciblées.

Et ainsi à atteindre les objectifs d'évolutivité, maintenabilité et d'exploitabilité de l'application dans le respect des contraintes de fiabilité, de vitesse de développement et de budget.

1.2 LIMITATIONS

Ces normes sont constituées de sous ensembles liés aux possibilités des trois différents langages XHTML, CSS et JavaScript (dont le DOM W3C HTML).

Ces sous ensembles permettent de couvrir l'ensemble des besoins de charte graphique et ergonomique des applications à une date donnée.

L'utilisation d'une possibilité d'un de ces langages non décrite dans ce document est proscrite.

Toutefois, les technologies Internet étant en perpétuelle effervescence, ces normes peuvent s'avérer obsolètes (disparition d'un navigateur du marché) ou incomplètes (nouveau besoin ou évolution des technologies). Dans ce cas, il sera nécessaire de faire évoluer ces normes avant d'engager les travaux de réalisation.

Enfin, ces normes s'appliquent à la couche présentation d'une application WEB uniquement, et non à la couche génération de cette présentation.

Ceci signifie que ces normes adressent le XHTML, CSS et JavaScript produit par une application et non pas la façon de le générer. Ce dernier point est couvert par un guide de développement côté serveur qui ne fait pas l'objet de ce document.

De plus, le cadre d'utilisation de ces normes est lié au cadre d'application des principes du client léger. D'autres normes de réalisation Internet client riche font l'objet d'un autre document explicitant les sous ensembles liés aux possibilités des langages XML, DTD, XSD et DOM W3C XML. Cet autre document est une surcouche des normes présentes dans le présent document dans le cadre d'application des principes du client riche.

1.3 CONTENU DU DOCUMENT

Le présent document est donc découpé en partie dont chacune décrit les possibilités de chacun des langages couverts en fonction d'un besoin donné avec pour chacune de ces possibilités une illustration par un exemple d'implémentation.

De plus, pour faciliter la lecture de ces normes, une mise en valeur de l'information contenue est proposée pour distinguer l'information liée aux normes W3C et leurs déclinaisons pour les langages normalisés et les préconisations sur ces normes propres au pôle d'architecture de DSI/PSI.

Ainsi, un lecteur connaissant déjà les normes W3C doit pouvoir accéder directement aux préconisations appliquées au sein des applicatifs du Ministère des Affaires Etrangères à l'aide de l'introduction de l'information de manière suivante :

Préconisation ou recommandation sur la norme du pôle d'architecture DSI/PSI

Les annexes du présent document offrent un référencement de l'ensemble des directives de ces langages avec des préconisations de périmètre d'utilisation par code couleur en fonction des navigateurs.

Ainsi, vous trouverez dans les présentes annexes un référencement pour les navigateurs cibles suivants :

- Internet Explorer 5.0 à 6.0,
- Netscape 6.2 et 7,
- Mozilla 1.x,
- Opera 7,
- Safari.

Le périmètre d'utilisation de ces directives peut être mis à jour pour une autre cible de navigateur supportée ou toute nouveau besoin formulé au pôle d'architecture.

2 NORME DE DEVELOPPEMENT HTML

2.1 INTRODUCTION

Le HTML ("HyperText Markup Language") est un système qui formalise l'écriture d'un document avec des balises de formatage indiquant la façon dont doit être présenté le document et les liens qu'il établit avec d'autres documents.

Il permet, entre autre, la lecture de documents sur Internet à partir de machines différentes grâce au protocole http (HyperText Transfer Protocol), permettant d'accèder via le réseau à des documents repérés par une adresse unique, appelée URL (Uniform Resource Locator).

Un fichier HTML est un simple fichier texte contenant des **balises** permettant de mettre en forme le texte, les images ...

Une balise est une commande (un nom) encadrée par le caractère inférieur (<) et le caractère supérieur (>) par exemple "*<h1>*". Les balises seront instanciées en minuscule.

Cette norme décrit les différents éléments HTML utilisables dans une page et suit les spécifications du HTML 4.01 (http://www.w3.org/TR/html401/).

2.2 DOCTYPE

Il s'agit en fait d'une ligne de *déclaration du type de document*, qui indique au navigateur dans quel type de HTML la page a été écrite (HTML-3.2 «classique», HTML-4 de transition ou strict, XHTML, etc...). Dans une écriture plus complète, cette ligne a l'allure suivante :

```
<!DOCTYPE HTML PUBLIC   "type_de_HTML"   "adresse_de_DTD">
```

où

type_de_HTML est *l'identificateur* de la version du HTML utilisé.

et où *adresse_de_DTD* donne l'URL de la «document type declaration» (DTD), à savoir un document spécifiant les propriétés de chaque élément, balises et attributs, de ce type de HTML.

Par exemple, on pourra rencontrer la déclaration :

```
<!DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

Le tableau ci-dessous donne une liste de quelques uns des éléments possibles :

type_de_HTML	adresse_de_DTD
-//W3C//DTD HTML 4.01 Transitional//EN	http://www.w3.org/TR/html4/loose.dtd
-//W3C//DTD HTML 4.01//EN	http://www.w3.org/TR/html4/strict.dtd
-//W3C//DTD HTML 4.01 Frameset//EN	http://www.w3.org/TR/html4/frameset.dtd

-//W3C//DTD XHTML 1.0 Strict//EN	http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd
-//W3C//DTD XHTML 1.0 Transitional//EN	http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd
-//W3C//DTD XHTML 1.0 Frameset//EN	http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd

Tableau 1 : types de déclaration de document

Cette balise sert avant tout à faire passer les pages au validateur du consortium W3C, afin de vérifier que les pages produites ne contiennent pas d'erreur sur la syntaxe du HTML.

Côté navigateur, cette ligne n'est prise en compte que par les navigateurs les plus récents (comme Mozilla, NN7, Opera7, IE5 sur Mac ou IE6 sur Windows). En principe, cela permet un fonctionnement plus propre du navigateur et une assurance d'un comportement uniformisé pour l'ensemble des navigateurs du marché.

Ces types de déclaration de document sont à proscrire dans le cadre de ce projet, il est souhaité de préférencier l'utilisation des types de déclaration liés au XHTML plus contraignantes dans l'implémentation d'une page Web (voir §4.2.2) tout en gardant l'ensemble des possibilités du langage HTML explicité dans les parties ci-dessous.

2.3 GESTION DES FENETRES ET DES CADRES

Un document HTML peut être affiché dans une fenêtre ou un cadre (subdivision d'une fenêtre).

Il est nécessaire de limiter tant que possible le nombre de cadres (Frames ou IFrames) et fenêtres de l'application. Les mises à jours des différents cadres et des différentes fenêtres sont délicates et de nombreuses anomalies peuvent découler de cette gestion. Cette limitation est déjà définie à l'aide du framework Ergonomique.

2.3.1 LES CADRES (FRAMES)

Les cadres (frames) permettent de fractionner la surface d'affichage en plusieurs parties afin d'afficher différentes pages HTML simultanément dans le navigateur.

Ces cadres servent notamment à afficher d'un côté une série de liens comme un menu ou un sommaire et de l'autre côté des pages contenant des informations en relation.

La commande **<frameset>** marque le début de la définition des cadres.

La commande **</frameset>** marque la fin de la définition des cadres.

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="..." />
    <meta http-equiv="Date" content="01/01/2000" />
    ...
  </head>
  <frameset>
    ...
    <body>
      ...
```

```
      Diverses commandes HTML
      ...
    </body>
  </frameset>
</html>
```

Les cadres sont créés au sein des balises *<frameset>*.

La balise fabriquant ces cadres possèdent deux attributs essentiels, un spécifiant la page cible et l'autre définissant un nom pour le cadre.

La commande **<frame>** crée un cadre dans le document HTML.

L'attribut **src=*"Adresse du contenu"*** définit le contenu du cadre.

L'attribut **name="Nom du cadre"*** permet de nommer le cadre.

L'attribut **longdesc=*"Adresse du document"*** spécifie un document définissant le contenu du cadre.

* Lorsqu'une page contient des cadres, il est impératif de les nommer pour un fonctionnement optimal avec des liens.

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="..." />
    <meta http-equiv="Date" content="01/01/2000" />
    ...
  </head>
  <frameset>
    <frame src="Adresse du document" name="Nom du cadre" />
      ...
    <body>
      ...
      Diverses commandes HTML
      ...
    </body>
  </frameset>
</html>
```

Les balises *<noframe>* permettent de déclarer à l'intérieur d'une structure de cadres une zone n'en faisant pas partie.

Cette zone permet notamment d'afficher un contenu dans le cas ou un navigateur ne prendrait pas en charge la gestion des cadres.
D'autre-part, cette zone permet de renseigner les robots indexeurs sur le contenu des documents compris dans les cadres en vue du référencement dans les moteurs de recherche.

La commande **<noframe>** exclut le corps de la définition des cadres.

La commande **</noframe>** termine l'exclusion de la structure de cadres.

* Lorsqu'une page contient des cadres, il est impératif de les nommer pour un fonctionnement optimal avec des liens.

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="..." />
    <meta http-equiv="Date" content="01/01/2000" />
    ...
  </head>
  <frameset>
    <frame src="Adresse du document" name="Nom du cadre" />
      ...
    <noframe>
      <body>
        ...
        Diverses commandes HTML
        ...
      </body>
    </noframe>
  </frameset>
</html>
```

2.3.2 LES DIMENSIONS

Afin d'agencer correctement les cadres sur la page, la balise *<frameset>* comporte deux attributs permettant de définir des dimensions pour chacun des cadres.

Evidemment, dans un environnement de cadres, la division se fera soit verticalement soit horizontalement, les deux ensemble étant impossible.

Pour opérer une division horizontale et verticale, il faudra créer deux *<frameset>* imbriqués l'un dans l'autre. Les deux attributs acceptent des valeurs en pourcentage ou en pixels. Une étoile peut remplacer une des valeurs et constitue, donc, une dimension variable.

L'attribut **cols="Largeur1,Largeur2,..."*** définit une largeur pour les cadres.

L'attribut **rows="Hauteur1,Hauteur2,..."*** définit une hauteur pour les cadres.

*Dimensions	Description
<frameset cols="33%,33%,34%"> <frame src="..." name="Cadre1"> <frame src="..." name="Cadre2"> <frame src="..." name="Cadre3"> </frameset>	Division verticale en trois parties égales
<frameset rows="20%,80%"> <frame src="..." name="Cadre1">	Division horizontale en deux parties

<frame src="..." name="Cadre2"> </frameset>	
<frameset cols="240,400"> <frame src="..." name="Cadre1"> <frame src="..." name="Cadre2"> </frameset>	Division horizontale en deux parties sur 640 pixels
<frameset rows="120,360"> <frame src="..." name="Cadre1"> <frame src="..." name="Cadre2"> </frameset>	Division horizontale en deux parties sur 480 pixels
<frameset cols="15%,*"> <frame src="..." name="Cadre1"> <frame src="..." name="Cadre2"> </frameset>	Division verticale en deux parties, une de 15% et l'autre du reste
<frameset rows="200,*"> <frame src="..." name="Cadre1"> <frame src="..." name="Cadre2"> </frameset>	Division horizontale en deux parties, une de 200 pixels et l'autre du reste
<frameset cols="15%,30%,*"> <frame src="..." name="Cadre1"> <frame src="..." name="Cadre2"> <frame src="..." name="Cadre3"> </frameset>	Division verticale en trois parties, deux de 15% et 30% et l'autre du reste
<frameset rows="150,*,150"> <frame src="..." name="Cadre1"> <frame src="..." name="Cadre2"> <frame src="..." name="Cadre3"> </frameset>	Division horizontale en trois parties, deux de 150 pixels et l'autre du reste
<frameset cols="75%,*"> <frameset rows="150,*,150"> <frame src="..." name="Cadre1"> <frame src="..." name="Cadre2"> <frame src="..." name="Cadre3"> </frameset> <frame src="..." name="Cadre4"> </frameset>	Division verticale en deux parties. Division horizontale en trois parties du premier cadre
<frameset rows="25%,*,25%"> <frameset cols="50%,50%"> <frame src="..." name="Cadre1">	Division horizontale en trois parties, deux de 25% et l'autre du reste. Division horizontale du premier cadre en deux parties égales

<frame src="..." name="Cadre2"> </frameset> <frame src="..." name="Cadre3"> <frameset cols="200,*,100"> <frame src="..." name="Cadre4"> <frame src="..." name="Cadre5"> <frame src="..." name="Cadre6"> </frameset> </frameset>	égales. Division horizontale du dernier cadre en trois parties

Tableau 2 : exemples de dimensionnement pour la balise Frameset

D'autres attributs de la balise *<frame>* permettent d'une part d'interdire à l'utilisateur de redimensionner un cadre et d'autre part de donner la possibilité d'afficher une barre de défilement si le cadre est trop étroit.

Ces attributs sont également optionnels, par défaut les navigateurs autorise le redimensionnement et affiche la barre de défilement si le besoin est présent.

L'attribut **noresize** spécifie que le cadre ne peut être redimensionné.

L'attribut **scrolling="yes/no/auto"*** détermine selon le choix la présence d'une barre de défilement.

*Choix	Description
Yes	La barre de défilement est toujours visible sur le cadre.
No	La barre de défilement n'est jamais visible.
Auto	Le navigateur détermine si la barre de défilement est nécessaire.

Tableau 3 : valeurs de l'attribut scrolling

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="..." />
    <meta http-equiv="Date" content="01/01/2000" />
    ...
  </head>
  <frameset>
    <frame noresize
                   scrolling="yes/no/auto"
                   src="Adresse du document"
                   name="Nom du cadre">
    ...
    <noframe>
      <body>
        ...
        Diverses commandes HTML
        ...
      </body>
```

```
    <noframe>
  </frameset>
</html>
```

2.3.3 LES BORDURES

Un attribut permet aux cadres de créer une bordure autour d'eux. D'autres appliquent une couleur à cette bordure ou en fixe la taille.

L'attribut **frameborder="Valeur"*** détermine selon le choix l'affichage d'une bordure en trois-dimensions ou standard.

L'attribut **border="Valeur en pixel"**** définit la largeur de la bordure pour les cadres.

L'attribut **bordercolor="couleur"** définit une couleur de bordure.

*Choix Netscape	*Choix Microsoft	Description
Yes	**1**	Le cadre a une bordure en 3D (valeur par défaut).
No	**0**	Le cadre a une bordure standard.

Tableau 4 : rendus du cadre pour les Frames

** La valeur par défaut de la bordure est de 5 pixels, 0 rend la bordure invisible.

*Dimensions	Description
<frameset rows="80,*" border="0"> <frame src="..." name="Cadre1"> <frame src="..." name="Cadre2"> </frameset>	Cadres simples sans bordures
<frameset rows="80,*" border="10" bordercolor="#FF00FF"> <frame src="..." name="Cadre1"> <frame src="..." name="Cadre2"> </frameset>	Cadres simples avec bordures en couleur et en 3D
<frameset rows="25%,*,25%"> <frameset cols="50%,50%" border="20" bordercolor="#FF0000"> <frame src="..." name="Cadre1"> <frame src="..." name="Cadre2"> </frameset> <frame src="..." name="Cadre3"	La première sous-division possède une bordure de 20 pixels d'une couleur rouge. La seconde sous-division a des bordures invisibles. Tandis que la division principale de cadres garde des valeurs par défaut que chaque division pourra définir différemment, hormis en ce qui concerne le cadre 3 pour lequel une couleur de bordure verte a été définie

bordercolor="#008000"> <frameset cols="200,*,100" border="0" frameborder="no"> <frame src="..." name="Cadre4"> <frame src="..." name="Cadre5"> <frame src="..." name="Cadre6"> </frameset> </frameset>	

Tableau 5 : exemples de dimensionnement pour la bordure des Frames

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="..." />
    <meta http-equiv="Date" content="01/01/2000" />
    ...
  </head>
  <frameset border="Valeur en pixel"
            frameborder="yes/no"
            bordercolor="couleur">
    <frame frameborder="yes/no"
           bordercolor="couleur"
           src="Adresse du document"
           name="Nom du cadre">
    ...
    <noframe>
      <body>
        ...
        Diverses commandes HTML
        ...
      </body>
    <noframe>
  </frameset>
</html>
```

2.3.4 LES MARGES

Plusieurs attributs de la balise *<frame>* permettent de spécifier des marges autour des cadres concernés ou de définir un espace entre eux.

Ces attributs sont optionnels, par défaut les navigateurs déterminent la grandeur appropriée pour la marge.

L'attribut **marginwidth="Valeur en pixel"** définit les marges de gauche et de droite des cadres.

L'attribut **marginheight="Valeur en pixel"** définit les marges du haut et du bas des cadres.

L'attribut **frameborder="Valeur en pixel"*** définit un espace entre les cadres.

* Cet attribut est utilisable également sous la balise *frameset*.

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="..." />
    <meta http-equiv="Date" content="01/01/2000" />
    ...
  </head>
  <frameset>
    <frame marginwidth="Valeur en pixel"
           marginheight="Valeur en pixel"
           framespacing="Valeur en pixel"
           src="Adresse du document"
           name="Nom du cadre">
    ...
    <noframe>
      <body>
        ...
        Diverses commandes HTML
        ...
      </body>
    </noframe>
  </frameset>
</html>
```

2.3.5 LES CADRES UNIQUES

Une commande permet d'insérer un cadre unique dans une page Web en s'insérant directement dans le corps du document.

Ce cadre unique fait référence à un autre document HTML que l'on peut afficher à l'intérieur d'un espace généré dans une page web.

La commande **<iframe>** affiche un cadre au sein d'un document.

La commande **</iframe>** termine l'instruction "cadre unique".

L'attribut **src="Adresse du cadre"** définit le contenu du cadre.

L'attribut **name="Nom du cadre"** permet de nommer le cadre.

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="..." />
    <meta http-equiv="Date" content="01/01/2000" />
    ...
```

```
  </head>
  <body>
    <iframe src="Adresse du document"
            name="Nom du cadre">
    </iframe>
    ...
    Diverses commandes HTML
    ...
  </body>
</html>
```

La balise possède les mêmes attributs cités plus bas que la commande *<frame>* en plus des siens propres.

L'ensemble de ces attributs a été vu avec les images et fonctionnent exactement de la même façon, hormis que l'élément graphique est remplacé par un cadre.

L'attribut **align="Type d'alignement"** spécifie un alignement du cadre dans le document.

L'attribut **width="Valeur en pixel"** définit la largeur du cadre, en pixels.

L'attribut **height="Valeur en pixel"** définit la hauteur du cadre, en pixels.

L'attribut **hspace="Valeur en pixel"** définit un espace horizontal.

L'attribut **vspace="Valeur en pixel"** définit un espace vertical.

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="..." />
    <meta http-equiv="Date" content="01/01/2000" />
    ...
  </head>
  <body>
    <iframe align="Type d'alignement"
            height="Valeur en pixel"
            width="Valeur en pixel"
            hspace="Valeur en pixel"
            vspace="Valeur en pixel"
            src="Adresse du document"
            name="Nom du cadre">
    </iframe>
    ...
    Diverses commandes HTML
    ...
  </body>
</html>
```

2.4 STRUCTURE GENERALE D'UNE PAGE HTML

Toute page HTML est constituée d'une zone d'entête (Elément <head></head>) et d'un corps (Elément <body></body>).

Exemple d'implémentation :

```
<html>
        <head>
                Zone d'entête…
        </head>
        <body>
                Corps de la page…
        </body>
</html>
```

2.4.1 ZONE D'ENTETE

Un document HTML doit comporter un en-tête permettant de regrouper des informations indispensables

pour le référencement sur la page.

Ces informations sont appellées des "meta-informations".

Ces informations peuvent être utilisées par les différents moteurs de recherche afin de déterminer les caractéristiques du document, comme le titre, les mots-clés, l'auteur, en vue de l'indexation du site dans leur base de données.

Il ne peut y avoir qu'un seul **<head>** par document. Il doit suivre la balise **<html>** ouvrante et précéder le **<body>**

A l'intérieur de l'en-tête, on trouve le titre du document, qui est défini dans la balise **<title>**, ainsi que d'autres informations définies dans les balises **<meta>**, **<base>**, etc... qui s'appliqueront à l'ensemble de la page (voir topo suivant).

L'élément head est un élément très important du langage HTML. Il permet de caractériser le document qu'il introduit sur différents points pour en permettre une utilisation optimum au niveau du couple serveur/utilisateur.

D'un point de vue formel, cet élément n'a pas d'attributs, mais des éléments apparentés qui lui sont rattachés. Il fait l'objet d'une évolution constante; depuis la version 2.0, des éléments lui ont été ajoutés qui font encore aujourd'hui l'objet de discussions au sein du groupe de travail.

Nous conseillons donc d'utiliser exclusivement les balises détaillées ci-dessous pour être conforme au standard actuel.

De tous ces éléments, le seul qui est requis est le titre (title); les autres sont purement optionnels.

La commande **<head>** marque le début de l'entête.

La commande **</head>** marque la fin de l'entête.

Seuls les tags suivants sont permis dans la zone d'en-tête :

Tag	Description
<script>	Conteneur d'un script exécutable par le navigateur
<title>	Confère un titre au document; seul paramètre obligatoire
<base>	Spécifie l'adresse de base du document HTML

<link>	Décrit les liens entre le document et d'autres documents
<meta>	Fournit des informations utiles serveur/navigateur
<style>	Conteneur d'une déclaraton de style : permet de définir et paramétrer le style général du document HTML

Tableau 6 : balises autorisées dans l'en-tête

La balise link

Il indique la présence d'un lien entre le document et un ou plusieurs autres pouvant être utilisés par un outil d'automatisation quelconque, par opposition à **<a>**, qui marque une action de la part de l'utilisateur dans le corps de la page.

Les utilisations privilégiées de <LINK> sont :

- indication de l'auteur avec renvoi à son adresse électronique :

```
<link rev="made" href="mailto:link@ministere.fr" />
```

- identification du titre :

```
<link title="Titre du document" href="http://une.adresse.fr/index.htm" />
```

- indication d'un lien entre le document courant et un autre document :

```
<link rel="prev" href="intro.htm" />
```

indique que le document courant est précèdé par le document intro.htm.

```
<link rel="next" href="suite.htm" />
```

indique que le document courant précède le document suite.htm

- liste des différentes versions :

```
<link rel="history" href="versions.htm" />
```

indique que le document versions.htm contient l'historique des versions du document courant

- appel d'une feuille de style :

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

indique que le document style.css est une feuille de style à charger pour formater le document courant

Dans le cadre des applicatifs du Ministère des Affaires Etrangères, il est permis d'utiliser cette balise uniquement. Pour les autres versions de la balise une demande explicite doit être faite.

Href	Définit un objet à l'aide de la notation URL (Uniform Resource Locator).	
	Syntaxe	<LINK href=".....">
Rel	Définit un type de relation qui a sa source dans le document courant.	
	Paramètres	Alternate - Author - Bookmark - Contents - Copyright - Glossary - Index - Help - Next - Previous - Start - Stylesheet
	Syntaxe	<LINK rel=".....">
Rev	Définit une relation inverse et spécifie que la liaison à sa destination dans le document courant.	
	Paramètres	Alternate - Author - Bookmark - Contents - Copyright - Glossary - Index - Help - Next -

		Previous - Start - Stylesheet
	Syntaxe	<LINK rev=".....">
Title	Définit un titre pour un document spécifié.	
	Syntaxe	<LINK title=".....">
Type	Spécifie le type MIME du document lié.	
	Syntaxe	<LINK type="text/css"> Définit le type MIME d'une feuille de style externe, le lien étant établi avec un fichier "style.css" .

Tableau 7 : attributs de la balise link

La balise base

Plutôt que d'attribuer le même attribut TARGET à tous les liens, la balise <base> permet de fixer cela une fois pour toute.

Exemple :

```
<base target="fenetre_principale" />
```

L'ensemble des liens de la page seront alors ouvert dans la frame qui est définie (ici fenetre_principale).

Il est également possible d'utiliser cette balise pour définir une arborescence par défaut ou un titre sur l'ensemble des liens.

Exemple :

```
<base href="http://www.unepage.fr/" title="Une page quelconque" />
```

Ensuite dans le corps de page le lien suivant :

```
<a href="index.html"></a>
```

mène vers http://www.unepage.fr/index.html et affiche une petite info-bulle "Une page quelconque" lors de son survol.

La balise meta

La balise <meta> permet normalement de fournir une description du site par le biais de mots clés et de phrases décrivant le site, ceci est donc l'utilisation la plus courante de cette balise mais elle a d'autres fonctions, comme notamment la redirection automatique vers une URL.

Attribut HTTP-EQUIV

Avec la valeur refresh on peut recharger une page toutes les n secondes. Cette fonction est identique à l'action actualiser (reload) de votre navigateur.

```
<meta http-equiv="refresh" content="60" />
```

Dans cet exemple, la page se recharge toutes les 60 secondes.

On peut aussi charger un autre document après n secondes.

```
<meta http-equiv="refresh" content="60; url="http://www.site.fr" />
```

Avec la valeur "expires", il est possible de définir une date limite aux pages et donc d'indiquer au serveur proxy qu'il doit impérativement recharger la page originale à la date indiquée.

```
<meta http-equiv="expires" content="Wed, 30 Sept 1998 12:00:00 GMT" />
```

Avec les valeurs pragma et no-cache il est possible d'interdire de mettre "en cache" les pages.

```
<meta http-equiv="pragma" content="no-cache" />
```

Pour plus de détail sur la gestion du cache et suivre des préconnisations d'utilisation pour prendre en compte l'ensemble des problématiques sous-jacentes, se référer à l'Annexe K : Gestion du cache.

Attributs Keywords, description, Author, et Lang

Ces attributs permettent d'indexer le site, ou plus justement, faciliter la tache du robot qui viendra visiter la page mentionnée lors de l'inscription sur le formulaire du moteur de recherche.

Voici un exemple d'indexation.

A noter que ces balises **<meta>** doivent être placées juste après le titre de la page (balise **<title>**).

- Nom de l'auteur du site et de la langue utilisée :

```
<meta name="Author" lang="fr" content="PRENOM NOM" />
```

- Description du site en une phrase de préférence :

```
<meta name="Description" content="description du site" />
```

- Une liste de mots clés correspondant au mieux au site :

```
<meta name="Keywords" lang="fr" content="renseigner les mots ou expressions clés
(espacés d'une virgule) qui correspondent au site" />
```

- L'URL complète du site :

```
<meta name="Identifier-URL" content="http://www.ministere.fr" />
```

- Le nom du responsable de publication :

```
<meta name="Publisher" content="PRENOM NOM" />
```

- Copyrights :

```
<meta name="Copyright" content="COPYRIGHT NOM ..." />
```

- Les outils de développement utilisés :

```
<meta name="Generator" content="WebExpert, FTP Expert, PSP 4.1" />
```

- Cette balise identifie le code ISO en vigueur :

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
```

Diriger les robots de référencement :

Avec la valeur robots il est possible d'indiquer aux robots de référencement comment référencer les pages. Cette valeur possède plusieurs attributs dont :

- All : Le robot indexe tout (par défaut).
- None : Stoppe le robot.
- Index : Votre page est indexée.
- Noindex : Votre page n'est pas indexée mais il suivra les liens de cette page.
- Follow : Le robot récupère les liens de votre page pour les référencer plus tard.
- Nofollow : Le robot ne récupère pas les liens de votre page.

Exemple d'implémentation :

```
<meta name="robots" content="index" />
```

La balise style

Crée une feuille de style **interne**. Cette balise est donc à banir dans le cadre du FrameWork ergonomique puisque la définition de ses styles sera faite dans une feuille de style **externe**.

D'autre part cette façon d'opérée s'avèrerait dangereuse en terme de respect de la présentation puisque les styles de éléments ainsi défini auraient une priorité plus haute que ceux définis dans la feuille de style externe.

syntaxe	<style> ... </style> style est un élément fermé.
type	Type définit le langage de styles employé dans le document. Syntaxe: type="...." Exemple: type="text/css"
title	Title confère un titre au style employé dans le document. Syntaxe: title="...." Exemple: title="monstyle"

Tableau 8 : attributs de la balise style

L'utilisation de cette balise est normalisée dans le §3.2.

La balise script

L'élément <script> contient des scripts côtés client qui sont exécutés par le navigateur. L'utilisation de cette balise est normalisée dans le §5.3.2.

2.4.2 CORPS DE LA PAGE

Le corps de la page contient tous les éléments visibles à partir d'un navigateur : les liens, le texte, les images, les éléments multimédias, les formulaires, etc.

La commande **<body>** marque le début du corps.

La commande **</body>** marque la fin de du corps.

Dans le standard HTML 4 élaboré par le W3C, les attributs possible dans la balise, sont ceux permettant la gestion des événements, les attributs classiques *lang* et *dir* et *title*, ainsi que les attributs liés aux feuilles de style (CSS), c'est-à-dire, les commandes *class*, *style* et *id*.

La plupart des attributs cités dans cette page, relève des spécifications du HTML 3.2 et est donc caduque dans les nouvelles normes, le formatage de la page s'effectuant désormais par l'intermédiaire des feuilles de style. Ils sont juste rappelés ci-dessous pour ne pas oublier de les renseigner dans la feuille de style.

Attributs de fond et de couleur de texte

Il est possible par des attributs de déterminer la couleur ou (et) une image de fond pour une page HTML et en outre celle du texte. Ces couleurs seront vos valeurs par défaut pour l'ensemble de la page.

L'attribut **bgcolor="Couleur"*** donne une couleur de fond au document HTML.

L'attribut **text="Couleur"*** donne une couleur au texte.

L'attribut **background=*"Adresse de l'image"*** permet l'affichage d'une image en arrière plan.

L'attribut **bgproperties="fixed"** empêche l'image précitée de défiler avec le contenu du document.

Exemple d'implémentation :

```
<html>
        <head>
                <title>Un titre pertinent</title>
                <meta name="Description" content="…" />
                <meta http-equiv="Date" content="01/01/2000" />
                …
        </head>
        <body background="Adresse de l'image" bgcolor="Couleur" text="Couleur"
bgproperties="fixed">
        …
        Diverses commandes HTML
        …
        </body>
</html>
```

La couleur des liens

De la même façon, des attributs permettent la mise en couleur des liens dans différentes situations.

Sinon, lorsque le lien n'est pas activé, il peut prendre une couleur différente de celle spécifiée par l'attribut correspondant par l'intermédiaire de la balise *<font…>*

L'attribut **link="Couleur"*** donne une couleur pour un lien non-activé.

L'attribut **alink="Couleur"*** donne une couleur pour un lien actif.

L'attribut **vlink="Couleur"*** donne une couleur pour un lien déjà activé.

Exemple d'implémentation :

```
<html>
        <head>
                <title>Un titre pertinent</title>
                <meta name="Description" content="…">
                <meta http-equiv="Date" content="01/01/2000">
                …
        </head>
        <body link="Couleur" alink="Couleur" vlink="Couleur"">
        …
        Diverses commandes HTML
        …
        </body>
</html>
```

Les marges

Les quatres premiers attributs sont propres à Internet Explorer et déterminent des marges en haut et à gauche ou en bas et à droite du document.
Les deux derniers attributs permettent de définir des marges en largeur et en hauteur et ne fonctionnent que sous Netscape.

L'attribut **topmargin="Valeur en pixel"** définit une marge en haut du document.

L'attribut **bottommargin="Valeur en pixel"** définit une marge en bas du document.

L'attribut **leftmargin="Valeur en pixel"** définit une marge à gauche du document.

L'attribut **rightmargin="Valeur en pixel"** définit une marge à droite du document.

L'attribut **marginwidth="Valeur en pixel"** définit une marge dans la largeur du document.

L'attribut **marginheight="Valeur en pixel"** définit une marge dans la hauteur du document.

Exemple d'implémentation :

```
<html>
        <head>
                <title>Un titre pertinent</title>
                <meta name="Description" content="…">
                <meta http-equiv="Date" content="01/01/2000">
                …
        </head>
        <body bottonmmargin="Valeur en pixel" topmargin="Valeur en pixel"
leftmargin="Valeur en pixel" rightmargin="Valeur en pixel" marginwidth="Valeur en
pixel" marginheight="Valeur en pixel">
        …
        Diverses commandes HTML
        …
        </body>
</html>
```

2.5 GESTION DU TEXTE ET DES IMAGES

2.5.1 GESTION DU TEXTE

Certaines balises permettent d'utiliser les styles pour gérer les différences d'affichage des paragraphes et des polices.

*Balise *

Pour inclure dans un passage commun plusieurs paragraphes comportant des éléments très différents comme du texte, des graphiques, des tableaux, etc...

** ** marque le début d'un passage dans lequel on peut inclure plusieurs éléments (*div = division = passage*). Tout ce qui se trouve entre ce repère et le repère de fermeture ** ** est interprété comme partie intégrante du passage.

Attributs utilisés :

Propriétés	Descriptif	Valeurs
align	alignement vertical ou horizontal	%|align
class	assigne une classe ou un ensemble de classes au passage	data

Tableau 9 : attributs de la balise DIV

Exemples d'implémentation :

```
<td width="185" class="classe1">
<div class="classe2" align="center">Type</div>
</td>
```

```
<td width="130" class="classe1">
<div align="center">03980 30000000076</div>
</td>
```

Afin d'avoir un code HTML compatible avec la cible des navigateurs, il est préférable de positionner l'attribut class dans le td plutôt que dans le div.

Balise

L'élément **span** est un élément "en ligne" et peut être utilisé à l'intérieur de paragraphes, éléments de liste, etc. lorsque l'on souhaite assigner une déclaration de classe ou des informations de langue à un groupe de mots.

L'élément span ne peut être utilisé pour réunir un groupe d'éléments de niveau "bloc". span n'a aucun effet sur l'apparence tant qu'un style ne lui est pas appliqué, par exemple, via un attribut class.

Exemple d'implémentation :

```
<td width="124" class="tableauFrameEcriture1">
<div align="right">+2697.84 <span class="monnaie">EUR</span></div>
</td>
```

Balise <p>

<p> (*p = paragraphe*) insère un changement de paragraphe. Le repère peut aussi bien se trouver à la fin du paragraphe précédent qu'au début d'une ligne distincte ou au début du paragraphe suivant.

Exemple d'implémentation :

```
<p>Ici commence un paragraphe et ici il se termine.</p>
```

Balise

**
** (*br = break = cassure*) insère à l'endroit désiré un passage à la ligne. Peu importe qu'il soit placé à la fin de la ligne précédente, ou sur une ligne distincte ou encore au début de la ligne suivante.

Ici se termine une ligne.

Ici commence une nouvelle ligne.

2.5.2 GESTION DES IMAGES

Balise <map>

Des graphiques composés de liens sont des graphiques dans lesquels l'utilisateur peut cliquer sur un détail. Il s'ensuit l'exécution d'un lien. Par ce moyen l'utilisateur peut dans certains cas parvenir à l'information beaucoup plus intuitivement et plus vite que par une longue liste prolixe de liens.

<map name=...> introduit la définition des surfaces composées de liens d'un graphique

<area...> définit des surfaces d'un certain graphique associées à des liens que l'on incorpore à un autre endroit.

L'attribut **usemap=** sert à marquer le graphique comme étant composé de liens

L'attribut **href=** détermine la cible à savoir le fichier qui doit être appelé quand l'utilisateur clique sur la surface associée au lien.

Balise <img>

La balise **<img ...>** est utilisée pour incorporer une image graphique dans un document HTML.

L'applicatif peut gèrer certaines images d'extension « .gif » et « .jpg » fournies par une Web Agency.

Propriétés	Descriptif	Valeurs
alt=	Affiche un message à la place d'une image si celle-ci n'a pu être chargée ou lue	data
ismap	Définit une image comme étant une image réactive, côté serveur	-
src=	Indique l'URL ou le fichier source de l'image	url
align=	Permet d'aligner de différentes manières texte et images dans un document HTML	top|middle|bottom
usemap=	Définit une image comme étant une image réactive, côté navigateur	image.map
width=	Permet de définir la largeur d'affichage (en pixels) d'une image importée	% ou pixels
height=	Permet de définir la hauteur d'affichage (en pixels) d'une image importée	% ou pixels
border=	alignement vertical ou horizontal	pixels
vspace=	hauteur suggérée	pixels
hspace=	largeur suggérée	pixels

Tableau 10 : attributs de la balise IMG

Exemple d'implémentation :

```
<img src="/images/consultation/liste.gif" width="590" height="35" usemap="#Map2"
border="0">
```

2.6 GESTION DES TABLEAUX

2.6.1 LA BALISE <TABLE>

Dans une page Web, le tableau demeure un élément indispensable. La conception d'un document HTML élaboré nécessite d'une façon quasi-systématique le recours aux tableaux.

En effet, un tableau est utilisé non seulement comme contenant d'informations, de données, de renseignements divers parfaitement agencés et structurés permettant une lecture claire, systématique ou méthodique.

Dans une page HTML, le tableau permet également de disposer sur le document différents éléments tels que du texte ou des images dans un but de présentation, ou de lisibilité.

Par sa puissance fonctionnelle, mais aussi par la discrétion dont il sait faire preuve, le tableau prend souvent une place centrale dans tous les documents HTML complexes.

L'utilisation pertinente d'un tableau résout dans de nombreux cas des situations insolubles.

Il est tout de même important de limiter tant que possible le nombre de tableaux imbriqués. Certains navigateurs rencontrent de gros problèmes à l'affichage des pages présentant plus de 4 tableaux imbriqués. Jusqu'à 3 tableaux imbriqués, les performances des navigateurs sont toujours bonnes. A partir de 4 tableaux imbriqués, des lenteurs à l'interprétation de la page se font sentir.

La balise **<table>** marque le début d'un tableau.

La balise **</table>** marque la fin d'un tableau.

2.6.2 LA STRUCTURE

Un tableau peut être composé de trois sous-parties permettant de distinguer respectivement, la partie supérieure, le corps et la partie inférieure d'un tableau.

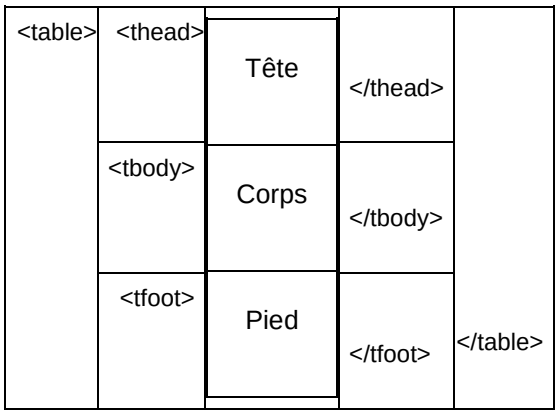


Figure 1 : représentation des parties constituant un tableau

Dans un tableau, le partie supérieure constitue la tête du tableau. Elle peut être composée de plusieurs lignes et cellules.

La balise **<thead>** marque le début d'une tête de tableau.

La balise **</thead>** marque la fin d'une tête de tableau.

Un tableau est constitué d'un corps formé par des cellules et constituant la partie recueillant les données.

La balise **<tbody>** marque le début du corps d'un tableau.

La balise **</tbody>** marque la fin du corps d'un tableau.

Au sein d'un tableau, la partie inférieure constitue le pied du tableau. Elle peut être composée de plusieurs lignes et cellules.

La balise **<tfoot>** marque le début du pied d'un tableau.

La balise **</tfoot>** marque la fin du pied d'un tableau.

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="..." />
    <meta http-equiv="Date" content="01/01/2000" />
    ...
  </head>
  <body>
    <table>
      <thead>
        ... Tête de tableau
      </thead>
      <tbody>
        ... Corps de tableau
      </tbody>
      <tfoot>
        ... Pied de tableau
      </tfoot>
    </table>
    ...
    Diverses commandes HTML
    ...
  </body>
</html>
```

L'illustration ci-dessous vous décrit un tableau à cinq colonnes comportant une cellule fusionnée pour accueillir le titre, un en-tête composé de deux cellules, cinq lignes de données constituées également de deux cellules chacune et enfin d'un pied composé d'une cellule fusionnée pour fournir un état du tableau (Ex : total …).

<table>		Première colonne	Seconde colonne	
Titre	**<caption>**	...Titre tableau...		**</caption>**

En-tête	**<thead>** **<tr>**	**<td>** **...** **</td>**	**<td>** **...** **</td>**	**</tr>** **</thead>**
Début corps	**<tbody>**			
Première ligne du corps	**<tr>**	**<td>** **...** **</td>**	**<td>** **...** **</td>**	**</tr>**
Seconde ligne du corps	**<tr>**	**<td>** **...** **</td>**	**<td>** **...** **</td>**	**</tr>**
Troisième ligne du corps	**<tr>**	**<td>** **...** **</td>**	**<td>** **...** **</td>**	**</tr>**
Quatrième ligne du corps	**<tr>**	**<td>** **...** **</td>**	**<td>** **...** **</td>**	**</tr>**
Cinquième ligne du corps	**<tr>**	**<td>** **...** **</td>**	**<td>** **...** **</td>**	**</tr>**
Fin corps				**</tbody>**

Pied	**<tfoot>** **<tr>**	**<td colspan="2">** **...** **</td>**	**</tr>** **</tfoot>**
			</table>

Figure 2 : illustration de l'instance d'un tableau

2.6.3 LES LIGNES DU CORPS

Un tableau est logiquement composé de colonnes et de lignes.

En conséquence, des balises spécifiques définissent des lignes qui contiendront de une à plusieurs cellules formant elle même par juxtaposition des colonnes.

La commande **<tr>** marque le début d'une ligne.

La commande **</tr>** marque la fin d'un ligne.

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="...">
    <meta http-equiv="Date" content="01/01/2000">
    ...
  </head>
  <body>
    <table>
      <caption>
        Légende du tableau
      </caption>
      <tr>
        Première ligne
      </tr>
      <tr>
        Seconde ligne
      </tr>
      ...
    </table>
    ...
    Diverses commandes HTML
    ...
  </body>
</html>
```

2.6.4 LES COLONNES ET LES CELLULES D'UNE LIGNE

Les colonnes d'un tableau sont déduites à partir du nombre de cellules d'une ligne (voir tableau exemple).

Néanmoins, un attribut de la balise de création d'un tableau permet de définir le nombre de colonnes du tableau afin d'accélérer le processus de chargement de la page Web.

Les cellules sont disposées exclusivement au sein d'une ligne. Les instructions de création de cellules sont toujours comprises entre les balises *<tr>* et *</tr>*.

La commande **<td>** marque le début d'une cellule.

La commande **</td>** marque la fin d'une cellule.

L'attribut **cols="Nombre de colonnes"** définit le nombre de colonnes (Attribut obsolète).

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="...">
    <meta http-equiv="Date" content="01/01/2000">
    ...
  </head>
  <body>
    <table>
      <tbody>
        <tr>
          <td>Première cellule</td>
          <td>Seconde cellule</td>
          ...
        </tr>
        <tr>
          <td>Première cellule</td>
          <td>Seconde cellule</td>
          ...
        </tr>
        ...
      </tbody>
    </table>
    ...
    Diverses commandes HTML
    ...
  </body>
</html>
```

2.6.5 L'EN-TETE ET PIED

Au sein d'un tableau, la ligne supérieure ou la première colonne constitue en général l'en-tête et la ligne inférieure ou la dernière colonne constitue en général le pied.

Des balises spéciales peuvent définir ce type de cellules pour l'en-tête afin de distinguer les informations contenues du reste des données.

La commande **<th>** marque le début d'un en-tête.

La commande **</th>** marque la fin d'un en-tête.

L'utilisation de cette balise s'oppose à celle définie dans la structure d'un tableau à l'aide des balises <thead>, <tbody> et <tfoot>. Pour cette raison et pour obtenir une structure de tableau normalisée au sein du MAE, il est préférable de se cantonner à l'implémentation de la balise <thead> et de proscrire le recours à la balise <th> à iso-périmètre.

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="...">
    <meta http-equiv="Date" content="01/01/2000">
    ...
  </head>
  <body>
    <table>
      <caption>
        Légende du tableau
      </caption>
      <thead>
        <tr>
          <td>Titre première colonne</td>
          <td>Titre seconde colonne</td>
          ...
        </tr>
      </thead>
      <tbody>
      ...
      </tbody>
      <tfoot>
        <tr>
          <td>Pied première colonne</td>
          <td>pied seconde colonne</td>
          ...
        </tr>
      </tfoot>
    </table>
    ...
    Diverses commandes HTML
    ...
  </body>
</html>
```

2.6.6 LES LEGENDES

Un tableau peut comporter une légende permettant de définir globalement son contenu.

Une balise placée entre les commandes *<table>* et *</table>* autorise l'assignation d'une légende au tableau.

La commande **<caption>** marque le début de la légende.

La commande **</caption>** marque la fin de la légende.

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="...">
    <meta http-equiv="Date" content="01/01/2000">
    ...
  </head>
  <body>
    <table>
      <caption>
        Légende du tableau
      </caption>
      <thead>
      ...
      </thead>
      <tbody>
      ...
      </tbody>
      <tfoot>
      ...

      </tfoot>
    </table>
    ...
    Diverses commandes HTML
    ...
  </body>
</html>
```

2.6.7 LES REGROUPEMENTS DE COLONNES

Les regroupements de colonnes permettent d'appliquer à des blocs de cellules des valeurs identiques au niveau des alignements horizontaux et verticaux ou de la largeur.

Les regroupements de colonnes peuvent se faire par deux balises citées plus bas. En effet, les attributs de largeur, d'alignements sont associés aux deux commandes. Mais ces dernières peuvent faire l'objet de combinaisons néanmoins déconseillées.

La commande <**col ...**> permet d'appliquer à des colonnes des caractéristiques identiques.

La commande <**colgroup ...**> définit le nombre de cellules à regrouper horizontalement.

La commande <**/colgroup**> définit le nombre de cellules à regrouper verticalement.

L'attribut **span="Nombre de colonnes"** définit le nombre de cellules à regrouper horizontalement.

Exemple d'implémentation :

```
<html>
<head>
  <title>Un titre pertinent</title>
  <meta name="Description" content="...">
  <meta http-equiv="Date" content="01/01/2000">
  ...
</head>
<body>
  <table>
    <caption>
      Légende du tableau
    </caption>
    <col span="Nombre de colonnes">
    <colgroup span="Nombre de colonnes">
    </colgroup>
    ...
    <thead>
      <tr>
        <td colspan="Nombre de cellules horizontales"
            rowspan="Nombre de cellules verticales">
          Titre colonne 1
        </td>
        <td>
          Titre colonne 2
        </td>
        ...
      </tr>
    </thead>
    <tbody>
      <tr>
        <td colspan="Nombre de cellules horizontales"
            rowspan="Nombre de cellules verticales">
          Première cellule
        </td>
        <td>Seconde cellule</td>
        ...
      </tr>
      ...
    </tbody>
  </table>
  ...
    Diverses commandes HTML
  ...
  </body>
</html>
```

2.6.8 LES ATTRIBUTS

2.6.8.1 LES ATTRIBUTS GENERIQUES

Enfin, plusieurs autres attributs variés sont employés au sein d'un tableau.

L'attribut **summary="Description"** crée un texte descriptif pour le tableau.

L'attribut **nowrap** désactive les changement de lignes.

L'attribut **char="Caractère"**** spécifie le caractère d'alignement du texte dans les cellules.

L'attribut **charoff="Valeur"** détermine la position où est placé le caractère d'alignement.

L'attribut **abbr="Abréviation"** détermine une abréviation associée au contenu de la cellule.

L'attribut **axis="Nom de l'axe"** définit un nom à un groupe de cellules afin de créer un axe imaginaire.

L'attribut **headers="liste"** énumère les noms de cellules d'en-tête associées à la cellule courante.

L'attribut **scope="Portée"***** définit la portée de la cellule d'en-tête concernée.

Ces attributs sont valables essentiellement pour les balises suivantes : ***col***, ***colgroup***, ***tbody***, ***td***, ***tfoot***, ***th***, ***thead***, ***tr***.

Le caractère d'alignement joue le rôle d'axe d'alignement pour la partie de texte concerné. Il peut avoir une valeur différente selon la langue utilisée dans le document HTML, par exemple un point (« . » : valeur par défaut) pour l'Anglais ou une virgule (« , ») pour le Français.

Les valeurs possibles de la portée sont ***row***, ***col***, ***colgroup*** ou ***rowgroup***.

2.6.8.2 LES ALIGNEMENTS

Dans un document HTML, il est possible d'appliquer un alignement au tableau lui permettant de se positionner au sein de la page. Un attribut, déjà bien connu, est utilisé au sein de la balise *<table>* pour un alignement horizontal.

L'attribut **align="Type d'alignement"** définit un alignement horizontal du tableau et des cellules.

`Type d'alignement= "right", "left" ou "center".`

Les options d'alignement horizontal sont « ***Left*** » (par défaut) pour la gauche, « ***right*** » pour la droite ou « ***center*** » pour le centre.

Dans un document HTML, il est possible d'appliquer un alignement des éléments au sein d'un tableau, à l'intérieur des cellules.

Deux attributs d'alignement sont utilisés au sein des balises *<td>*, *<th>* et enfin *<tr>* ainsi que *<caption>* pour un alignement horizontal et un autre permet un alignement vertical dans les trois premiéres commandes.

L'attribut **align="Type d'alignement"*** définit un alignement horizontal du tableau et des cellules.

L'attribut **valign="Type d'alignement"**** définit un alignement horizontal dans les lignes ou les cellules.

Les options d'alignement horizontal sont « ***Left*** » (par défaut) pour la gauche, « ***right*** » pour la droite ou « ***center*** » pour le centre.

Les options d'alignement vertical sont « ***middle*** » (par défaut) pour le milieu, « ***top*** » pour le haut ou « ***bottom*** » pour le bas ou « ***baseline*** » pour le bas de la ligne courante.

Tableau aligné au centre par <table align="center">

align="left"	align="center"	align="right"	align="right" valign="top"
valign="top"	valign="middle"	valign="bottom"	valign="baseline"
align="left" valign="top"	align="center" valign="middle"	align="right" valign="bottom"	align="left" valign="bottom"
align="center" valign="top"	align="right" valign="middle"	align="left" valign="middle"	align="center" valign="bottom"

Figure 3 : illustrations des attributs align et valign d'une cellule de tableau

2.6.8.3 LES DIMENSIONS

Tous les éléments du tableau, les colonnes, les en-têtes et les cellules et le tableau lui-même par rapport à la surface d'affichage, peuvent être redimensionnés indépendamment.

Des attributs identiques pour chaque élément précité permettent de régler en largeur ou en hauteur. D'ailleurs, ces attributs ont déjà été rencontrés dans la rubrique : Les dimensions des images, et fonctionnent exactement de la même manière.

L'attribut **width="Valeur"*** définit la largeur.

L'attribut **height="Valeur"*** définit la hauteur.

Les valeurs peuvent s'exprimer en pixels ou en pourcentage de la surface d'affichage : *width="250"* ou *width="80%"*.

Il est donc possible d'affecter les deux attributs de dimensionnement à pratiquement tous les éléments, c'est-à-dire *<table>* et *<td>* sauf la commande de création de lignes *<tr>*.

D'autre part, les éléments du tableau acceptant ces attributs ne réagissent pas de la même façon devant leurs spécifications.

La commande *<table>* effectue une modification globale du tableau en ne tenant compte que de la bordure extérieure.

Les balises d'*en-têtes* et de *cellules* s'occupent essentiellement de la largeur de la colonne dont elles font partie et de la hauteur de la ligne dont elles dépendent. Lorsqu'une valeur en hauteur ou en largeur est affectée à une cellule, c'est respectivement à la ligne et à la colonne, dans leur totalité, que seront appliquées les dimensions de la cellule. Ainsi, en dimensionnant une seule cellule par ligne ou par colonne, le reste des cellules s'alignera automatiquement.

Voici un exemple ou la cellule supérieure gauche a des dimensions spécifiées et agit sur la colonne rose et la ligne bleue entière tandis que les cellules grises n'en dépendant pas, conservent leurs dimensions par défaut.

	Première colonne	Seconde Colonne	Troisième colonne
Première ligne	**height="120" width="200"**	**Hauteur de 120 pixels**	**Hauteur de 120 pixels**
Seconde ligne	**Largeur de 200 pixels**		
Troisième ligne	**Largeur de 200 pixels**		

Tableau 11 : règles d'application du dimensionnement

2.6.8.4 LES ESPACEMENTS

Plusieurs attributs de *<table>* permettent de régler les espacements entre les données contenues dans les cellules mais aussi entre les cellules elles-mêmes.

L'attribut **cellpadding="Valeur"*** définit l'espace de remplissage entre les données et le quadrillage.

L'attribut **cellspacing="Valeur"*** définit l'espacement entre les cellules.

Les valeurs s'expriment en pixels : *border="3"*, *cellspacing="15"* ou *hspace="20"*.

2.6.8.5 LES BORDURES

Un tableau possède en principe une bordure et un quadrillage afin de séparer les informations contenues en son sein et également pour améliorer sa lisibilité.

Plusieurs attributs de la commande de *<table>* concourent à appliquer un quadrillage au tableau.

L'attribut **border="Valeur"** définit la largeur de la bordure et le quadrillage.

Les valeurs s'expriment en pixels : *border="3"*, *border="15"* ou *border="20"*.

Exemples :

```
<table border="0" align="center">
  <tbody>
    <tr>
      <td>Contenu</td>
      <td>Contenu</td>
      <td>Contenu</td>
    </tr>
    <tr>
      <td>Contenu</td>
      <td>Contenu</td>
      <td>Contenu</td>
    </tr>
    <tr>
      <td>Contenu</td>
```

```
      <td>Contenu</td>
      <td>Contenu</td>
    </tr>
  </tbody>
</table>
```

Contenu	Contenu	Contenu
Contenu	Contenu	Contenu
Contenu	Contenu	Contenu

```
<table border="10" align="center">
  <tbody>
    <tr>
      <td>Contenu</td>
      <td>Contenu</td>
      <td>Contenu</td>
    </tr>
    <tr>
      <td>Contenu</td>
      <td>Contenu</td>
      <td>Contenu</td>
    </tr>
    <tr>
      <td>Contenu</td>
      <td>Contenu</td>
      <td>Contenu</td>
    </tr>
  </tbody>
</table>
```

Contenu	Contenu	Contenu
Contenu	Contenu	Contenu
Contenu	Contenu	Contenu

2.6.8.6 QUADRILLAGE AVEC FRAME

Un tableau peut comporter un quadrillage spécial en affichant seulement une partie des bordures. Un attribut de la balise *<table>* se charge de cette fonction, mais elle ne peut être appliquée que si l'attribut *border* est lui-même activé.

L'attribut **frame="Type de bordure"*** définit le quadrillage du tableau.

Type de bordure	Description
above	bordures externes en haut du tableau seulement
below	bordures externes en bas du tableau seulement

border	bordures sur tous les côtés du tableau
box	bordures autour du tableau seulement
insides	bordures sur le dessus et le dessous du tableau seulement
hsides	bordures externes du coté horizontal du tableau
lhs	bordures externes du coté gauche du tableau seulement
rhs	bordures externes du coté droit du tableau seulement
void	enlève toutes les bordures externes du tableau
vsides	bordures externes du coté droit et gauche du tableau

Tableau 12 : types de bordure (attribut frame)

2.6.8.7 L'ATTRIBUT RULES

Un tableau peut comporter un quadrillage spécial en affichant seulement une partie des bordures.Un attribut de la balise *<table>* se charge de cette fonction, mais elle ne peut être appliquée que si l'attribut *border* et les commandes *<thead>*, *<tbody>* et *<tfoot>* sont eux-mêmes activés.

L'attribut **rules="Type de bordure"*** définit le quadrillage du tableau.

Type de bordure	Description
all	toutes les bordures dans le tableau
cols	bordure horizontale entre toutes les colonnes du tableau
groups	bordure horizontale entre les sections <thead>, <tbody> et <tfoot>
none	sans les bordures internes du tableau
rows	bordure horizontale entre toutes les lignes du tableau

Tableau 13 : types de bordure (attribut rules)

2.6.8.8 LES COULEURS DE BORDURE

Les balises *<table>*, *<td>*, et enfin *<th>* acceptent des attributs permettant de modifier la couleur des bordures.

L'attribut **bordercolor="couleur"** définit une couleur de bordure.

L'attribut **bordercolordark="couleur"** définit une couleur de bordure foncée.

L'attribut **bordercolorlight="couleur"** définit une couleur de bordure claire.

2.6.8.9 LES FONDS

Le langage HTML offre à l'utilisateur la possibilité d'appliquer à un tableau et à chacune des cellules des fonds de couleur ou des fonds à base d'images.

L'attribut **background="Adresse de l'image"** définit une image en arrière-plan de l'élément.

L'attribut **bgcolor="couleur"** définit une couleur de fond.

2.6.8.10 LES FUSIONS

Le langage HTML procure des fonctionnalités équivalentes à celles offerts par les tableurs ou traitement de texte en matière de fusion des cellules. Par exemple, le titre d'un tableau se place idéalement dans une seule cellule chapeautant les autres, ou pour des effets particuliers, il peut être utile de fusionner des cellules horizontalement ou verticalement selon les besoins.

Les attributs de fusions se rencontrent dans les balises de cellules.

L'attribut **colspan="Nombre de cellules"** définit le nombre de cellules à fusionner horizontalement.

L'attribut **rowspan="Nombre de cellules"** définit le nombre de cellules à fusionner verticalement.

Exemple :

```
<table border="1">
  <thead>
    <tr>
      <td colspan="5">
      <!– Cellules fusionnées horizontalement -->
      </td>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td rowspan="5">
      <!– Cellules fusionnées verticalement -->
      </td>
      <td colspan="4">
      <!– Cellules fusionnées horizontalement -->
      </td>
    </tr>
    <tr>
      <td>Cellule</td>
      <td>Cellule</td>
      <td colspan="2" rowspan="4">
      <!– Cellules fusionnées verticalement et horizontalement -->
      </td>
    </tr>
    <tr>
      <td>Cellule</td>
      <td>Cellule</td>
    </tr>
    <tr>
      <td>Cellule</td>
      <td>Cellule</td>
    </tr>
    <tr>
      <td>Cellule</td>
      <td>Cellule</td>
    </tr>
  </tbody>
</table>
```

<table>
<tr><td colspan="3">Cellules fusionnées dans l'en-tête horizontalement par <td colspan="5"></td></tr>
<tr><td rowspan="5">Cellules fusionnées dans le corps verticalement par
<td rowspan="5"></td></td><td colspan="2">Cellules fusionnées horizontalement dans le corps par <td colspan="4"></td></td></tr>
<tr><td>Cellule</td><td>Cellule</td><td rowspan="4">Cellules fusionnées verticalement et horizontalement dans le corps par
<td colspan="2" rowspan="4"></td></td></tr>
<tr><td>Cellule</td><td>Cellule</td></tr>
<tr><td>Cellule</td><td>Cellule</td></tr>
<tr><td>Cellule</td><td>Cellule</td></tr>
</table>

Figure 4 : illustrations de fusions entre cellules d'un tableau

2.7 GESTION DES FORMULAIRES

2.7.1 LA BALISE <FORM>

2.7.1.1 PRESENTATION

HTML donne la possibilité d'établir des formulaires grâce à certaines commandes.

Dans des formulaires, l'utilisateur peut compléter des champs de saisie, dans des champs texte entrer plusieurs lignes de texte, faire des choix dans des listes et cliquer sur des boutons.

Quand le formulaire est rempli complètement l'utilisateur peut cliquer sur un bouton pour envoyer le formulaire.

Avec **<form …>** on définit un formulaire (*form = Formulaire*). Tout ce qui se trouve entre ce repère d'ouverture et le repère de fermeture **</form>** fait partie du formulaire. Ce sont principalement des éléments du formulaire comme des champs de saisie, des listes de choix ou des boutons.

2.7.1.2 LES ATTRIBUTS

A ces attributs sont associés un certain nombre d'attributs explicités par la suite.

Propriétés	Descriptif	Valeurs
action	Définit le type d'action à déclencher	url
method	Définit la méthode de transfert des données	GET | POST
enctype	Définit le format de codage des données si le protocole du serveur n'a pas imposé le sien	type_MIME | multipart/form-data | application/x-www-form-urlencoded (par défaut)
target	Définit une fenêtre vers laquelle sont dirigés les résultats en provenance du serveur	nom_fenetre_cible
accept	Définit une liste de types MIME acceptés par le serveur traitant le formulaire.	Type_MIME,multipart/form-data, application/x-www-form-urlencoded ….
accept-charset	Définit une liste de jeux de caractères acceptés par le serveur	ISO-8859-1 (par défaut)

Tableau 14 : attributs de la balise form

2.7.1.3 LES CARACTERISTIQUES DE TRANSIMISSION

La balise *<form>* possède deux attributs spécifiant d'une part une adresse ou le formulaire devra être envoyé : **une adresse destinatrice** et d'autre part **la méthode de transmission** sur le réseau Internet.

L'attribut **method="Type de transmission"*** définit le type de transmission.

L'attribut **enctype="Type d'encodage"**** définit la méthode d'encodage des données.

L'attribut **action="Adresse cible"***** définit l'adresse destinatrice sur le site.

**** Il y a deux méthodes d'accès au serveur http :***

Soit la **method=GET** (par défaut, si la méthode n'est pas spécifiée), elle permet d'envoyer les éléments du formulaire au travers de l'URL, en ajoutant l'ensemble des paires nom/valeur à l'URL, séparé de celui-ci par un point d'interrogation.

Toutefois, la longueur de la chaîne URL étant limitée à 255 caractères, les informations situées au-delà de cette limite seront irrémédiablement perdues. De plus, cela crée une URL surchargée dans la barre d'adresse d'un navigateur (ceci n'est pas le cas au MAE du fait de l'utilisation de Frames) ou dans l'historique.

Elle peut ainsi dévoiler des informations sensibles.

Soit la **method=POST**, elle est une bonne alternative à la méthode GET. Cette méthode code les informations de la même façon que la méthode GET (encodage URL, paires nom/valeur) mais elle envoie les données à la suite des entêtes http, dans un champ appelé *corps de la requête*. De cette façon la quantité de données envoyées n'est plus limitée. Elle est connue du serveur grâce à l'entête permettant de connaître la taille du *corps de la requête*.

Méthode préconisée pour les formulaires :

POST :

lorsque la saisie transactionnelle pour une action (Ex : demande de connexion) nécessite une seule page de formulaire.

Lorsque la soumission du formulaire entraîne une URL en méthode GET dépassant les 255 caractères.

Lorsque la saisie contient des données sensibles (Ex : mot de passe).

GET :

lorsque la saisie transactionnelle pour une action (Ex passation d'un ordre) nécessite plusieurs pages de formulaire et que cette saisie ne contient aucune donnée sensible. Cette méthode permet ainsi de pouvoir gérer le BACK du navigateur pour revenir à la première page formulaire liée à l'action transactionnelle et éviter toutes revalidations ou resaisies à un état non stable de la transaction (voir §5.7.1 pour explication technique).

Lorsque les contraintes de limite de taille d'URL et de non saisie de données sensibles décrites ci-dessus sont respectées.

Point de vue sécuritaire lors d'un applicatif mis en œuvre pour Internet/Extranet :

Si des champs complémentaires non saisis par l'utilisateur sont nécessaires à la transaction et contiennent des données sensibles, il est nécessaire de crypter ces informations côté serveur en utilisant

des champs cachés pour garder des informations sur le contexte de cryptage. Le détail de cette utilisation sera explicitée dans un document ultérieur recensant l'ensemble des préconisations sécuritaires.

*** Les types d'encodage :***

Le type *application/x-www-forme-urlencoded* est inefficace pour envoyer de grandes quantités de données binaires ou des textes qui contiennent des caractères non-ASCII. C'est le type par défaut. Le type "*multipart/form-data*" doit être utilisé pour soumettre des formulaires qui contiennent des données non-ASCII ou/et des données binaires.

En général, l'attribut "*enctype*" spécifie le type MIME utilisé pour soumettre le formulaire au serveur. Lorsque la valeur de l'attribut "*method*" vaut "*post*", la valeur associée par défaut est "*application/x-www-form-urlencoded*". La valeur "*multipart/form-data*" est employée lorsque le formulaire inclut le téléchargement de fichiers associés.

Comme avec tous les types MIME multipart, chaque partie a un en-tête "*Content-type*" facultatif qui par défaut est "*text/plain*". les navigateurs doivent fournir l'en-tête du "*Content-type*", accompagné par un paramètre "*charset*". Bref, à savoir, la valeur "*text/plain*" est utlisé le plus couramment afin d'encoder les formulaires.

Si le contenu d'un fichier est soumis au serveur par l'intermédiaire d'un formulaire, l'entrée du fichier devrait être identifiée par le "*content-type*" approprié, par exemple, "*application/octet-stream*". Si les fichiers multiples sont retournés comme le résultat d'une entrée unique du formulaire, ils devraient être retournés comme "*mulltipart/mixed*" intégré à l'intérieur du "*multipart/form-data*".

Pour en savoir plus sur les types d'encodage, le site de l'organisation W3C vous offre de plus amples explications, mais le contenu de ce site est en langue anglaise.

L'utilisation de type MIME multipart doit faire l'objet d'une étude pour justifier son besoin. Des préconisations ciblées suivant le besoin exprimé et les contraintes de sécurité associées sont fournis pour toute demande de prise en compte.

*** L'action induite après validation du formulaire:***

Dans le repère d'ouverture **<form>**, on peut indiquer avec **action=** ce qui doit se passer avec le formulaire rempli quand l'utilisateur l'envoie. La mention doit être placée entre guillemets.

action peut aussi être défini via « javascript : », afin d'éviter que le navigateur ne recharge la page lorsque l'utilisateur tape « ENTER » et que le focus se trouve dans une zone de saisie.

Toutefois **action** n'est pas obligatoire dans les cas les plus courants d'utilisation de formulaire, c'est à dire avec la méthode GET pour passer des informations non sécurisées à la page suivante définie dans un lien.

Les protocoles autorisés pour cet attribut sont :

Protocole	Description
file:	Les fichiers locaux sur votre matériel informatique (c:, d:, etc.)
ftp:	Protocole de transfert de fichiers

gopher:	Protocole Gopher, un modèle client-serveur permettant de lire des menus distants sur une machine
http:	Protocole de transfert hypertexte
mailto:	Adresse électronique ou adresse eMail
news:	Protocole de réseau de transfert de nouvelles (forum de discussions)
nntp:	Nouvelles Usenet réseau des News du Web
telnet:	Protocole utilisant Telnet logiciel permettant de se connecter sur un serveur pour y exécuter des commandes
wais:	Protocole de serveur d'information à large zone en fait un système de base de données textuelles (Wide Area Information Server Protocol)

Tableau 15 : protocoles d'envoi d'un formulaire

2.7.2 LA SAISIE DE L'UTILISATEUR

2.7.2.1 LES CHAMPS D'ENTREE

La balise champ d'entrée permet, par l'intermédiaire de son attribut de définition d'un type, d'affecter à ce genre de champ un emploi particulier.

En effet, le champ de saisie peut se muer aussi bien en bouton, qu'en case à cocher ou encore en champ de mot-de-passe, etc.

La commande **<input>** définit un champ d'entrée.

L'attribut **name="Nom du champ d'entrée"** définit le une chaîne de caractères pour le champ d'entrée.

L'attribut **type="Type de champ"** définit le type de donnée du champ de saisie.

Type de champ	Description
button	Création d'un bouton poussoir
checkbox	Création de cases à cocher
file	Création d'un bouton de sélection de fichier
hidden	Création d'un champ cachée
image	Création d'un bouton image
password	Création d'un champ autorisant la saisie de mot de passe
radio	Création de cases radios
reset	Création d'un bouton de réinitialisation
submit	Création d'un bouton de soumission
texte	Création d'un élément de saisie de texte.

Tableau 16 : valeurs de l'attribut type de la commande input

Exemple d'implémentation :

```
<html>
  <head>
    <title>Un titre pertinent</title>
    <meta name="Description" content="...">
    <meta http-equiv="Date" content="01/01/2000">
    ...
  </head>
  <body>
    <form>
      <input name="Nom du champ"
             type="Type de champ">
      ...
    </form>
    ...
    Diverses commandes HTML
    ...
  </body>
</html>
```

Diverses commandes et attributs permettent d'apporter à balise précitée de nombreuses fonctionnalités améliorant l'ergonomie, la convivialité et la présentation.

L'attribut **disabled** permet la désactivation de l'élément.

L'attribut **readonly** spécifie le mode lecture seule de l'élément.

L'attribut **tabindex="Valeur"** définit un ordre de tabulation.

L'attribut **notab** exclut l'élément de l'ordre de tabulation.

L'attribut **accesskey="Raccourci"** définit un raccourci (lettre + Alt).

L'attribut **align="Type"** d'alignement" définit un alignement de l'élément.

L'attribut **lang="Langue"** définit une langue pour l'élément.

L'attribut **dir="rtl/ltr**" définit une direction d'écriture.

L'attribut **title="Description"** définit une description affichée dans une bulle.

L'attribut **value="Chaîne de caractères**" définit la valeur du paramètre envoyé à l'application traîtant le formulaire si l'élément est sélectionné.

Le nom du champ de saisie ne doit pas comporter des caractères spéciaux : **name="NomAuteur"**.

2.7.2.2 LES CHAMPS DE SAISIE

En général, dans un formulaire se trouvent des champs de saisie.

Les champs de saisie servent dans la plupart des cas à saisir des valeurs (âge, nombre quelconque, etc.) ou des mots (nom, adresse, etc.) ainsi que des mots-de-passe.

La commande **<input type="text">** crée un champ de saisie d'une seule ligne.

La commande **<input type="password">** crée un champ de saisie mot-de-passe.

En général, dans un formulaire se trouvent des champs de saisie.

Les champs de saisie servent dans la plupart des cas à saisir des valeurs (âge, nombre quelconque, etc.) ou des mots (nom, adresse, etc.).

L'attribut **size="Valeur"*** définit la longueur de champ de saisie.

L'attribut **maxlength="Valeur"**** définit le nombre total de caractères dans le champ.

* Nombre de caractères visibles dans le champ de saisie, pour un champ de 12 caractères : *size="12"*.

** Nombre maximum de caractères dans le champ de saisie, pour un champ de 64 caractères : *maxlength=64*.

2.7.2.3 LES CASES RADIOS

Les cases radios permettent à l'utilisateur de faire un seul choix parmi plusieurs options possibles.

La commande **<input type="radio">** crée une case radio.

L'attribut **checked** sélectionne une case par défaut.

2.7.2.4 LES CASES A COCHER

Les cases à cocher font parties en général d'un groupe d'options dans lequel un usager peut sélectionner un ou plusieurs items contrairement aux cases radios.

La commande **<input type="checkbox">** marque la fin du formulaire.

L'attribut **checked** sélectionne une case par défaut.

2.7.2.5 LES BOUTONS PREDEFINIS

Les boutons de commande permettent après le renseignement des différents éléments d'un formulaire par un utilisateur d'envoyer vers le destinataire les informations ou encore d'annuler le processus.

La commande **<input type="submit">** crée un bouton d'enregistrement.

La commande **<input type="reset">** crée un bouton d'annulation.

La commande **<input type="file">** crée un bouton fichier attaché.

L'attribut **accept="Nom du bouton"** spécifie le type de fichier attaché.

2.7.2.6 LES BOUTONS

Les boutons de commande permettent après le renseignement des différents éléments d'un formulaire par un utilisateur d'envoyer vers son destinataire les informations ou encore d'annuler le processus.

La commande **<button>** crée un bouton poussoir.

La commande **</button>** ferme la balise précédente.

L'attribut **type="type de bouton"** permet de définir la fonction du bouton.

L'attribut **value="Nom de la commande"** permet d'affecter une valeur au bouton.

L'attribut **name="Nom du bouton"** permet de nommer la commande.

Type	Description	Exemple

button	Création d'un simple bouton permettant de déclencher un script.	<button type="button" value="button" name="script"> Lancement script </button>
submit	Création d'un bouton de provoquer la soumission du formulaire.	<button type="submit" value="submit" name="soumission"> Enregistrer </button>
reset	Création d'un bouton de réinitialisation du formulaire.	<button type="reset" value="reset" name="rechargement"> Recommencer </button>

Tableau 17 : types de bouton

Les attributs possibles de la balise *button*.

L'attribut **disabled** permet la désactivation de la zone de texte.

L'attribut **tabindex="Valeur"** définit un ordre de tabulation.

L'attribut **usemap="Adresse de l'image"** définit une image en coordonnées.

L'attribut **accesskey="Raccourci"** permet d'affecter un raccourci clavier (ALT + Lettre) à l'élément.

L'attribut **title="Description"** définit une description affichée dans une bulle.

2.7.2.7 LES BOUTONS IMAGES

Les images peuvent se substituer aux boutons de commandes standards à l'instar des hyperimages et partant d'accomplir des fonctions identiques aux boutons précités.

La commande **<input type="image">** spécifie la création d'un bouton à base d'une image.

L'attribut **src= *"Adresse de l'image"*** permet d'affecter une image à la commande.

L'attribut **usemap= *"Adresse de l'image"*** permet d'affecter une image en coordonnées (image-map) à la commande.

L'attribut **alt="Texte"** affiche une description de l'image.

2.7.2.8 PRECONISATION SUR L'UTILISATION DES BOUTONS

On peut remarquer que lors de l'implémentation d'un bouton dans un formulaire, plusieurs codages sont donc possibles.

Afin d'être en adéquation avec la charte graphique, il est préconisé d'enrichir les boutons de formulaire (validation, reset, actions…) à l'aide d'image. On serait donc amené à préférencier l'implémentation de bouton à l'aide des balises **<input type="image">**.

Hélas, pour ce type de bouton, le problème rencontré est que des champs X=x_coordonnée&Y=y_coordonnée sont ajoutés à l'url définissant ainsi les coordonnées du pointeur de la souris au moment où l'utilisateur a cliqué sur l'image et dans le cas où un nom est associé à la balise alors les champs le_nom.X=x_coordonnée&le_nom.Y=y_coordonnée sont ajoutés à la suite de la requête soumise par validation du formulaire.

L'ajout non contrôlé de ces champs est indésirable surtout dans un cas de filtrage des requêtes entrantes pour des raisons de sécurités.

De ce fait, il est préconisé pour remédier à ce problème d'utiliser un lien qui appelle une fonction javascript sous une image choisie. C'est cette fonction qui soumet la requête au serveur soit en construisant l'url soumise en méthode get du formulaire soit en faisant appel au « submit » du formulaire en méthode post (voir §5.7.1 pour détail de l'implémentation de cette fonction.

Exemple d'implémentation :

```
<a href="javascript:valider('url')"><img src="valider.gif" width="91" height="20"
border="0"></a>
```

2.7.2.9 LES ENTREES CACHEES

Les entrées cachées permettent, lors de l'envoi, de joindre au formulaire une information non visible pour l'utilisateur. Lorsque la balise *<input>* possède le type *hidden*, elle n'accepte plus que deux attributs, en l'occurrence *name* et *value*.

La commande **<input type="hidden">** spécifie la création d'une entrée cachée.

L'attribut **name="Nom du champ"** permet d'affecter un nom à l'entrée cachée.

L'attribut **value="Valeur de la variable"** permet d'affecter une valeur à l'entrée cachée.

2.7.2.10 LES CHAMPS MULTILIGNES

Un champ multiligne permet d'entrer un texte de plusieurs lignes dans une zone de saisie.

La commande **<textarea>** définit une zone de texte.

La commande **</textarea>** ferme la zone de texte.

L'attribut **name="Nom du champ"** définit un nom pour la zone de texte.

L'attribut **cols="Valeur"** spécifie un nombre de caractères sur une ligne.

L'attribut **rows="Valeur"** spécifie un nombre de lignes.

Un attribut de la zone de texte permet de spécifier le mode de coupure de ligne dans le champ et plusieurs autres apportent des fonctionnalités diverses à la commande.

L'attribut **wrap="off/virtual/physical"*** définit la présence de coupure de ligne(cf. tableau ci-dessous).

L'attribut **disabled** permet la désactivation de la zone de texte.

L'attribut **readonly** spécifie le mode lecture seule de l'élément.

L'attribut **tabindex="Valeur"** définit un ordre de tabulation.

L'attribut **align="Type d'alignement"** définit un alignement de l'élément.

L'attribut **lang="Langue"** définit une langue pour l'élément.

L'attribut **dir="rtl/ltr"** définit une direction d'écriture.

L'attribut **title="Description"** définit une description affichée dans une bulle.

* Choix	Description
off	La coupure de ligne désactivée.
physical	La coupure des lignes est activée.
virtual	La coupure des lignes est activée pour l'usager mais transmise sans coupure des lignes.

Tableau 18 : valeurs de l'attribut wrap de la commande textarea

2.7.2.11 LES LISTES DE CHOIX

Les listes de choix sont des champs affichant une série d'items sous forme soit de liste déroulante, soit de liste simple.

De plus, les listes de choix peuvent autoriser la multisélection d'items.

La commande **<select>** définit une liste de choix.

La commande **</select>** ferme la liste de choix.

L'attribut **name="Nom du champ"** spécifie un nom pour l'élément.

L'attribut **size="Valeur"*** définit le type de liste et le nombre d'items.

L'attribut **multiple** permet de créer une liste à choix multiple.

L'attribut **tabindex="Nombre"** définit la position de la liste dans l'ordre de tabulation.

L'attribut **disabled** permet de rendre la liste inactive.

La commande **<option>** déclare une entrée dans la liste.

La commande **</option>** ferme l'entrée précitée.

L'attribut **value="Nom de l'entrée"** définit la sélection d'un item par défaut.

L'attribut **selected** définit la sélection d'un item par défaut.

L'attribut **label="Chaîne de caractères"** définit un label pour l'option.

*Valeur	Description
1	Une valeur définie à 1 provoque une liste déroulante.
>1	Une valeur supérieure à 1 provoque une liste simple.

Tableau 19 : valeurs de l'attribut size de la commande select

2.7.2.12 LES REGROUPEMENTS

Le langage HTML permet de **regrouper différents éléments d'un formulaire** dans un cadre représentant un thème permettant par là d'améliorer la lisibilité de la page.

La commande **<fieldset>** marque le début d'un regroupement.

La commande **</fieldset>** marque la fin du regroupement.

La commande **<legend>** attribue une légende au regroupement.

La commande **</legend>** permet de fermer la légende.

L'attribut **align="Type d'alignement"*** définit un alignement pour la légende.

L'attribut **accesskey="Raccourci"** définit un raccourci (lettre + ***Alt***).

* Les options possibles d'alignement sont "**Left**" (par défaut) pour la gauche, "**right**" pour la droite, "**bottom**" pour le bas ou "**top**" pour le haut.

2.7.2.13 LES LABELS

La balise *<label>* permet d'affecter une étiquette associée à un élément d'un formulaire

La commande **<label>** insère une étiquette à proximité d'un élément du formulaire.

La commande **</label>** marque la fin de l'étiquette.

L'attribut **for="Identificateur"** spécifie l'identificateur de l'élément devant accueillir le label.

L'attribut **accesskey="Raccourci"** définit un raccourci (lettre + *Alt*).

L'attribut **disabled** permet de rendre la liste inactive.

L'attribut **tabindex="Nombre"** définit la position de la liste dans l'ordre de tabulation.

2.8 GESTION DU DYNAMISME DES PAGES

L'utilisation conjointe des feuilles de style, du code JavaScript, du Document Object Model (DOM) avec HTML permet de gérer du dynamisme dans les pages. Ainsi, le rendu de la page différera suivant les événements utilisateur ou propres à la page.

JavaScript, DOM et CSS sont abordés dans la suite du document.

2.9 ICONES DE PAGES WEB

Avec Mozilla et Internet Explorer, il est possible d'avoir une petite icône au niveau des signets ou dans la barre d'adresse. Cela permet de faire ressortir un site dans la liste et c'est la favicon.ico.

2.9.1 LE FORMAT DES ICONES

Un seul format est supporté par tous, c'est le **ICO.**

Internet Explorer n'accèpte que ce type de format (ICO) alors que Mozilla lui est compatible avec les types suivants : JPEG, GIF, PNG, MNG, XBM, BMP et ICO.

La taille des icônes doit être de format **16x16** ou **32x32**.

2.9.2 LE CODE HTML DANS LA PAGE

Si on respecte les standards on doit mettre le lien suivant dans la page entre les balises **<head>** et **</head>** :

```
<link rel="icon" type="image/png" href="/favicon.png" />
```

Voici les types utilisables pour les 4 classiques :

- **image/png**: image PNG.
- **image/gif**: image GIF.
- **image/jpeg**: image JPEG.
- **image/x-icon**: image de type icon, attention ce format n'est pas un format standard.

Si on ne respecte pas les standards d'image pour une compatibilité vis à vis d'Internet Explorer, il est préférable d'implémenter la ligne ci-dessous :

```
<link rel="shortcut icon" type="image/x-icon" href="/favicon.ico" />
```

A noter : l'utilisation du type **shortcut icon** à la place de **icon** et l'obligation d'utiliser une image au format ICO.

Il est à noter qu'il est tout à fait posible d'avoir à l'instar des gifs animés des favicon animées.

Voici une explication à travers un exemple :

```
<link rel="icon" href="images/favicon.mng" type="video/x-mng" />
```

Le type MNG c'est comme les gifs animés pour le PNG. Pour faire supporter le tout aux deux navigateurs cibles, il faut instancier le code dans la page de la manière suivante :

```
<head>
...
  <link rel="shortcut icon" type="image/x-icon" href="/favicon.ico" />
  <link rel="icon" type="video/x-mng" href="/favicon.mng" />
</head>
```

En mettant après le lien pour Internet Explorer, le lien vers la version animée, Internet Explorer va prendre la première solution qu'il est capable d'interpréter et ne s'occupe pas du reste. Mozilla, lui interpréte tout, mais ne va garder que la dernière déclaration, c'est à dire la version animée pour l'affichage.

On peut bien entendu utiliser directement un gif animé pour l'icône, mais c'est un format, même si utilisé partout, propriétaire. Comme les navigateurs qui supportent les gifs animés pour l'icône supportent aussi les MNG, autant suivre les standards. Le fichier MNG permet par ailleurs d'avoir plus de 256 couleurs dans l'icône.

3 NORME DE DEVELOPPEMENT CSS

3.1 INTRODUCTION

Les feuilles de style CSS (Cascading Style Sheets) permettent de décrire un modèle de présentation dans le but, toujours omniprésent dans les développements actuels, de bien séparer les données de leur présentation.

Dans le cas d'un document HTML, le contenu est de l'information brute (sans aucune notion de présentation) mais structurée, et la présentation est un ensemble de règles définissant précisément la façon dont chaque élément doit être présenté.

L'homogénéité des pages d'un site Web est ainsi plus facile à obtenir et la gestion des modifications s'en trouve simplifiée.

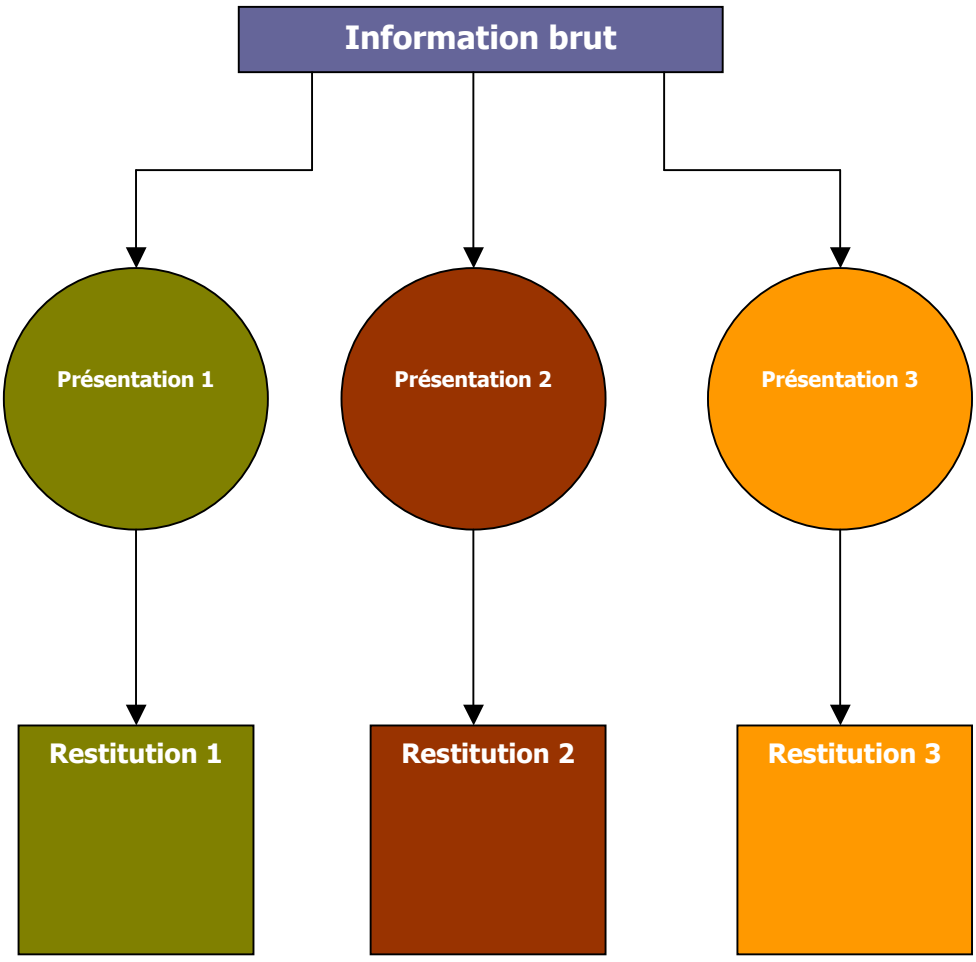


Figure 5 : concept de la feuille de style

Dans les différents chapitres concernant la norme de réalisation CSS, une liste de propriété a été présentée pour différents modules (background, font, text, border, margin, padding…). Il s'agit des propriétés les plus courantes.

Une liste plus exhaustive est présentée en Annexe B : Références CSS répertoriant les différentes propriétés par type, avec les différentes valeurs associées à ces propriétés. Cette liste spécifie la compatibilité de l'applicatif vis à vis des styles et la version CSS associée à chaque style.

Il existe deux spécifications :

- CSS1 (level 1).
- CSS2 (level 2).

Ces spécifications ont été validées par le W3C.

Il existe une troisième spécification : le CSS3 (level 3).

3.2 SYNTAXE

3.2.1 DEFINITION

Normalement, la syntaxe des déclarations CSS est toujours identique. La méthode est simple : il s'agit de redéfinir le rôle d'une balise html en lui imposant de nouvelles propriétés.

Il existe quatre méthodes différentes permettant d'intégrer les feuilles de styles en html :

- L'instruction style en tant qu'élément,
- L'élément « link » pour établir un lien vers une feuille de style externe (fichier texte avec l'extension .css),
- L'instruction style en tant qu'attribut,
- Importer une feuille de style (fichier texte avec l'extension .css).

A noter, que sur nos applicatifs, nous utilisons la deuxième méthode. En effet, il est préférable d'importer un fichier externe avec l'extension css par lot ergonomique. Ces lots ergonomiques sont définis soit par le découpage fonctionnel, soit par une personnalisation par profil utilisateur ou soit par spécificités ergonomiques (menu, aide, popup…). Cette externalisation permet un découpage plus fin et une facilité de mise à jour lors des problèmatiques de maintenance.

De plus, ces fichiers .css ne doivent pas contenir de code html.

3.2.2SYNTAXE GENERALE

Le langage CSS utilise sa propre « syntaxe » et « vocabulaire » pour effectuer son distinguo avec les autres langages :

- le mot « **déclaration** » indique l'ensemble. Ex : **`h2 {font-size : 16pt;}`**
- le mot « **sélecteur** » en lieu et place de « tag » (balise) du code pour HTML. Ici : **`h2`**
- le mot « **bloc** » spécifie un ensemble comme la ligne propre au sélecteur h2. Ici : **`{font-size : 16pt;}`**
- la « **propriété** » est ici **`font-size`**.
- La « **valeur** » est ici **`16pt;`**
- **`{……}`** les **accolades**.
- **" : " deux points** pour séparer la propriété de sa valeur.

- **" ; " point-virgule** placé à la fin des blocs lorsqu'une déclaration en contient plusieurs.
- **" . " point** devant chaîne de caractères pour définir des classes.
- **<……> chevrons** toujours par paire.

3.2.3 L'INSTRUCTION STYLE EN TANT QU'ELEMENT (EMBEDDING)

La syntaxe est incorporée à **chaque page** dans un bloc <STYLE> --//-- </STYLE> placé entre les tags HTML <HEAD> de la page. Cette insertion s'applique à la seule page ainsi marquée.

Comme pour les navigateurs ignorant le javascript, il est nécessaire de cacher ce code des feuilles de styles à l'aide des commentaires :

```
<!-- //-->
```

3.2.4 L'INSTRUCTION STYLE EN TANT QU'ATTRIBUT (ADDING IN LINE)

Une définition peut se faire **individuellement** pour un tag HTML donné.

Par exemple pour un paragraphe de la manière suivante :

```
<p style="font-size: 12pt; color: green">
```

3.2.5 METHODE EXTERNE (LINKING)

Dans ce cas, l'ensemble des déclarations est effectué dans un **fichier séparé** dont l'extension est **.css**.

Ce fichier précise les styles utilisés pour un groupe de pages. Chacune des pages utilisatrices place dans son en-tête l'appel à ce fichier de la manière suivante :

```
<link title="test" type="text/css" rel="stylesheet" href="mafeuille.css">
```

C'est cette méthode d'insertion de styles qui est privilégié dans le cadre de ce projet par soucis de simplicité et de lotissement.

3.2.6 METHODE EXTERNE PAR IMPORT

Tout comme la méthode externe par lien, tout est dans un fichier séparé qui portera l'extension **.css.**

```
<style type="text/css">
@import url(http://www.domaine.com/mafeuille.css);
</style>
```

3.3 SELECTEURS

Un sélecteur est une série de conditions sur la structure d'un document HTML. Si les conditions exprimées par le sélecteur sont vraies, alors le style défini dans la déclaration sera appliqué à l'élément.

On distingue plusieurs catégories de sélecteurs :

- Les sélecteurs simples.
- Les combinateurs.
- Les pseudo-éléments.

3.3.1 LES SELECTEURS SIMPLES

Les sélecteurs simples permettent d'exprimer une condition unitaire sur un élément.

Les différents sélecteurs sont :

- **Les sélecteurs de type d'élément :** composés uniquement d'un nom d'élément ou en d'autres termes, d'une balise HTML (par exemple H1 ou P).
- **Les sélecteurs universels :** permettent de sélectionner indifféremment tous les types d'élément. Sa notation est *. La règle ainsi définie * {…} est appliquée à tous les éléments du document.
- **Les sélecteurs d'attribut :** ils permettent d'exprimer une condition sur l'existence d'un attribut porté par un élément ou sur la valeur de cet attribut. Par exemple, la règle *[LANG]{…} s'applique à tous les éléments portant l'attribut LANG, et la règle *[LANG="fr"]{…} s'applique à tous les éléments dont l'attribut LANG vaut "fr".
- **Les sélecteurs de classe :** Un sélecteur de classe est représenté par un point « . » Suivi du nom de classe à sélectionner. Il permet de sélectionner tous les éléments portant cette classe. Exemple : la règle **.bleu** { color : blue } serait appliquée à l'élément suivant : <p **class="bleu**"> paragraphe .
- **Les sélecteurs d'ID :** L'ID est un attribut spécial défini pour tous les éléments d'un document HTML. Le sélecteur d'ID permet donc de sélectionner tous les éléments portant cet attribut. Il est représenté par un « # » suivi d'un nom d'ID. Exemple : là règle #**nom** {…} s'applique à l'élément suivant : ceci est un nom .
- **Les pseudo-classes :** ce sont des sélecteurs simples spéciaux qui définissent des conditions difficilement exprimables par des sélecteurs standards. Elles sont représentées par « : » suivi du nom de pseudo-classe. Il existe 6 pseudo-classes, que nous ne détaillons pas toutes ici, mais parmi elles seulement les pseudo-classes liées aux hyperliens sont souvent utilisées. La pseudo-classe a:link permet de sélectionner les éléments de type hyperlien dont l'URL associée n'a pas encore été visitée, alors que la pseudo-classe a:visited sélectionne les liens déjà visités. Exemple : a:link(color:blue} et a:visited{color:grey}. Dans cet exemple les liens non visités apparaîssent en bleu alors que les liens déjà visités sont affichés en gris.

Pour faciliter les developpements dans le cadre de ce projet, il est recommandé de préférencier l'utilisation des sélecteurs de type d'élément et de classe. L'utilisation des sélecteurs universels, d'attribut et d'ID doivent faire l'objet d'une étude particulière pour justifier leur utilisation. L'utilisation des pseudo-classes doit se limiter à son application sur les éléments de type hyperlien.

Cette complexité est justifiée lors de l'utilisation du XHTML support du CSS comme explicitée dans le §4.3 et est préférable d'être simplifiée dans le cadre du framework ergonomique pour en simplifier son implémentation et mutualiser les efforts effectués dans sa mise en œuvre.

3.3.2 LES COMBINATEURS

Sans entrer dans les détails, signalons seulement qu'il existe des possibilités de combiner plusieurs conditions à l'aide de 3 combinateurs :

- l'espace pour exprimer la descendance.
- Le > pour exprimer une relation filiale.
- Et le + pour les relations d'adjacence.

Donnons quelques exemples :

- La règle div.nom strong{…} : sera appliquée à tous les éléments strong qui descendent d'un div portant la classe nom.
- La règle p>em{…} s'appliquera à tous les éléments em fils directs d'un élément p.

- La règle h1+p{…} s'appliquera au premier élément p qui suivra un élément h1.

De même que pour l'utilisation des sélecteurs simples, le développement de combinateurs dans les feuilles de style doit faire l'objet d'une étude explicitant la nécessité de cette complexité d'utilisation.

3.3.3 LES PSEUDO-ELEMENTS

A l'instar des pseudo-classes, les pseudo-éléments permettent d'exprimer des conditions qu'il serait difficile d'exprimer par des sélecteurs simples. Dans CCS2 ils sont au nombre de 4 et permettent de poser des conditions sur des fragments de textes, dont :

- :first-line : pour accéder à la première ligne d'un élément.
- :first-letter : pour accéder au premier caractère d'un élément.

Dans le cadre de ce projet, l'implémentation de la complexité des pseudo-éléments n'est pas justifié.

3.4 SPAN ET DIV

3.4.1 DEFINITION

Les balises div et span permettent de désactiver certaines instructions de la feuille de styles. Vous pouvez de cette façon modifier le style appliqué à certaines balises, tout en conservant les autres effets définis dans le fichier CSS.

A noter que :

- div est normalement utilisé pour des blocs de texte entier.
- span est lui utilisé que pour des petites portions de texte (mots ou lettres).

En sachant que span et div prennent en charge les attributs id et class.

3.4.2 EXEMPLE SPAN

Cette page utilise la feuille de style contenu.css.

```
<html>
<head>
<title>Exemple Span</title>
<link rel="stylesheet" href="/css/contenu.css" type="text/css">
….
<td width="124" class="classeEcriture1"><div align="right">+95 270,49 <span
class='monnaie'>EUR</span></div></td>
</head>
</html>
```

3.4.3 EXEMPLE DIV

Cette page utilise la feuille de style contenu.css.

```
<html>
<head>
<title>Exemple Div</title>
<link rel="stylesheet" href="/css/contenu.css" type="text/css">
```

```
….
<td width="124" class="classeEcriture1"><div class="classeEcriture2"
align="center">En-tête colonne</div></td>
</head>
</html>
```

3.5 ARRIERE-PLAN

3.5.1 DEFINITION

Les développeurs peuvent spécifier l'arrière-plan d'un élément comme étant une couleur ou bien une image.

Bien que les propriétés d'arrière-plan ne s'héritent pas, l'arrière-plan de la boîte du parent transparaît par défaut, du fait de la valeur initiale 'transparent' de la propriété background-color.

Dans le cas de documents HTML, les développeurs doivent spécifier un arrière-plan à l'élément BODY, plutôt qu'à l'élément HTML. Ils doivent ainsi suivre les règles de préséance suivantes pour remplir le fond du canevas : quand la valeur de la propriété background-color pour l'élément HTML diffère de 'transparent', alors utiliser la valeur spécifiée, autrement utiliser celle spécifiée par la propriété background de l'élément BODY. Le rendu n'est pas défini si la valeur finale reste 'transparent'.

3.5.2 PROPRIETES

3.5.3PROPRIETES

Propriétés	Descriptif	Valeurs	Exemple
background-color	Permet de définir la couleur de l'arrière plan	- **color-rgb** (Valeur rgb) - **color-hex** (Valeur Héxadécimal ex: #FFFFFF) - **color-name** (Nom de la couleur en anglais) - **transparent**	**h1{background-color:#ccffff}**
background-image	Permet de définir une image en arrière plan (gif ou jpg)	- **url** **- none**	**Texte**
background-repeat	Permet de définir le mode de répétition de l'image d'arrière-plan	- **repeat** : En mosaîque, valeur par défaut - **repeat-x** : Répétition horizontale - **repeat-y** : Répétition Verticale - **no- repeat** : Pas de répétition	**Texte**

background-attachment	Permet de bloquer le défilement de l'arrière plan	- **top left** **- top center** **- top right** **- center left** **- center center** **- center right** **- bottom left** **- bottom center** **- bottom right** **- x-% y-%** **- x-pos y-pos**	**Texte**
background-position	Permet de définir la position de l'image d'arrière plan	- **top left** **- top center** **- top right** **- center left** **- center center** **- center right** **- bottom left** **- bottom center** **- bottom right** **- x-% y-%** **- x-pos y-pos**	**Texte**
background	Permet de regrouper plusieurs informations sur l'arrière plan	**- background-color** **- background-image** **- background-repeat** **- background-attachment** **- background-position**	**Texte**

Tableau 20 : propriétés CSS concernant l'arrière-plan

3.6 TEXTE

3.6.1 DEFINITION

Les propriétés définies ci-dessous influencent la représentation visuelle des caractères, des blancs, des mots et des paragraphes.

3.6.2 PROPRIETES

Propriétés	Descriptif	Valeurs	Exemple
text-align	Permet d'aligner les éléments	- **left** : Alignement à gauche - **right** : Alignement à droite - **center** : Centrer - **justify** : Justifier	`<style>img{text-align:right}</style>`
text-indent	Permet de définir le retrait de la première ligne d'un texte	- **length** - **%**	`<span style="text-indent:20px;">Texte</span>`
text-decoration	Permet de définir la décoration des caractères	- **none** : Aucun effet (par défaut) - **underline** : Souligné - **overline** : Ligne au-dessus - **line-through** : Barré - **blink** : Clignotant	`<style>a:hover { color:#ff9900;text-decoration:underlineoverline; }</style>`
text-transform	Permet de définir la présentation d'un caractère	- **none** : Texte normal - **capitalize** : Majuscule en début de chaque mot - **uppercase** : Mots tout en majuscule - **lowercase** : Mots tout en minuscule	`<span style="text-transform:lowercase;">Texte</span>`
text-shadow	Permet de définir l'ombre d'un texte	**- none** - **color** - **length**	`<span style="text-shadow:#FF0099;">Texte</span>`

Tableau 21 : propriétés CSS pour le texte

3.7 POLICES

3.7.1 DEFINITION

Une *police* se compose d'un jeu de glyphes, ceux-ci étant fondés sur un même squelette pour leur dessin, leur taille, leur aspect et d'autres attributs associés au jeu entier, et d'un système de correspondance entre les caractères et ces glyphes abstraits.

En CSS2, les évolutions survenues accordent une plus grande liberté aux :

- développeurs de feuilles de style, pour décrire les polices que ceux-ci souhaitent voir utiliser;
- utilisateurs, pour sélectionner une police quand celle demandée n'est pas immédiatement disponible.

Cette évolution est gérée par la propriété « font » mais il est préférable de proscrire son utilisation du fait d'une différence de comportement et de support des différents postes client (configuration des polices, polices liées à version d'OS…). Ainsi, la limitation à l'utilisation des autres propriétés « font- » est préférenciée pour contrôler la qualité du rendu de l'affichage proposé à l'utilisateur.

3.7.2 PROPRIETES

Propriétés	Descriptif	Valeurs	Exemple
font	Permet de définir les propriétés de polices de caractères d'un élément	- **font-style** - **font-variant** - **font-weight** - **font-size/line-height** - **font-family** - **caption** - **icon** - **menu** - **message-box** - **small-caption** - **status-bar**	`<span style="font:oblique 10pt Arial">Texte</span>`
font-size	Permet de définir la taille des caractères d'un élément	- **xx-small** : Minuscule - **x-small** : Très petit - **small** : Petit - **medium** : Moyen - **large** : Grand - **x-large** : Très grand - **xx-large** : Immense - **smaller** : Plus petit que sa taille courante - **larger** : Plus grand que sa taille courante - ***valeur*** *: En unite de longueur (em, ex, px, in, cm, mm, pt, pc) ou pourcentage*	`<span style="font-size:16pt">Texte</span>`
font-family	Permet de définir la famille de la police de caractères d'un élément	- **family-name** - **generic-family**	`<span style="font-family:cursive">Texte</span>`

font-weight	Permet de définir la graisse de la police	- **normal** : Graisse normale : 400 (par défaut)
		- **bold** : Gras : 700
		- **bolder** : Plus gras
		- **lighter** : Plus maigre
		- **100** : Très fin
		- **200**
		- **300**
		- **400**
		- **500**
		- **600**
		- **700**
		- **800**
		- **900** : Très épais
			`<style>`
			`h3 { font-weight:900>`
			`</style>`

font-style	Permet de définir le style de la police d'un élément	- **normal** : Ecriture normale (défaut) - **italic** : Ecriture italique ou cursive - **oblique** : Ecriture penchée	`<span style="font-style:oblique">Texte</span>`
font-variant	Permet de générer des petites majuscules pour un élément	- **normal** : Ecriture normale - **small-caps** : Petite lettre capitale	`<style>h3 { font-variant:small-caps></style>`

Tableau 22 : propriétés CSS pour les polices

3.8 BORDURES

3.8.1 DEFINITION

Les propriétés de bordures définissent une bordure (style, couleur et taille) autour d'un élément.

3.8.2 PROPRIETES

Propriétés	Descriptif	Valeurs	Exemple
border	Permet de définir une bordure autour d'un objet	- **border-width** - **border-style** - **border-color** - **valeur**	`<span style="border:5;">Texte</span>`
border-x (x à	Permet de définir la bordure à droite.	- **valeur**	`<span style="border-bottom:double outset;">Texte</span>`

remplacer par right, left, top ou bottom)	bordure à droite, gauche, en haut ou en bas d'un élément		
border-width / border-x-width (x à remplacer par right, left, top ou bottom)	Permet de définir l'épaisseur d'une bordure d'un élément. L'épaisseur de la bordure peut se faire aussi sur une partie de l'élément : à droite, à gauche, en haut ou en bas d'un élément	- **valeur**	`<span style="border-bottom-width:7 px;">Texte</span>` `<span style="border-width:4 px;">Texte</span>`
border-style / border-x-style (x à remplacer par right, left, top ou bottom)	Permet de définir le style d'une bordure d'un élément. Le style de la bordure peut se faire aussi sur une partie d'un élément : à droite, à gauche, en haut ou en bas	- **none** : Pas de bordure - **hidden** - **dotted** : Pointillé - **dashed** : Discontinu - **solid** : Continu - **double** : Double - **groove** : Effet 3D - **ridge** : Effet 3D - **inset** : Effet 3D - **outset** : Effet 3D	`<span style="border-style:groove; border-color:#ccffff">Texte</span>` `<span style="border-top-style:groove; border-top-color:#ccffff">Texte</span>`
border-color / border-x-color (x à remplacer par right, left, top ou bottom)	Permet de définir la couleur d'une bordure d'un élément. La couleur de la bordure peut se faire aussi sur une partie d'un élément : à droite, à gauche, en haut ou en bas.	- **valeur color**	`<span style="border-color:#ccffff;">Texte</span>` `<span style="border-right-color:#ccffff;">Texte</span>`

Tableau 23 : propriétés CSS pour les bordures

3.9 MARGES

3.9.1 DEFINITION

Les propriétés de marges définissent les espacements autour des éléments.

3.9.2 PROPRIETES

Propriétés	Descriptif	Valeurs	Exemple
margin	Permet de définir les marges d'une page	**- margin-top**	`BODY {margin:2em} /*les quatres marges reçoivent la valeur 2em*/`

	marges d'une page	**- margin-right** **- margin-bottom** **- margin-left**	**BODY {margin:1em 2em} /*les marges du haut et du bas = 1em, de droite et de gauche = 2em*/** **BODY {margin:1em 2em 3em} /*haut=1em, droite=2em & gauche=2em, bas 3em*/**
margin-x (x à remplacer par right, left, top ou bottom)	Permet de définir la marge à droite, à gauche, en haut ou en bas d'une page	**- auto** **- length** **- %**	**Texte**

Tableau 24 : propriétés CSS pour les marges

3.10 ESPACEMENT

3.10.1 DEFINITION

Ces propriétés spécifient la largeur de l'aire d'espacement d'une boîte.

À la différence des propriétés de marge, les valeurs d'espacement ne peuvent pas être négatives. Les valeurs de pourcentage des propriétés d'espacement, tout comme celles des propriétés de marge, se réfèrent à la largeur du bloc conteneur de la boîte générée.

3.10.2 PROPRIETES

Propriétés	Descriptif	Valeurs	Exemple
padding	Permet de définir la marge entre le contenu et les délimitations haut, bas, gauche et droite de l'élément.	**- padding-bottom** **- padding-top** **- padding-right** **- padding-left**	**<style>** **div{ border: 6px #ccccff; padding:15%; }** **</style>**
padding-x (x à remplacer par right, left, top ou bottom)	Permet de définir la marge entre le contenu et les délimitations haut, bas, gauche et droite de l'élément	**- length** **- %**	**<style>** **td { padding-bottom:5px; }** **</style>**

Tableau 25 : propriétés CSS concernant l'espacement

3.11 LISTES

3.11.1 DEFINITION

Les *propriétés de liste* permettent une mise en forme limitée des listes. Tout comme les marqueurs, aux champs d'action plus étendus, un élément dont la propriété 'display' a la valeur 'list-item' génère une boîte principale pour son contenu ainsi qu'une boîte de marqueur facultative. Les propriétés de liste permettent de préciser le type

(image, glyphe ou nombre) et la position d'une boîte de marqueur par rapport à la boîte principale (à l'extérieur, ou à l'intérieur avant le contenu). Par contre, celles-ci ne permettent pas de spécifier un style distinct (de couleur, de police, d'alignement, etc.) pour le marqueur de liste ou la position de celui-ci par rapport à la boîte principale.

3.11.2 PROPRIETES

Propriétés	Descriptif	Valeurs	Exemple
list-style	Permet de définir les informations concernant les listes.	- **list-style-type** - **list-style-position** - **list-style-image**	 - Item 1 - Item 2 - Item 3
list-style-image	Permet de définir une image comme puce pour une liste.	- **none** **- url**	<style> ol { list-style-image:url(logo_puce.gif) } </style> ... - Item 1 - Item 2 - Item 3
list-style-position	Permet de définir le retrait des éléments d'une liste.	- **inside** : Retrait vers l'intérieur - **outside** : Retrait vers l'extérieur	 - Item 1 - Item 2 - Item 3
list-style-type	Permet de définir le type de puces ou de numérotation concernant les listes.	- **none** : Pas de puce - **disc** : Rond (défaut) - **circle** : Cercle square : Carré - **decimal** : 1, 2, 3…. - **decimal-leading-zero** : 01, 02, 03…. - **lower-roman** : i, ii, iii…. - **upper-roman** : I, II, III… - **lower-apha** : a, b, c… - **upper-alpha** : A, B, C…	<style> ul.cercle { ul list-style-type:circle } ul.carre { ul list-style-type:square } ol.dec { ol list-style-type:decimal } </style>

Tableau 26 : propriétés CSS concernant les listes

3.12 DIMENSIONS

3.12.1 DEFINITION

Les propriétés de dimensions permettent de contrôler la hauteur et la largeur d'un élément d'une part et l'espacement entre les lignes d'autre part.

3.12.2 PROPRIETES

Propriétés	Descriptif	Valeurs	Exemple
height	Permet de définir la hauteur d'un élément	**- auto** **- length** **- %**	`<span style="height:150px">Texte</span>`
line-height	Permet de définir l'espacement entre les lignes d'un élément.	**- normal** **- number** **- length** **- %**	`<span style="line-height:20pt">Texte</span>`

max-height	Permet de définir la hauteur maximale d'un élément.	**- none**
		- length
		- %
			`<style>`
			`p { max-height:72pt; }`
			`</style>`

max-width	Permet de définir la largeur maximale d'un élément.	**- none**
		- length
		- %
			`<style>`
			`p { max-width:75%; }`
			`</style>`

min-height	Permet de définir la hauteur minimale d'un élément.	**- length**
		- %
			`<style>`
			`p { min-height:27pt }`
			`</style>`

min-width	Permet de définir la largeur minimale d'un élément.	**- length**
		- %
			`<style>`
			`p { min-width:45px; }`
			`</style>`

width	Permet de définir la largeur d'un élément.	**- auto** **- length** **- %**	`<span style="position:relative; top:15px; left:auto;width:350px;">Texte</span>`

Tableau 27 : propriétés CSS concernant les dimensions d'un élément

Les propriétés max-height, max-width, min-height et min-width ne sont présentes que dans la norme CSS2 et peuvent entraîner des problèmes d'incompatibilité vis à vis de certain navigateur dans leur comportement. Leur utilisation doit être proscrite.

3.13 POSITIONNEMENT

3.13.1 DEFINITION

Cela permet de positionner des éléments (blocs de textes, images…) au pixel près.

Pour placer un élément, il existe quatre types de positionnement :

- Le positionnement relatif.
- Le positionnement absolu.
- Le positionnement fixe.

- Le positionnement statique.

Il est recommandé dans la mesure du possible de préférencier l'utilisation d'un positionnement relatif à tout autre type de positionnement pour faciliter la maitenance ultérieur dans l'ajout de nouveaux éléments dans le document HTML.

3.13.2 PROPRIETES

Propriétés	Descriptif	Valeurs	Exemple
position	Permet de définir le mode de positionnement d'un élément	**- absolute** : position absolue par rapport à la limite de la fenêtre (défilement possible) **- fixed** : position absolue par rapport à la limite de la fenêtre (pas de défilement possible) **- relative** : position relative par rapport à l'élément précédent. **- static** : position normale (défaut)	`<span style=" position:absolute; top:25px; left:30px; ">Texte</span>`
Top	Permet de définir le point 0 à partir du haut, en relation avec position.	- **auto** : Positionnement automatique **- length** **- %**	`<span style="position:absolute; top:20 px;">Texte</span>`
bottom	Permet de définir le point 0 à partir du bas, en relation avec position.	**- auto** : Positionnement automatique **- length** **- %**	`<span style="position:absolute; bottom:20 px;">Texte</span>`
left	Permet de définir la position du point gauche de l'élément.	**- length** **- %**	`<span style=" position:relative; left:150px; top:100px">Texte</span>`
right	Permet de définir la position du point droite d'un élément.	**- length** **- %**	`<span style=" position:relative; right:150px; top:100px">Texte</span>`

Tableau 28 : propriétés CSS concernant le positionnement d'un élément

3.14 LIENS

3.14.1 DEFINITION

Les liens vont pouvoir se présenter sous de nouvelles facettes allant du non soulignement jusqu'à les faire changer de couleur au survol du pointeur de la souris.

Syntaxe : selector:pseudo-class {property : value}

Une exception doit tout de même être prise en compte sur les liens de type HTML <A HREF="javascript">Lien</A> sont surlignés par défaut sur l'ensemble des navigateurs du marché sauf pour MSIE. Il est donc nécessaire pour ce type de lien d'utiliser l'application de style propre au surlignement d'un texte : « text-decoration ».

3.14.2 PROPRIETES

Propriétés	Descriptif	Exemple
:active	Intervient lorsqu'un utilisateur clique sur un lien	**a:active {color: #330099}**
:hover	Intervient lorsqu'un utilisateur passe la souris sur un lien.	**a: hover {color: #330099}**
:link	Permet de définir les liens non visités.	**a:link {color: #330099}**
:visited	Permet de définir l'aspect du lien lorsqu'il a déjà été visité.	**a:visited {color: #330099}**

Tableau 29 : propriétés CSS pour les liens

3.15 BARRE DE DEFILEMENT : OVERFLOW

3.15.1 DEFINITION

Permet de définir la manière dont la zone, contenant un élément trop grand, est géré vis à vis de l'apparition ou non de la barre de défilement.

3.15.2 PROPRIETES

Propriétés	Descriptif	Exemple
auto	Adaptation automatique	** <img src="image.jpg" style="overflow:scroll">**
visible	La zone parent est adaptée en fonction des dimensions de l'élément	
hidden	L'élément est rogné en fonction de la taille de la zone parent	
scroll	Des barres de défilement sont affectées à la zone	

Tableau 30 : propriétés CSS de la barre de défilement overflow

3.16 AFFICHAGE : DISPLAY

3.16.1 DEFINITION

Permet de définir le type de boîte pour l'affichage des éléments.

Ce style est important lors de la gestion de l'apparition ou non d'un élément en sachant que la valeur par défaut de ce style est vide ("") pour indiquer son apparition.

3.16.2 PROPRIETES

Propriétés	Descriptif	Exemple
block	Boîte de type bloc	`<p style=" display:none ">Texte non visible</p>` `<p style=" display:block ">Texte visible dans un bloc</p>`
inline	Boîte intra-ligne (Défaut)	
list-item	Boîte de type bloc incluant une boîte à puce	
none	Pas de boîte, contenu invisible	

Tableau 31 : propriétés CSS pour l'affichage des éléments (display)

3.17 POINTEUR : CURSOR

3.17.1 DEFINITION

Permet de définir un curseur par rapport à un élément.

Ce style est utile lors de l'utilisation d'action onClick en lieu et place de lien classique comme lors de la gestion d'un menu pour gérer l'affichage d'une main lors d'un rollover sur un texte. Il est nécessaire afin d'assurer l'affichage dans les navigateur Mozilla et IE du pointeur, de positionner la propriété "pointer" à la fois à

Il est nécessaire afin d'assurer le même affichage de la main dans les navigateur Mozilla et IE du pointeur de positionner la propriété à "pointer" et à "hand". Dans ce cas précis, le positionnement "cursor : hand" dans une feuille de style génère une non conformité au niveau du validateur CSS W3C mais cela est nécessaire afin d'obtenir le même affichage sous Mozilla et IE.

3.17.2 PROPRIETES

Propriétés	Descriptif	Exemple
url(curseur.gif)	Curseur personnalisé (gfi ou jpg)	`<span style="cursor:help;">Un peu d'aide...</span>`
auto	Définition automatique	
crosshair	Croix fine	
pointer	Pointeur hyperlien (main)	
hand	Pointeur hyperlien (main).	

move	Pointeur indiquant la possibilité de bouger un élément	
n-resize	Flèche vers le haut	
ne-resize	Flèche vers le haut et la droite	
e-resize	Flèche vers la droite	
se-resize	Flèche vers le bas et la droite	
s-resize	Flèche vers le bas	
sw-resize	Flèche vers le bas et la gauche	
w-resize	Flèche vers la gauche	
nw-resize	Flèche vers la gauche et le haut	
text	Curseur texte	
wait	Curseur symbolisant l'attente	
help	Curseur d'aide	

Tableau 32 : propriétés CSS pour les pointeurs (cursor)

4 NORME DE DEVELOPPEMENT XHTML

4.1 INTRODUCTION

Le XHTML (E**X**tensible **H**yper**T**ext **M**arkup **L**anguage), qui signifie langage de balisage hypertexte extensible, est le successeur du HTML 4.0. Il a pour but à terme de remplacer le HTML du fait qu'il soit identique au HTML mais en étant plus stricte et plus propre que lui puisqu'il reprend les caractéristiques les plus intéressantes du XML telles que la structuration et l'extensibilité des données (il est possible d' étendre les fonctions standards du langage, en employant des bibliothèques externes chargeables sous forme de module par le biais du DTD).

Ainsi, un fichier source XHTML prend en en-tête le DTD (Document Type Définition) propre à la syntaxe XML qui définit la structure des attributs et éléments qui sont utilisés dans le fichier source.

Cette norme ne reprend pas l'ensemble des balises déjà décrites dans la norme HTML à iso-périmètre, il fournit des préconisations sur celle-ci.

Dans le cadre des applicatifs du Ministère des Affaires Etrangères, le XHTML est préférencié toujours par rapport au HTML.

De plus, il est nécessaire de se reporter à l'Annexe C : Références XHTML pour vérifier avant utilisation le cadre d'utilisation d'une balise ou d'un attribut pour savoir la pertinence de son utilisation ou savoir si un attribut doit être déporté dans une feuille de style.

4.2 DIFFERENCES ENTRE LE HTML ET LE XHTML

4.2.1 FORMULATION DES TAGS

Le langage XHTML demande aux développeurs de formuler correctement les documents balisés.

Dans HTML, il était jusqu'alors possible de commettre des fautes syntaxiques dans les documents sans que leur fonctionnement soit réellement perturbé.

Dans le langage XHTML, certaines erreurs autrefois tolérées, sont désormais rédhibitoires à un fonctionnement correct de la page web.

En conséquence, la rédaction des documents XHTML demande plus de précaution et d'application.

La formulation d'un document XHTML doit obéir à plusieurs règles incontournables.

Le chevauchement des éléments est illégal

```
<h1>Eléments <u>correctement emboités</u></h1>
<h1>Eléments <u>incorrectement emboités</h1></u>
```

Les noms des éléments et des attributs doivent être écrits en minuscule

```
<p id="p1">Balisage correct</p>
<p id="p1">Balisage incorrect</p>
```

Toutes les balises non-vides doivent posséder une balise de fin

```
<li>Elément liste correct</li>
<li>Elément liste incorrect
```

Les éléments vides doivent également posséder une indication de fin

```
<img src="http://www.google.com" alt="élément correct"/>
<img src="http://www.google.com" alt="élément correct"></img>
<img src="http://www.google.com" alt="élément incorrect">
```

Toutes les valeurs d'attributs doivent être encadrées par des guillemets

```
<img src="http://www.google.com" height="120" width="60" alt="Rédaction
correcte">
<img src="http://www.google.com " height=120 width=60 alt=Rédaction
incorrecte>
```

L'usage des simples ou doubles quotes est permis. Si dans la valeur de l'attribut une simple quote apparaît alors il convient d'englober le nom entre double quote et inversement.

La minimisation des attributs est interdite

```
<option sected="selected">Attribut correct</option>
<option selected>Attribut incorrect</option>
```

Les éléments *script* et *style* sont déclarés comme possédant un contenu de données textuelles analysées (PCDATA : Parsed Character DATA)

```
<script type="text/javascript">
   ![CDATA[prix = compte <= 1000 ? 500 : 450;]]>
</script>
```

L'attribut *name* doit être progressivement abandonné au bénéfice de l'identificateur *id*.

```
<form id="formulaire_correct">...</form>
<form name="formulaire_incorrect">...</form>
```

Si l'attribut *name* est nécessaire pour implémenter du DOM Level 0, il est alors nécessaire de le doubler avec l'attribut *id* renseigné avec la même valeur.

```
<form id="formulaire_correct" name="formulaire_correct">...</form>
```

Automatiquement, les développeurs suppriment les espaces de début et de fin des valeurs d'attributs et si plusieurs espaces se suivent, ils sont remplacés par un unique espace

```
attribut="  valeur   de  l'attribut   "
```

```
<!-- donne après analyse --> attribut="valeur de l'attribut"
```

XHTML interdit l'inclusion de certains éléments dans d'autres

- *a* ne peut contenir d'autres éléments *a*,
- *pre* ne peut contenir les éléments *big*, *img*, *object*, *small*, *sub* ou *sup*,
- *button* ne peut contenir les éléments *button*, *form*, *fieldset*, *iframe*, *input*, *isindex*, *label*, *select* ou *textarea*,
- *label* ne peut contenir d'autres éléments *label*,
- *form* ne peut contenir d'autre éléments *form*.

4.2.2 FORMAT DU DOCUMENT

XHTML 1.0 est décliné en trois variantes que le W3C désigne sous les noms Strict, Transitional et Frameset. Dans cet article, je me concentre sur les deux plus utiles, à savoir les variantes Strict et Transitional.

XHTML 1.0 Strict

XHTML 1.0 Strict est la variante la plus contraignante du XHTML, mais elle produit le balisage structurel le plus propre. Le code Strict est dépourvu de tout balisage servant à définir la mise en page. Il utilise les feuilles de style en cascade (CSS) pour contrôler la présentation. C'est grâce à cette séparation de la structure et de la présentation que le XHTML Strict est suffisamment flexible pour s'afficher sur différents appareils. Le recours aux feuilles de style en cascade pour contrôler la présentation peut s'avérer problématique pour les développeurs; en effet, ce choix n'est pas judicieux pour le contenu web qui doit être visualisé sur des appareils ou dans des navigateurs qui ne reconnaissent pas les feuilles de style.

XHTML 1.0 Transitional

XHTML 1.0 Transitional est la variante la plus conciliante du XHTML. Contrairement à la variante Strict, qui sépare totalement la structure de la présentation, Transitional vous permet d'utiliser des balises pour contrôler l'aspect de votre mise en page. Elle vise à offrir une solution intermédiaire entre les pages HTML, dont les balises contrôlent la présentation, et XHTML Strict, qui ne le permet pas. Le principal avantage de la variante Transitional est de pallier la dépendance aux CSS de Strict. Les utilisateurs qui ont recours à des navigateurs de versions antérieures ou qui possèdent des appareils ne reconnaissant pas les feuilles de style peuvent toujours accéder aux pages Transitional.

XHTML 1.0 Frameset

XHTML 1.0 Frameset est la variante qui permet d'utiliser des frames [cadres] pour diviser la fenêtre du navigateur en plusieurs parties de fenêtre, appelées des frames. En fait, le DTD Frameset inclut le DTD Transitional et les balises des frames.

Cette DTD correspond à une version XHTML qui tolère l'élément "frame" en plus des autres éléments déconseillés bien qu'encore spécifiés.

Dans le cadre du projet deux des DTD ci-dessus détaillée sont utilisées :

- Frameset : pour les pages avec frames. Néanmoins, l'utilisation des deux attributs "frameborder"

et "framespacing" (cf. GDFB) de la balise <frameset> afin d'éliminer tout bord avec le cadre rendent la page non valide par le validateur W3C. En CSS il n'est pas possible de gérer ce rendu. Ces deux attributs sont nécessaire afin de produire le même effet dans les naivgateurs cibles IE et Mozilla. Mozilla n'est pas sensilble à l'attribut "framespacing" alors que IE ne l'est pas avec "frameborder ».

- Strict : pour le reste des pages. Strict permet un découplage complet entre le contenu XHTML et le rendu en terme de feuille de style CSS.

- La feuille de style Transitional pourra s'avérer nécessaire dans certains cas de réalisation mais son utilisation devra être validée avec le chef de projet après prise en compte du cas particulier. Même dans ce cas précis, le maximum d'information devra être déporté dans la feuille de style.

4.2.3 VALIDATION DU DOCUMENT PAR DTD

La validation d'un document XHTML est accordée par le processeur XHTML si le document est bien formé et s'il est conforme à la DTD associée.

Un document est dit bien formé s'il correspond parfaitement aux règles de bonne formulation.

C'est-à-dire, si le document ne comporte aucune erreur de syntaxes comme la casse des éléments et attributs, leur emboîtement, la fermeture impérative des éléments vides et non vides, l'encadrement des valeurs d'attributs par des guillemets, etc..

La conformité d'un document XHTML par rapport à une Définition de Type de Document est effective lorsqu'il n'existe, dans ce document, aucune transgression des règles définies par la DTD.

D'ailleurs, une Définition de Type de Document doit être spécifiée impérativement avant l'élément racine <html> du document XHTML par le truchement de l'instruction <!DOCTYPE>. La DTD peut être l'une des trois variantes recommandées par le World Wide Web Consortium (W3C) ou une autre DTD personnelle créée pour une utilisation spécifique.

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

Si le document respecte l'architecture énoncée par la DTD, comme le strict respect des types de données (datatypes), des valeurs d'attributs ou d'éléments, l'utilisation pertinente des composants en fonction de leur caractère obligatoire ou optionnel ou encore de ne pas employer d'éléments ou d'attributs non définis.

L'élément racine du document doit être <html>, lequel doit posséder un espace de noms XHTML par l'intermédiaire de l'attribut xmlns.

```
<html xmlns="http://www.w3.org/1999/xhtml">
    ...
</html>
```

L'élément d'entête <head> doit obligatoirement apparaître avec en son sein la balise <title>.

```
<head>
  <title>Titre</title>
</head>
```

Un document XHTML minimal doit impérativement comporter une déclaration DOCTYPE suivie de l'élément <html> lequel comprend une balise <body> précédée par <head> et <title>.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/dtD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <title>Titre du document</title>
  </head>
  <body>
  <!-- Contenu du document -->
  </body>
</html>
```

Les attributs de langue peuvent apparaître dans n'importe quelle balise XHTML hormis : base, br, frame, frameset, hr, iframe, param, et script.

```
<html xml:lang="FR" lang="FR">...</html>
<div xml:lang="FR" lang="French [fr]">...</div>
```

L'utilisation de l'attribut de langue n'est pas nécessaire est risque de pauser des problèmes de mise en œuvre lors de la mise en place ou de l'évolution d'un applicatif en multi-langues dans l'archtecture définie dans le projet A3 basée sur le client riche. Cet attribut figure dans notre cas dans le rappatriement des données ou des libellés sous format XML.

4.3 XHTML SUPPORT DU CSS

4.3.1 ILLUSTRATION SUR UN CAS D'UTILISATION : NAVIGATION PAR ONGLETS

Lors de la création d'une présentation avec la seule utilisation de CSS, il faut choisir le code XHTML qui sera le support. Ce code doit *avoir un sens*, c'est la **sémantique**. Afin de comprendre les tenants et aboutissants il convient de prendre un exemple et prenons le cas de la navigation par onlgets.Dans le cas de cette navigation avec onglets, le rendu souhaité est **une liste horizontale** de liens, avec un **rendu en tabs ou onglets** avec un onglet sélectionné.

On souhaite présenter une liste, voici le code XHTML d'une liste de liens :

```
<ul id="tabnav">
   <li class="active"><a href="#">Voiture</a></li>
   <li><a href="#">Vélo</a></li>
```

```
   <li><a href="#">Avion</a></li>
   <li><a href="#">Bateau</a></li>
   <li><a href="#">Pieds</a></li>
</ul>
```

Les deux éléments importants sont l'**activation** d'un lien avec **class="active"**, et l'attribution d'un identifiant unique dans toute la page à la liste **id="tabnav"**, pour ne styler que cette liste.

4.3.2 CHOIX DU TYPE DE RENDU

Voici un exemple de présentation de cette liste en onlgets.

On choisit une méthode pour *mettre en ligne* les éléments de la liste. On choisit un rendu de l'onglet actif différent des autres, et finalement on propose un changement de rendu des autres onglets au passage de la souris.

Les points clefs de l'implémentation sont les suivants :

```
ul#tabnav {
   font: bold 11px verdana, arial, sans-serif;
   list-style-type: none;
   padding-bottom: 24px;
   border-bottom: 1px solid #6c6;
   margin: 0;
}
```

On ne prend aucun des styles par défaut de la liste, et on met une bordure verte sur le bas.

```
ul#tabnav li {
   float: left;
   height: 21px;
   background-color: #cfc;
   margin: 2px 2px 0 2px;
   border: 1px solid #6c6;
}
```

Pour un onglet normal, on le rend flottant sur la gauche. Les éléments ne seront donc plus les uns en dessous des autres, mais accolés les uns aux autres sur une ligne.

```
ul#tabnav li.active {
   border-bottom: 1px solid #fff;
   background-color: #fff;
}
```

Pour l'onglet actif, on écrase la bordure définie dans le **ul#tabnav** par une bordure blanche, mais uniquement au niveau de l'onglet. On change aussi la couleur de fond.

```
#tabnav a {
```

```
    float: left;
    display: block;
    color: #666;
    text-decoration: none;
    padding: 4px;
}
```

On force l'empilement aussi par la gauche, mais surtout on fait un affichage **block** ce qui permet de cliquer partout dans l'onglet pour activer le lien.

```
#tabnav a:hover {
    background: #fff;
}
```

Petit effet pour changer la couleur de fond en passant la souris sur un onglet. Comme vous voyez, le code est simple propre et élégant.

5 NORME DE DEVELOPPEMENT JAVASCRIPT

5.1 REFERENCES ET SPECIFICATIONS DU JAVASCRIPT

5.1.1 JAVASCRIPT/JSCRIPT/ECMASCRIPT

A l'origine, le langage JavaScript a été développé par Netscape pour sa version 2.0 de son navigateur. Un besoin de standardisation de ce langage a été exprimé rapidement lorsque Microsoft de son côté a créé aussi son propre langage de script : JScript.

De ce fait, un organisme indépendant a été chargé de spécifier la standardisation des langages de script : ECMA (European Computer Manufacturers Association). La spécification du standard des langages script s'appelle ECMAScript en sachant que la première version a pour nom ECMA-262 spécification. Cette spécification ne prend pas en compte la gestion des objets DOM car elle a été confiée un autre organisme indépendant : W3C.

Les équivalences entre JavaScript et ECMAScript sont fournies sur le site de Netscape :

JavaScript version	Relationship to ECMA version
JavaScript 1.1	ECMA-262, Edition 1 is based on JavaScript 1.1.
JavaScript 1.2	ECMA-262 was not complete when JavaScript 1.2 was released. JavaScript 1.2 is not fully compatible with ECMA-262, Edition 1, for the following reasons: • Netscape developed additional features in JavaScript 1.2 that were not considered for ECMA-262. • ECMA-262 adds two new features: internationalization using Unicode, and uniform behavior across all platforms. Several features of JavaScript 1.2, such as the `Date` object, were platform-dependent and used platform-specific behavior.
JavaScript 1.3	JavaScript 1.3 is fully compatible with ECMA-262, Edition 1. JavaScript 1.3 resolved the inconsistencies that JavaScript 1.2 had with ECMA-262, while keeping all the additional features of JavaScript 1.2 except == and !=, which were changed to conform with ECMA-262.
JavaScript 1.4	JavaScript 1.4 is fully compatible with ECMA-262, Edition 1. The third version of the ECMA specification was not finalized when JavaScript 1.4 was released.
JavaScript 1.5	JavaScript 1.5 is fully compatible with ECMA-262, Edition 3.

Tableau 33 : équivalences entre JavaScript et ECMAScript

Une version 2.0 du JavaScript est en cours d'implémentation chez Netscape, elle ne fait donc pas partie pour l'instant de ce document.

Les versions de JavaScript liées aux navigateurs Netscape :

Netscape Browser	1.0	1.1	1.2	1.3	1.4	1.5	2.0
Netscape 2.0	x						
Netscape 3.0		x					
Netscape 4.0			x				
Netscape 4.5				x			
Netscape 6.X						x	
Netscape 7.0						x	

Tableau 34 : versions de JavaScript vis à vis des navigateurs Netscape

Pour garder une compatibilité vis à vis de la cible des navigateurs Netscape fixée, il est nécessaire d'implémenter l'applicatif en version compatible JavaScript 1.5 au maximum. Cette contrainte est aussi applicable pour la compatibilité vis à vis des navigateurs Gecko (Mozilla, Chimera, Konqueror, K-Meleon…). Mais pour une meilleure adaptabilité du marché, il est préférable de se limiter à une version compatible JavaScript 1.3 au maximum (Opera, Safari …). (Voir annexe Annexe E : Références JavaScript pour détail des directives compatibles)

Les équivalences entre JScript et ECMAScript sont plus complexes du fait que JScript n'est pas seulement un langage lié à la programmation Internet mais peut servir aussi de composant autonome pour Windows. Ainsi, les équivalences sont liées au cas par cas et vous sont fournies par le site de Microsoft (voir Annexe D : Références JScript)

Les versions de JScript liées aux logiciels Microsoft sont les suivantes :

Application/Version JScript	1.0	2.0	3.0	4.0	5.0	5.1	5.5	5.6
Microsoft Internet Explorer 3.0	x							
Microsoft Internet Information Server 3.0		x						
Microsoft Internet Explorer 4.0			x					
Microsoft Internet Information Server 4.0			x					
Microsoft Visual Studio 6.0				x				
Microsoft Internet Explorer 5.0					x			
Microsoft Internet Explorer 5.01						x		
Microsoft Windows 2000						x		
Microsoft Internet Explorer 5.5							x	
Microsoft Windows Millennium Edition							x	
Microsoft Internet Explorer 6.0								x
Microsoft Windows XP								x

Tableau 35 : versions de JScript vis à vis des logiciels Microsoft

Pour garder une compatibilité vis à vis de la cible des navigateurs MSIE fixée, il est nécessaire d'implémenter l'applicatif en version compatible JScript 5.0 au maximum (Voir Annexe D : Références JScript pour détail des directives compatibles).

5.1.2 DOM LEVEL 0

Le Document Object Model (DOM) est la modélisation qui décrit l'ensemble des éléments constituant la page HTML. Ainsi, en appelant un élément de la page par son nom DOM, nous pouvons l'influer.

Historiquement, ce besoin est apparu dès la naissance du langage JavaScript. Ainsi, Netscape a commencé à insérer dans ce langage la possibilité d'accéder à quelques éléments HTML, les principaux éléments d'un formulaire et après les images, la fenêtre du navigateur et la configuration du navigateur. Cette première vision du DOM était une modélisation sous forme de collection sur l'élément parent de la page HTML : l'objet document ou window eux-mêmes. Cette vision du DOM est communément appelée DOM Level 0 ou encore Original DOM.

Le DOM Level 0 a été communément intégrée pour des raisons de compatibilité par les autres concurrents du marché (Ex : Microsoft lors de la sortie de la version 3.0 d'Explorer). Même si cette vision limitée du DOM a fortement évolué depuis, l'ensemble des éditeurs du marché continue à l'intégrer dans l'évolution de leur navigateur.

Ainsi, l'applicatif utilise un certain nombre de directives du DOM Level 0 compatibles avec la cible des navigateurs fixée (Voir Annexe F : Références DOM Level 0 pour liste des directives compatibles).

Cette vision du DOM a été bouleversée avec l'arrivée sur le marché des versions 4 des navigateurs. Un nouveau besoin est apparu d'ajout de dynamismes sur une partie du document HTML. Ce besoin appelé communément DHTML nécessitait de découper un document HTML en zones dynamiques.

A ce moment là, une « guerre intense » entre les deux éditeurs principaux du marché s'est déclenchée pour imposer la version 4 de leur produit. Ainsi, la zone dynamique devant découper un document HTML reposait pour Netscape sur la notion de « layer » et pour Microsoft sur la notion de « DIV ».

De ce fait, la stratégie autour du DOM a été partagée entre :

- Pour Netscape, l'enrichissement du DOM Level 0 en offrant une nouvelle collection au document : document.layers.
- Pour Microsoft, la compatibilité du DOM Level 0 et la création d'une norme propriétaire appelée DOM DHTML basée sur la nouvelle collection au document : document.all. Cette nouvelle norme propriétaire à Microsoft reprenait ainsi les besoins du DOM Level 0 et répondait au nouveau besoin de DHTML dans les pages. De plus, avec l'arrivée de la version 5 de leur navigateur et d'un nouveau format d'échange XML, elle s'est vue renforcer d'un certain nombre de fonctionnalités supplémentaires.

Ainsi par le passé, l'ajout de dynamismes dans les pages pour améliorer l'ergonomie du site entraînait le développement d'un code cross-browser permettant d'intégrer ces deux stratégies autour d'un même code. Ce code cross-browser distinguait ainsi l'implémentation spécifique à Microsoft et à Netscape autour de clause if et mutualisait l'implémentation commune aux deux éditeurs.

L'arrivée d'un standard préconisé par un organisme indépendant W3C (décrit ci-dessous) a offert la possibilité à l'applicatif de s'affranchir des modélisations DOM propriétaires pour tendre vers un DOM W3C.

Ainsi, l'applicatif ne doit plus implémenter de code spécifique à un éditeur ni effectuer aucune distinction entre type et version de navigateur.

5.1.3 DOM W3C

Les normes propriétaires sur le DOM nécessitaient d'effectuer un consensus pour qu'un standard puisse émerger. Dans ce contexte, l'organisme indépendant W3C a créé les recommandations du DOM Level 1 originairement prévu pour les langages de programmation d'accès à un document XML. Du fait que le XML et le HTML découlent d'un même standard le SGML, il a été facile d'élargir cette norme pour intégrer les documents HTML.

La vision du DOM W3C est de permettre le lire le texte et les attributs de chaque balise HTML de son document et d'ainsi permettre de détruire des balises et leurs contenus, de créer de nouvelle balise et de les intégrer dans le document. Ceci offre donc la possibilité de réécrire une page sans aller-retour avec le serveur Web.

Le DOM W3C repose donc sur une modélisation arborescente où chaque objet du document HTML est un nœud (Node en anglais) de cette arborescence. Ainsi, il est facile de se déplacer dans cette arborescence pour atteindre le nœud désiré et influer sur celui-ci.

Depuis cette première version du DOM W3C, une nouvelle préconisation DOM W3C Level 2 a émergé. Cette version reprend les avancés du DOM Level 1 en les découpant par spécialités et en ajoutant certaines fonctionnalités :

- DOM Level 2 HTML : spécifications pour l'accès et la mise à jour dynamiques d'un document HTML.
- DOM Level 2 Core : spécifications pour l'accès et la mise à jour dynamiques d'un document XML (Non utiles pour la mise en œuvre d'IHM sur client léger mais lors de la mise en œuvre des principes d'un client riche).
- DOM Level 2 Style : spécifications pour l'accès et la mise à jour dynamiques d'un document de type style sheets.
- DOM Level 2 Events : spécifications pour une gestion générique des événements (voir §5.2).
- DOM Level 2 Traversal and Range : (Non utilisées pour compatibilité vis à vis des navigateurs ciblés).
- DOM Level 2 Views : spécifications pour l'accès et la mise à jour dynamiques d'une représentation d'un document (Non utilisées pour compatibilité vis à vis des navigateurs ciblés).

Ainsi, l'applicatif utilise les préconisations W3C DOM Level 2 compatibles avec les navigateurs cibles pour la réalisation JavaScript (voir Annexe G : Références DOM W3C pour liste des directives JavaScript liées au DOM W3C compatibles à la msie en œuvre d'un client léger).

5.2 GESTION D'EVENEMENTS

Lorsqu'un utilisateur entreprend une action, un événement est associé à la page. Il existe aussi des événements non liés à une action de l'utilisateur comme par exemple l'événement « load » lors du chargement d'une page.

Le langage JavaScript permet de détecter ces événements pour déclencher des traitements sur le navigateur client à ce moment là.

5.2.1 LES EVENEMENTS

Cette liste n'est pas exhaustive mais a pour but de préciser ceux communément utilisés (voir Annexe H : Références des événements JS pour liste exhaustive et compatibilité).

5.2.1.1 LES EVENEMENTS LIES A LA SOURIS

Les événements liés à la souris pouvant être gérés sont :

- click : déclenché lors d'un click de la souris.

- dblclick : déclenché lors d'un double click de la souris.
- mouseover : déclenché lors de l'entrée du curseur de la souris dans l'aire gérée.
- mouseout : déclenché lors de la sortie du curseur de la souris dans l'aire gérée.
- mousemove : déclenché lors du passage du curseur de la souris dans l'aire gérée.
- mousedown : pression du bouton de la souris.
- mouseup : relachement du bouton de la souris.

5.2.1.2 LES EVENEMENTS LIES AU CLAVIER

Les événements liés au clavier pouvant être gérés sont :

- keydown : pression sur une touche du clavier.
- keypress : utilisation d'une touche du clavier.
- keyup : relachement d'une touche du clavier.

5.2.1.3 LES EVENEMENTS LIES A UN FORMULAIRE OU A UN ELEMENT DU FORMULAIRE

Les événements liés à un formulaire ou à un élément du formulaire pouvant être gérés sont :

- submit : validation d'un formulaire.
- reset : annulation d'un formulaire.
- focus : focus reçu sur un élément du formulaire.
- blur : focus perdu sur un élément du formulaire.
- change : changement de valeur d'un élément du formulaire.

5.2.1.4 LES EVENEMENTS LIES A UNE FENETRE

Les événements liés à à une fenêtre pouvant être gérés sont :

- load : chargement complèt d'une page.
- unload : déchargement d'une page lors de changement pour une autre page ou fermeture de la fenêtre.
- resize : redimensionnement de la fenêtre.

5.2.2 ACCES A UN EVENEMENT

A tout événement est lié un traitement JavaScript.

Ce traitement a pour canevas :

```
function doSomething(e)
{
  //Code du traitement
}
```

De plus, ce traitement JavaScript peut avoir besoin d'accéder à l'événement pour interroger ses propriétés. Mais, l'accès à un événement diffère suivant la modélisation W3C ou celle de Microsoft. Ainsi, pour W3C "e" contient l'événement alors que pour Microsoft, il faut utiliser "window.event".

Pour pouvoir interroger un événement, il est donc nécessaire d'effectuer une détection non pas de navigateur mais de modélisation (voir §5.2.3 pour exemple d'implémentation).

5.2.3 LES PROPRIETES LIEES A UN EVENEMENT

5.2.3.1 LES PROPRIETES LIEES A UN EVENEMENT

Les propriétés liées à un événement permettent de répondre à certaines questions que le développeur peut se poser lors de l'implémentation du traitement lié.

5.2.3.2 QUEL EST LE TYPE DE L'EVENEMENT ?

La propriété de l'événement à implémenter est ".type".

Exemple :

```
function doSomething(e)
{
  var elm = e ? e.target : event.srcElement;
}
```

5.2.3.3 QUEL CARACTERE A UTILISE L'UTILISATEUR SUR SON CLAVIER ?

De même pour cette question, il est nécessaire d'ajouter une détection non pas de navigateur mais de modélisation suivant W3C pour implémenter la propriété ".keyCode" ou d'autres navigateurs plus anciens pour la propriété ".which".

Il est aussi possible de détecter les caractères spéciaux avec les propriétés normalisées ".shiftKey", ".altKey" et ".ctrlKey".

Exemple :

```
function doSomething(e)
{
  var key = e ? (e.which ? e.which : e.keyCode) : event.keyCode;
}
```

5.2.3.4 QUEL BOUTON A UTILISE L'UTILISATEUR SUR SA SOURIS ?

Il est vrai qu'il existe plusieurs propriétés accessibles pour répondre à cette question mais celles-ci de part leur implémentation ou de part la valeur qu'elles renvoient, aucun consensus n'est applicable.

Cette règle technique ne doit donc pas être implémentée sur un applicatif.

5.2.3.5 QUELLE ETAIT LA POSITION DE LA SOURIS LORS DU DECLENCHEMENT DE L'EVENEMENT ?

Cette question est soumise aux mêmes contraintes que pour la question précédente et ne doit pas être implémentée sur un applicatif tant qu'aucune recommandation W3C ne statue pas sur une solution

standardisée.

5.2.4 GESTIONNAIRE D'EVENEMENTS

Sur l'ensemble des navigateurs du marché, un gestionnaire d'événements permet de pister les événements sur des éléments du document HTML ou sur le document lui-même ou sur la fenêtre du navigateur.

Il est donc nécessaire pour le développeur d'enregistrer dans ce gestionnaire l'événement désiré et l'élément du document HTML sur lequel cet événement est associé. Pour effectuer cette implémentation, plusieurs modélisations sont possibles :

1. Enregistrement en ligne sur le document HTML.
2. Enregistrement traditionnel en JavaScript.
3. Enregistrement avancé selon la modélisation W3C ou la modélisation Microsoft.

Dans cette partie, nous allons décrire les différentes modélisations pour pouvoir préconiser celles qui sont applicables sur un applicatif compatible avec l'ensemble des navigateurs ciblés.

5.2.4.1 ENREGISTREMENT EN LIGNE

L'enregistrement en ligne consiste à intégrer l'événement dans la balise HTML. L'événement est déclaré dans cette balise en ajoutant le préfixe « on ». Les événements propres au document HTML sont rattachés eux à la balise <BODY>.

Par convention de nommage, il est préférable de distinguer par des majuscules et des minuscules le nom des événements (Ex : onClick, onMouseOver…) pour faciliter la relecture du code. Cette mise en valeur n'influe pas sur le comportement du code HTML car contrairement au langage JavaScript, le langage HTML n'est pas « case sensitive ».

La valeur de cet attribut sur la balise HTML contient l'appel à la fonction JS implémentant le traitement lié à l'événement. Si plusieurs traitements distincts sont associés à cet événement, il suffit de séparer par une virgule les appels aux différentes fonctions JS. Il est aussi possible d'attacher directement le code simple d'une instruction JavaScript à l'événement en utilisant l'appel de la fonction JS référencée "eval('Code de l'instruction')".

Dans le code du traitement JS, il est possible d'influer sur l'élément HTML rattaché à l'événement en l'interrogeant par l'objet référencé « this ». Cet objet pointe sur l'objet DOM lié à l'élément HTML. Ceci permet d'attacher le même traitement JS à plusieurs éléments HTML en les distinguant par l'objet « this ».

La liste des événements compatibles avec l'applicatif et les conditions d'application du rattachement d'un événement à un élément HTML sont précisées dans l'Annexe H : Références des événements JS.

Exemple :

```
<body onLoad="doSomethingChargement()" onResize="doSomethingResize()">

<a href="javascript:doSomething1(param1,param2)" onMouseOut="
doSomething1(param1,param2)" onMouseOver=" doSomething3(param1,param2)"
onClick="doSomething4(param1,param2)" style="text-decoration:none">

<form id="Formulaire">
<select id="select1"
onChange="eval('window.location=document.Formulaire.select1[document.Formulaire.selec
t1.options.selectedIndex].value')">
//Liste des options du select dont les valeurs sont des urls
</form>
```

5.2.4.2 ENREGISTREMENT TRADITIONNEL

L'enregistrement traditionnel ressemble à l'enregistrement en ligne. La différence se situe sur le fait qu'il est utilisé dans un code JavaScript et non pas dans un code HTML.

Ainsi, il consiste à rattacher un événement à un élément HTML en ne le déclarant pas comme un attribut d'une balise HTML mais en le déclarant dans un code JavaScript comme une propriété de l'objet DOM.

Exemple :

```
element.onclick = doSomething;
```

Contrairement au code HTML, le code JavaScript est « case sensitive » et le nom de l'événement doit donc être en minuscule. De plus, lorsque le traitement JS rattaché à l'événement ne nécessite pas d'argument en entrée, il n'est pas nécessaire d'ajouter les « () » à l'appel de la fonction JS.

Attention, lors de l'enregistrement de plusieurs traitements sur un même événement pour un même élément HTML, seul le dernier enregistrement est pris en compte par le gestionnaire d'événements. Pour pouvoir effectuer plusieurs traitements sur un même événement pour un même élément HTML, il est nécessaire d'encapsuler les traitements dans une fonction.

Exemple :

```
element.onclick = doSomething1;
element.onclick = doSomething2;
// doSomething2 est exécutée lors du déclenchement de l'événement

element.onclick = function () {doSomething1(); doSomething2();};
// doSomething1 et doSomething2 sont exécutées lors du déclenchement de l'événement
```

Il est aussi possible de retirer un enregistrement sur le gestionnaire d'événement en valorisant l'événement sur l'élément HTML à « null ».

Exemple :

```
element.onclick = null;
```

L'enregistrement traditionnel suit les même contraintes d'application que l'enregistrement en ligne (voir Annexe H : Références des événements JS pour compatibilité) et utilise le même objet référencé « this » pour pointer sur l'élément HTML associé à l'événement.

5.2.4.3 ENREGISTREMENT AVANCE

L'enregistrement avancé repose sur une modélisation proche de l'enregistrement traditionnel sauf qu'il intègre la gestion de l'ordonnancement des événements lorsque plusieurs traitements sont déclarés pour un même événement.

A l'heure actuelle, ce type d'enregistrement s'implémente suivant deux modélisations distinctes :

1. les spécifications DOM Level 2 Event.
2. les spécifications Microsoft.

Ainsi, la gestion d'un événement dans le gestionnaire d'événement selon la modélisation du DOM Level 2 Event s'implémente de la manière suivante :

```
//Ajout d'un événement
element.addEventlistener('name event',doSomething,boolean);

//Suppression d'un événement
element.removeEventlistener('name event',doSomething,boolean);
```

Le "boolean" en paramètre de ces fonctions est expliqué dans le §5.2.5 pour gérer l'ordonnancement.

La gestion d'un événement dans le gestionnaire d'événement selon la modélisation Microsoft s'implémente de la manière suivante :

```
//Ajout d'un événement
element.attachEvent('name event',doSomething);

//Suppression d'un événement
element.detachEvent('name event',doSomething);
```

Lors de la déclaration de plusieurs traitements pour un même événement sur un même élément contrairement à l'enregistrement traditionnel, les deux traitements sont enregistrés (Voir §5.2.5 pour ordonnancement).

5.2.4.4 ENREGISTREMENTS PRECONISES

Tant que l'enregistrement avancé n'obtient pas un consensus des navigateurs du marché, l'applicatif implémente seulement les enregistrements en ligne et les enregistrements traditionnels.

5.2.5 ORDONNANCEMENT DES EVENEMENTS

L'ordonnancement des événements concerne la propagation des événements dans le gestionnaire d'événements. Ainsi, lors de la déclaration d'un même événement sur un élément de la page et son parent, l'ordonnancement des événements spécifie lequel intervient en premier.

Pour illustrer ces notions, nous prendrons pour exemple le cas de la définition d'un événement "click" sur un tableau HTML et aussi d'un événement "click" sur une ligne de ce tableau. Quel traitement associé à ces événements est déclenché en premier lors du click par l'utilisateur sur la ligne de ce tableau, celui du tableau ou celui de la ligne ?

A l'origine, deux modélisations de l'ordonnancement ont vu le jour lors de la sortie des versions 4 des principaux navigateurs :

1. Pour Netscape 4.X, le traitement lié au tableau était déclenché en premier. Cet ordonnancement est appelé en anglais phase de « capturing ».
2. Pour Microsoft, le traitement lié à la ligne était déclenché en premier. Cet ordonnancement est appelé en anglais phase de « bubbling ».

Ces modélisations ont été intégrées dans les préconisations du W3C DOM Level 2 Event. Ainsi, l'utilisation du booléen spécifié lors de l'enregistrement avancé permet de préciser le type d'ordonnancement désiré par « true » pour une phase de « capturing » et par false pour une phase de « bubbling ».

La non compatibilité (vis à vis de Microsoft) de cette modélisation W3C avec l'applicatif implique l'utilisation d'un ordonnancement de type « bubbling ».

D'autant plus que la phase de « bubbling » est celle implémentée lors d'un enregistrement en ligne ou traditionnel sur l'ensemble des navigateurs cibles depuis la sortie des navigateurs cibles de Netscape 4.X.

L'application d'un ordonnancement de type « bubbling » permet de pouvoir arrêter la propagation d'un événement lié à un élément HTML sur les autres éléments HTML parent en implémentant les directives "event.cancelBubble = true" pour la modélisation Microsoft d'accès à un événement et "e.stopPropagation()" pour la modélisation W3C d'accès à un événement.

Pour éviter une distinction de navigateur et avoir tout de même un code cross-browser, il est nécessaire de l'implémenter de la manière suivante :

```
function doSomething(e)
{
  if (!e)
    event.cancelBubble = true;
  else
    e.stopPropagation();
}
```

5.3 CONVENTION D'UTILISATION JS

5.3.1 EMPLACEMENT DU CODE JAVASCRIPT

Tous les scripts en JavaScript doivent faire partie intégrante d'un document en HTML. C'est pourquoi a été ajouté au standard HTML la balise : **script**.

Cette balise accepte en attribut :

- **src** : valeur précisée sous forme d'URL de fichier contenant le code source en JavaScript. Ce fichier doit comporter l'extension « .js ».
- **type** : indique le type de langage utilisé dans le script. Dans notre cas, sa valeur est généralement "text/javascript".

Attention : contrairement à la norme XHTML autorisant l'utilisation des balises fermantes de type <script></script> ou <script /> pour un élément vide, la balise script doit forcément être sous la forme <script></script> pour un élément vide pour garder une compatibilité vis à vis d'IE. Cette restriction concerne seulement cette balise.

Ainsi, le code JavaScript d'une page peut être défini soit dans un fichier externe soit directement dans le fichier HTML. Il est préférable de déporter le maximum de code dans des fichiers externes par soucis de mutualiser le code sous forme de bibliothèque et de diminuer les coûts de maintenance (ne nécessite pas de recompilation des pages générées sur le serveur pour une modification JS et diminue les coûts de livraison, de tests de non-régression et de risques inhérents au langage de génération des pages dynamiques sur le serveur). Lorsque le script est de longueur limitée (pas plus de 10 lignes) et spécifique à une page, il peut être intégré dans la page pour diminuer le nombre de requêtes TCP.

Exemple d'implémentation pour insérer une bibliothèque JavaScript externe :

```
<script src="url" type="text/javascript"></script>
```

De plus, il est préférable d'intégrer le code JavaScript dans l'en-tête des fichiers en HTML. Cette technique permet de faire en sorte que toutes les fonctions aient été « analysées » avant que les événements utilisateur puissent invoquer l'une d'entre elles. Elle est particulièrement adaptée aux cas où une fonction n'est pas définie au bon endroit du gestionnaire d'événements et peut être invoquée sans avoir été évaluée et sans que le navigateur sache qu'elle existe. Lorsque c'est le cas, un message d'erreur apparaît à l'écran.

Exemple d'implémentation pour insérer du code JavaScript dans une page :

```
<head>
…
<script type="text/javascript">
//Code JavaScript
</script>
…
</head>
```

Pour spécifier l'emplacement du code lors d'une gestion en multi-fenêtrage, l'implémentation est détaillée dans le §5.9.1.

Le codage des bibliothèques JavaScript doit prendre en compte un certain nombre de préconisations :

- Prendre en compte la portée des variables (locales, globales).
- Préciser les besoins vis à vis des variables (Entrée/Sortie, Optionnelles/Obligatoires, Paramètres/Arguments) pour gérer la robustesse du code.
- Faciliter la maintenance dans le découpage, la mutualisation et l'encapsulation des fonctions.

5.3.2 APPEL DE FONCTION JAVASCRIPT DANS UN FICHIER EN HTML

Trois techniques sont possibles pour appeler une fonction JS dans un fichier en HTML :

1. Utilisation de la balise <script> à l'emplacement désiré dans le fichier en HTML.
2. Utilisation d'un lien <a href="javascript:nomDeFonction(paramètres, arguments)"> avec l'utilisation du « ' » pour délimiter les chaînes de caractères passées en paramètres ou en arguments et du style text-decoration pour définir les besoins de soulignement ou pas (voir §3.14).
3. Utilisation de la gestion des événements sur tag HTML (voir §5.2).

Ces trois techniques ne répondent pas au même besoin.

La première technique est utilisée soit :

- Pour générer un contenu à l'emplacement désiré à l'aide d'un document.write() (voir §5.12.1).
- Pour initialiser le document ou un formulaire ou un élément du formulaire ou un objet DOM. Dans ce cas, l'emplacement de la balise se situe à la fin de l'élément concerné. Attention, ce cas d'utilisation doit être limité pour privilégier l'événement onLoad de l'élément concerné dans un souci d'optimisation (éviter le parsing de la page HTML une deuxième fois).

La deuxième technique permet d'effectuer un traitement client lors du click sur un lien. Cette technique est exploitée par exemple lors de la validation d'un formulaire (voir §2.7.2.8 et §5.7.1).

La troisième technique est celle préconisée pour gérer la plupart des cas (voir §5.2 pour détail des événements). Elle est largement utilisée et décrite dans la suite de ce document.

5.4 GESTION DES OBJETS JS

5.4.1 INTERROGATION D'OBJET DOM

Pour interroger un objet spécifique d'un document HTML à titre informatif ou pour manipulation, la directive normalisée par W3C est document.getElementById("ID de l'objet").

Bien sûr, cette directive nécessite d'avoir au préalable positionné sur l'objet l'attribut ID.

Cette directive remplace les anciennes directives non normalisées propres à Microsoft document.all("ID de l'objet") ou à Netscape document.layers("ID de l'objet") en sachant que celles-ci fonctionnaient aussi sur l'attribut NAME de ces objets.

Pour récupérer une collection d'objet d'un certain type, l'utilisation de la directive document.getElementsBytagName("Type d'objet (Tag)") est préconisée par la W3C. Cette collection étant un tableau d'éléments (item), on peut isoler un de ces éléments pour manipulation spécifique (Ex : document.getElementsBytagName("Type d'objet (Tag)")[N° Item]).

A noter, l'interrogation d'un objet DOM W3C par l'utilisation de ces directives nous positionne dans l'arborescence de la modélisation DOM. Il est ainsi possible de récupérer les nœuds « fils » dans une collection avec la directive ".childNodes[]" ou le nœud « parent » avec la directive ".parentNode". A partir de la collection ".childNodes[]", il est possible d'accéder à un nœud particulier soit en précisant le N° item de la collection ou en utilisant les directives ".firstChild" ou ".lastChild" pour accéder au premier ou au dernier nœud.

Cette gestion de l'arborescence DOM W3C est renforcée lors de l'interrogation d'un tableau par l'ajout de deux autres directives ".rows[]" et ".cells[]" pour obtenir la collection des lignes d'un tableau d'une part et la collection des cellules d'une ligne d'autre part.

De plus, l'utilisation de directives du DOM level 0 compatible pour l'ensemble des navigateurs du marché depuis la version 3 peut s'avérer utile pour récupérer les collections d'objet d'un certain type (collection construite par le navigateur au chargement de la page) :

- document.images pour la collection des images du document.
- document.forms pour la collection des formulaires et document.forms[N° du formulaire].elements pour les éléments de saisie.
- document.frames pour la collection des IFRAME et FRAME présent dans un document.
- document.links pour la collection des liens présents dans un document.
- document.cookies pour la collection des cookies.

5.4.2 VERIFICATION DE L'EXISTENCE D'UN OBJET OU D'UNE PROPRIETE

Pour vérifier l'existence d'un objet avant manipulation ou pour un besoin de règle technique, il suffit de l'interroger dans une condition if.

Ainsi, il est préférable lors de la manipulation d'un objet optionnel suivant la dynamique de la page de vérifier son existence avant toute manipulation pour éviter les erreurs javascript qui s'ensuivent si manipulation d'objet inexistant.

Cette préconisation est valable aussi pour les propriétés des objets interrogés sauf que la clause if n'est pas suffisante dans ce cas.

Pour pouvoir tester l'existence d'une propriété d'un objet, il est donc nécessaire de passer par un « cast » de l'objet en objet « String » et de tester si la valeur de cet objet « String » est "undefined" lors de son inexistence.

Exemple :

```
var test=new String(objet ou variable à tester);
if (test != "undefined")
{
  //Existence de l'objet ou de la propriété testé
}
```

5.4.3 UTILISATION D'OBJET JS EN VARIABLE

Il est judicieux lors d'un besoin de variable complexe sur des données liées de type « structure » de déclarer cette variable sous forme d'objet.

Pour plus de clarté, il est préférable de définir un constructeur pour initialiser cet objet suivant le besoin. Ceci permet ainsi d'itérer le nombre d'objets nécessaires sous forme de tableau pour faciliter le parcours d'une liste d'objets lors du traitement de ces objets.

Exemple pour constituer un référentiel personne pour un annuaire :

```
//Constructeur de l'objet
function Personne(id, nom, prenom, telephone, adresse)
{
  this.id = id;
  this.nom = nom;
  this.prenom = prenom;
  this.telephone = telephone;
  this.adresse = adresse;
}
//Déclaration des variables
var TableauAnnuaire = new Array();
var i = 0;
//Itération sous forme de tableau d'objets
TableauAnnuaire[i] = new Personne(i++,"DUPONT","Pierre","01.30.27.59.69","17, avenue du général de Gaulle 75012 PARIS");
TableauAnnuaire[i] = new Personne(i++,"DUBOIS","Frank"","01.30.27.55.70","17, rue du général Leclerc 75003 PARIS");
//…
```

5.5 DETECTION DE COMPATIBILITE

5.5.1 VIS A VIS DU DOM W3C

Pour vérifier la compatibilité d'un navigateur vis à vis du DOM W3C, il suffit de tester la capacité d'interprétation du navigateur d'un certain nombre de méthodes propres au DOM W3C de la même manière que lors du test de l'existence d'un objet avant manipulation (voir §5.4.2).

Ainsi, l'appel des fonctions "document.getElementById" et "document.createElement" dans une clause "if" permet de certifier que le navigateur possède un degré de compatibilité DOM W3C suffisant pour se connecter à l'applicatif. Le choix de ces deux méthodes représentatives du DOM W3C doit s'effectuer à l'aide des préconisations des éditeurs de navigateur du marché et des besoins propres aux applicatifs. Le prototypage de cette détection de compatibilité a été conforme à nos attentes (Ex : Rejet de Netscape 4.X…).

Exemple d'implémentation pour vérifier la compatibilité des navigateurs ciblés (voir §1.3) :

```
//Redirection vers une page d'alerte de navigateur non conforme ou passage vers une
page passée en paramètre (Ex : demande d'authentification)
function valider(page)
{
  var alertpage = "/alertW3C.html";
  if ((document.getElementById) && (document.createElement))
    window.location.href=page;
  else
    window.location.href=alertpage;
}
```

5.5.2 VIS A VIS D'UNE VERSION D'UN NAVIGATEUR OU D'UN OS

Il est préférable de ne pas utiliser une détection de type ou de version de navigateur mais d'utiliser une détection de compatibilité vis à vis d'une technologie (Ex : DOM W3C) ou d'existence d'objet référencé (voir §5.4.2) pour gérer les particularités de certains navigateurs vis à vis d'une fonctionnalité donnée.

La seule détection de configuration client nécessaire est liée à l'OS (Ex : Macintosh) lors de la gestion de particularité de résolution d'un pixel, de taille des polices… Ainsi, la détection d'un client sur Mac permet de fournir au client une ergonomie adaptée en lui appliquant la bonne feuille de style lors de besoin ergonomique complexe.

L'implémentation de ce type de détection s'effectue en interrogeant l'objet référencé JS "navigator.userAgent". Cette technique de détection renforce l'inutilité de l'utiliser pour détecter une version ou un type de navigateur du fait que certains navigateurs (Ex : Opéra, Safari, Konqueror…) peuvent simuler un autre userAgent selon la configuration du navigateur désirée par le client.

Cette détection peut être utilisée aussi pour des besoins de statistique sur l'ensemble des configurations client mises en place par les utilisateurs.

Exemple d'implémentation pour utiliser une feuille de style spéciale Mac :

```
var detect = navigator.userAgent.toLowerCase();

function checkIt(string)
{
  place = detect.indexOf(string) + 1;
  thestring = string;
```

```
  return place;
}

function checkOS()
{
  var OS;
  if (checkIt('konqueror'))
  {
    browser = "Konqueror";
    OS = "Linux";
  }
  if (!OS)
  {
    if (checkIt('linux')) OS = "Linux";
    else if (checkIt('x11')) OS = "Unix";
    else if (checkIt('mac')) OS = "Mac";
    else if (checkIt('win')) OS = "Windows";
    else OS = "an unknown operating system";
  }
  return OS;
}

var test = checkOS();
if(test == "Mac")
  document.write('<LINK REL="stylesheet" TYPE="text/css"
HREF="/css/style_mac.css">');
else
  document.write('<LINK REL="stylesheet" TYPE="text/css"
HREF="/css/style_nonmac.css">');
```

Pour aider ce type d'implémentation le framework fournit par Aubay propose un ensemble de déatection de la configuration client afin de fournir des éléments de test ou des éléments de statistiques sur l'utilisateur de l'applicatif.

Cette fonction avancée du framework fournit les informations suivantes :

- OS de l'utilisateur (Windows/Mac/Os2/Unix/Vms/Autres).
- Compatibilté W3C de l'utilisateur (Oui/Non).
- Type de navigateur de l'utilisateur (Opera/Gecko/Msie/TV/NS4/Autres).
- Version du navigateur en fonction du type de navigateur.

5.5.3 VIS A VIS DES COOKIES

Pour savoir si un navigateur est capable de gérer les cookies session (voir §5.11 pour manipulation avec le navigateur), l'utilisation de la fonction référencée JS navigator.cookieEnabled est préconisée.

Cette fonction retourne un booléen indiquant si le navigateur est configuré pour accepter les cookies ou si la version du navigateur est compatible pour gérer des cookies session.

5.5.4 VIS A VIS D'UN PLUGIN

5.5.4.1 DETECTION VIS A VIS DU FLASH

La détection d'un navigateur vis à vis du Flash s'effectue par l'intermédiaire des tableaux référencés en JS "navigator.plugins" et "navigator.mimeTypes". Ces tableaux indiquent l'ensemble des plugins et mime types que le navigateur est capable de gérer.

Les navigateurs Explorer 3 et 4 sur Mac, Konqueror et Ice Browser ne supportent pas ces tableaux. Ils sont donc dans l'incapacité de détecter leur compatibilité avec le Flash.

Le tableau des plugins gérés par le navigateur apporte en plus comme information la version compatible du plugin tandis que le mime types renseigne seulement l'accessibilité du navigateur sur les objets de type "application/x-shockwave-flash".

Les navigateurs Opera sur Mac et Linux, iCab et Hotjava supportent seulement le tableau mimeTypes. De ce fait, pour ces navigateurs, il est impossible de détecter la version du Flash player.

De plus, la stratégie mise en place par MSIE est ambivalente. Ainsi, les deux tableaux indiqués ci-dessus sont toujours vide sauf pour MSIE 5.X sur Mac.

La seule manière alors de détecter le Flash sur MSIE est d'utiliser du VBScript en sachant que Mac ne gère pas le VBScript. Pour s'en sortir, il est donc nécessaire d'utiliser à la fois du JavaScript et du VBScript pour détecter la présence ou non du player Flash sur MSIE.

La détection en VBScript d'un player Flash s'effectue en tentant de créer un objet par la directive CreateObject("ShockwaveFlash.ShockwaveFlash.version").

Pour simplifier la compréhension et l'utilisation de ce type de détection, un code script est proposé pour mutualiser et normaliser son implémentation dans le framework proposé par Aubay.

Ce script renvoie une variable "flashinstalled" indiquant :

- 2 : player Flash installé.
- 1 : player Flash non installé.
- 0 : installation d'un player Flash non connue.

Ce script ne présente pas la stratégie mise en œuvre en fonction du résultat de cette détection. Cette stratégie est définie par les responsables applicatifs en fonction du périmètre d'utilisation du Flash (Ex : remplacement du flash par des gifs animés dans le cadre d'un besoin seulement ergonomique, affichage d'un message et d'un lien de téléchargement du plugin dans le cadre d'un besoin transactionnel …)

5.5.4.2 DETECTION VIS A VIS D'AUTRES PLUGINS

D'autres applications de ce script peuvent être utilisées vis à vis d'autres plugins en spécifiant d'autres mime types et d'autres objets VBScript (Remarque : ce besoin est à implémenter que si il est exprimé dans le cadre de l'applicatif).

Exemples :

Pour Adobe Acrobat en utilisant :

- MIME type : application/pdf
- VB Object : Pdf.PdfCtrl.[version]

Pour Real Player en utilisant

- MIME type : RealPlayer
- VB Object: RealPlayer.RealPlayer(tm) ActiveX Control (32-bit) pour Real5 ou rmocx.RealPlayer G2 Control pour Real G2.

5.6 GESTION DE LA NAVIGATION

5.6.1 CONSTRUCTION DES URLS

Pour faciliter la maintenance, l'évolutivité et la cohérence du code, il est nécessaire de définir l'ensemble des urls (pages, codes javascript, images, plugins…) sous leur forme relative.

Attention, cette forme relative doit s'exprimer directement à partir de la racine du document-root du serveur Web pour éviter toute utilisation de remonté d'arborescence ou de positionnement d'arborescence du style « . », « ./ », « ../ »…

Cette mise en forme permet ainsi d'avoir qu'une seule version de code sans condition if suivant l'environnement (Ex : Développent, Recette fonctionnelle, Recette d'homologation et Production) et le protocole (http/https) lors d'un utilisateur sur le site sécurisé ou non (Ex : lors de composant applicatif à la fois accessible crypté ou non).

Dans le cas où l'url spécifiée n'a pas le même host que l'applicatif, l'utilisation de ce type d'url relative n'étant pas possible, il est préférable de construire l'url absolue en passant en paramètre le host désiré ou le path de l'url. La construction de cette url s'effectuera ainsi suivant le protocole et l'environnement où l'utilisateur est en cours de navigation.

Exemple de construction d'url absolue pour l'appel d'un code javascript fournissant un contenu situé sur un back office :

```
//Détermination du Host Name de l'url du code javascript du back office suivant le
protocole et l'environnement accédé par l'utilisateur
function HostNameBackOffice()
{
  var protocol = document.location.protocol;
  var hostname = document.location.hostname;
  var hostnamebackoffice;
  if (hostname.search("Mot clé du domaine pour le back office") != -1)
    hostnamebackoffice = "";
  else if (document.location.port != "")
  {
    if (protocol == "https:")
      hostnamebackoffice = "https://www.domaine.dev.backoffice.com:1443";
    if (protocol == "http:")
      hostnamebackoffice = "http://www.domaine.dev.backoffice.com:1080";
  }
  else if (hostname.search("homo") != -1)
    hostnamebackoffice = protocol + "//www.domaine.homo.backoffice.com";
  else
    hostnamebackoffice = protocol + "// www.domaine.homo.backoffice.com";
  return hostnamebackoffice;
}
//Construction de l'url du code javascript du back office
function EcrireAppelJSBack(urlJS)
```

```
{
  var lien = HostNameBackOffice();
  var urlJSBack = lien + urlJS;
  document.write('<script type="text/javascript" src="' + urlJSBack + '"></script>');
}
```

5.6.2 APPEL D'UNE PAGE LORS DE LA SELECTION D'UN ITEM D'UNE BOITE DE SELECTION

Il peut être utile d'utiliser une boîte de sélection pour effectuer un appel d'une page dédiée à la sélection d'un des items.

Pour cela, il suffit de réévaluer l'attribut window.location sur la valeur de l'item sélectionnée (contenant l'url spécifique à cet item) lors de l'événement « onChange » de la boîte de sélection.

Exemple d'implémentation :

```
<form id="navigation" name="navigation">
<select id="select_navigation" name="select_navigation"
onChange="eval(window.location=document.navigation.select_navigation[document.navigat
ion.select_navigation.options.selectedIndex].value)">
  <option selected value="/url1.html">Choix de navigation 1</option>
  <option value="/url2.html">Choix de navigation 2</option>
  <option value="/url3.html">Choix de navigation 3</option>
  <option value="/url4.html">Choix de navigation 4</option>
  <option value="/url5.html">Choix de navigation 5</option>
  <option value="/url6.html">Choix de navigation 6</option>
  <option value="/url7.html">Choix de navigation 7</option>
  <option value="/url8.html"> hoix de navigation 8</option>
</select>
</form>
```

5.6.3 GESTION DE L'HISTORIQUE DES NAVIGATEURS (BOUTONS BACK ET FORWARD DU NAVIGATEUR)

La gestion de l'historique porte sur les objets WINDOW, FRAME et IFRAME.

Etant donné qu'il est impossible de tracer le click par un client sur le bouton BACK du navigateur, seul l'historique du navigateur (c'est à dire les dernières pages appelées par le client) est accessible. Malheureusement, cet historique est en lecture seule du fait qu'il appartient au navigateur client.

Le seul moyen pour modifier l'historique est de ne pas incorporer l'appel d'une page dans l'historique par le biais de la fonction javascript:window.location.replace('URL').

Exemple de fonctionnement pour gérer l'historique du navigateur :

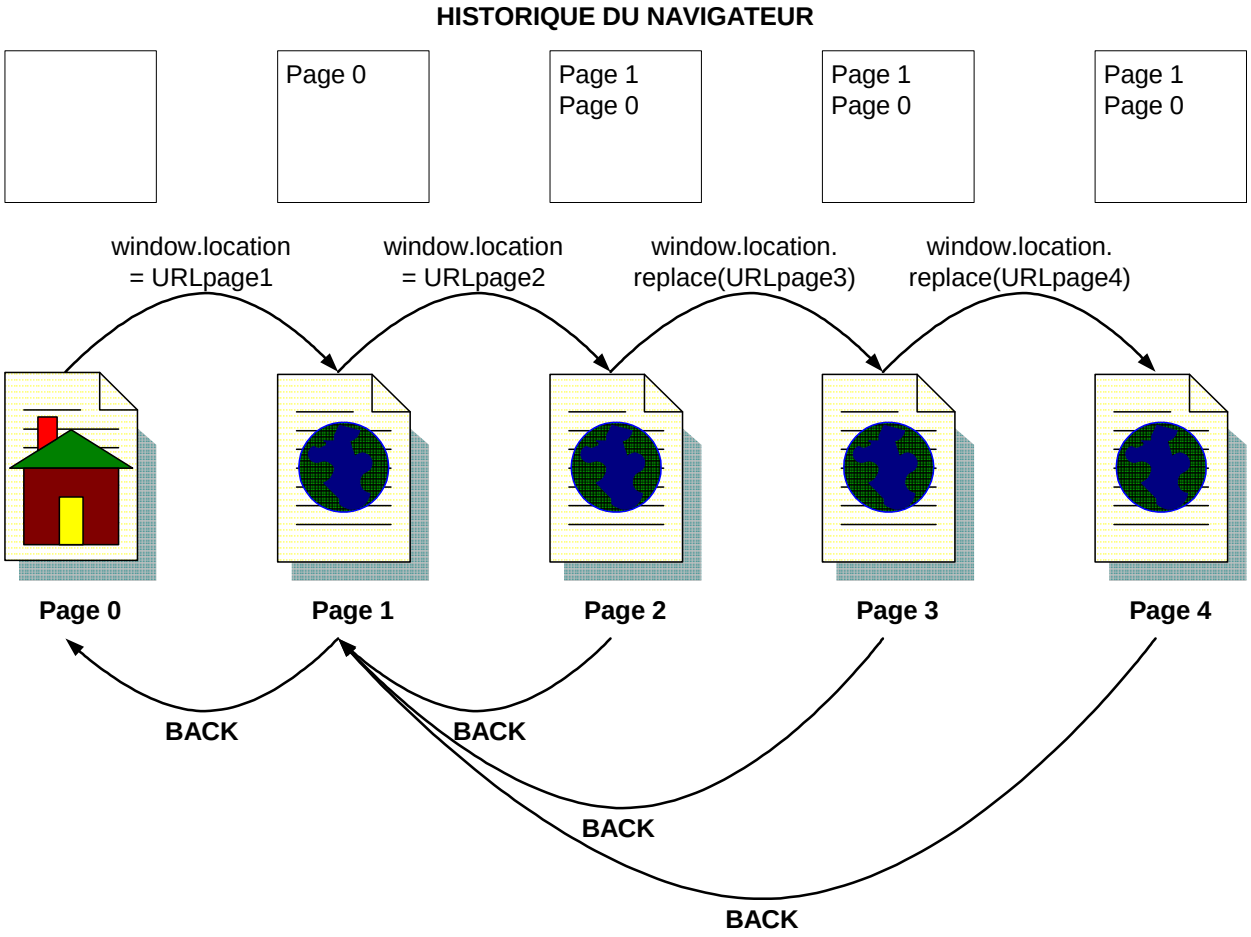


Figure 6 : gestion de l'historique du navigateur

La fonction javascript:window.location=URLpage est équivalente à l'appel de l'URLpage par un lien.

Pour pouvoir spécifier un TARGET comme dans les liens, il suffit de préciser la fenêtre sur laquelle s'applique la redirection (Exemple pour un target="_top" ou "_parent" → parent.window.location="URLpage")).

Pour contrôler la navigation du client vis à vis de l'historique et éviter un certain nombre de dysfonctionnements propres à chaque éditeur de navigateur (historique propre à chaque frame, page précédente avec formulaire mal initialisé, page précédente non souhaitée lors de l'utilisation de formulaire en méthode POST…), il est préférable d'éviter d'utiliser les fonctions liées à l'objet JS référencé history tels que history.go, history.back ou history.next en les remplaçant par des liens (pouvant être recryptés lors de la présence de données confidentielles) sur les pages souhaitées en précisant l'initialisation souhaitée du formulaire précédent.

Cette préférence reste une préconisation non une interdiction car dans certain cas l'utilisation des fonctions liées à l'objet history peut être plus simple ou nécessaire. Ainsi, par exemple, il est préférable d'utiliser des liens pour les boutons « Retour » dans des pages d'erreur et un history.back lors de l'appel par un « BACK » du client sur une page d'attente (Chargement en cours…) pour éviter d'afficher cette page.

5.7 GESTION DES FORMULAIRES

Avant de soumettre un formulaire au serveur, il faut vérifier que la saisie de l'utilisateur est correcte. Ceci nécessite donc d'appeler une fonction de validation de saisie en tenant compte de :

1. Appel de la fonction sur une action de l'utilisateur soit sur click de la souris ou sur l'appui de la touche « Entrée ».
2. Eviter d'appeler de nouveau cette fonction si elle est déjà en cours d'exécution.
3. Si la validation est OK, désactiver toutes les actions que l'utilisateur peut entreprendre pour qu'il attende le résultat de sa requête.

La gestion de la touche « Entrée » pour valider un formulaire est nécessaire du fait que certains navigateurs (MSIE Mac et Opéra) gèrent cette touche par défaut suivant la configuration du navigateur. Pour pallier à certains dysfonctionnements liés à cette gestion par défaut, l'applicatif doit implémenter la gestion de cette touche pour obtenir un comportement commun à l'ensemble des navigateurs ciblés.

L'ensemble de ces besoins est détaillé dans les paragraphes suivants.

5.7.1 VALIDATION D'UN FORMULAIRE

L'utilisateur peut lancer la validation d'un formulaire soit par un click de la souris ou soit par l'utilisation de la touche « Entrée ». Nous allons donc étudier le comportement d'un click de la souris sur différents objets d'une page et ensuite l'utilisation de la touche « Entrée ».

Pour faire les tests suivants, nous avons associé à chaque événement, une fonction du type :

```
OBJET.onclick = function (){alert(this.type+".onclick"); return false;};
```

Etudes de comportement navigateur :

Les différents objets DOM testés sont :

- Link :
- Boutton : <input type="button">
- Submit : <input type="submit">
- Image : <input type="image">

Objet DOM	IE5	NS4.x	NS7	Opera7
Link	Link onClick	Link onClick	Link onClick	Link onClick
Button	Button onClick	Button onClick	Button onClick	Button onClick
Submit	Submit onClick	Submit onClick	Submit onClick	Submit onClick
Image	Form onSubmit	Form onSubmit	Form onSubmit **(2 fois)**	**Image onClick**

Tableau 36 : événements sur un click de l'utilisateur

Résultats conformes à nos attentes sauf pour :

- Un click sur un INPUT de type 'image' peut provoquer deux événements dans NS7,
- Dans Opera, si on ne capte pas l'événement Image.onClick, on obtient aussi Form.onSubmit.

Les formulaires testés du fait de leur différence de comportement en fonction de l'action utilisateur sont :

- Formulaire avec élément de saisie texte : <input type="text">
- Formulaire avec élément de saisie case à cocher : <input type="checkbox">
- Formulaire mixte.

Form avec :	IE5	NS4.x	NS7	Opera7
elem txt + submit	elem.onKeyDown form.onKeyDown document.onKeyDown	elem.onKeyDown	**submit.onClick** elem.onKeyDown form.onKeyDown document.onKeyDown	elem.onKeyDown form.onKeyDown document.onKeyDown **submit.onClick**
elem txt + img	elem.onKeyDown form.onKeyDown document.onKeyDown	elem.onKeyDown	elem.onKeyDown form.onKeyDown document.onKeyDown	elem.onKeyDown form.onKeyDown document.onKeyDown **form.onSubmit**
2 elem txt	elem.onKeyDown form.onKeyDown document.onKeyDown	elem.onKeyDown	elem.onKeyDown form.onKeyDown document.onKeyDown	elem.onKeyDown form.onKeyDown document.onKeyDown **form.onSubmit**
1 elem txt	elem.onKeyDown form.onKeyDown document.onKeyDown	elem.onKeyDown	**form.onSubmit** elem.onKeyDown form.onKeyDown document.onKeyDown	elem.onKeyDown form.onKeyDown document.onKeyDown **form.onSubmit**

Tableau 37 : événements sur l'utilisation de la touche « Entrée » dans un élément Text du formulaire

Form avec :	IE5	NS4.x	NS7	Opera7
elem chk box + submit	elem.onKeyDown form.onKeyDown document.onKeyDown	(checked / unchecked) + elem onKeyDown sur MAC	**form.onSubmit** elem.onKeyDown form.onKeyDown document.onKeyDown	elem.onKeyDown form.onKeyDown document.onKeyDown
elem chk box + img	elem.onKeyDown form.onKeyDown document.onKeyDown	(checked / unchecked) + elem onKeyDown sur MAC	**form.onSubmit** elem.onKeyDown form.onKeyDown document.onKeyDown	elem.onKeyDown form.onKeyDown document.onKeyDown
elem chk box + elem txt	elem.onKeyDown form.onKeyDown document.onKeyDown	(checked / unchecked) + elem onKeyDown sur MAC	**form.onSubmit** elem.onKeyDown form.onKeyDown document.onKeyDown	elem.onKeyDown form.onKeyDown document.onKeyDown

Tableau 38 : événements sur l'utilisation de la touche « Entrée » dans un élément Check Box du formulaire

Résultats conformes à nos attentes sauf pour :

- Les événements submit.onClick et form.onSubmit sont différents suivant le navigateur et la composition des éléments d'un formulaire.
- Sous NS4.X, l'événement onKeyDown n'est pas propagé d'un élément au formulaire, puis au document, par contre le code suivant permet de capter l'événement par le document :

```
if(navigator.appName.indexOf("Netscape") >= 0) document.captureEvents(Event.KEYDOWN);
```

Analyses :

- Un click sur un INPUT de type 'image' n'est pas identique suivant le type de navigateur.
- La saisie de la touche 'Entrée' n'est pas identique suivant le type de navigateur et si le formulaire contient un objet SUBMIT ou un seul objet TEXT.
- La saisie de la touche 'Entrée' n'est pas identique dans un objet CHECKBOX suivant le type de navigateur.
- La saisie de la touche 'Entrée' n'est pas identique dans un objet TEXT suivant le type de navigateur.

Pour avoir un fonctionnement homogène sur tous les types de navigateur et éviter d'avoir 2 événements simultanés pour une seule action entraînant une double validation du formulaire, il est nécessaire de :

- Désactiver l'événement "form.onSubmit" dans les formulaires.
- Désactiver les événements elem.onKeyDown et form.onKeyDown pour avoir seulement un document.onKeyDown.

De plus, conformément à la norme de réalisation HTML sur la gestion des formulaires, il est préférable d'utiliser un lien sur une image pour appeler la fonction de validation lors d'un click de l'utilisateur (voir §2.7.2.8 pour explication). Ceci nous affranchit du besoin de gérer les particularités liées au click de la souris sur les éléments INPUT de type "image" et "submit" dans les formulaires.

Palliatif :

Pour pallier au problème décrit ci-dessus, il est nécessaire de désactiver l'événement "onSubmit" de tous les formulaires d'un document et initialiser la fonction de validation pour la saisie de la touche « Entrée ».

Pour cela, il faut appeler une fonction présente dans le framework JS technique placée juste avant la fin du code HTML pour l'exécuter après le chargement des formulaires et avant le chargement des images (Voir §5.7.3 pour explication) :

```
<script type="text/javascript">
  initForm(valider);
</script>
</body>
</html>
```

A l'aide de cette initialisation des événements, nous obtenons le comportement sssuivant pour tous les types de navigateur :

- Un click sur un élément LINK provoque un link.onClick.
- Un KeyDown sur un élément du document provoque un document.onKeyDown.

Solution d'implémentation de la validation par un click :

A l'aide du framework JS technique d'Aubay, l'appel de la fonction de validation gérée par un click sur le formulaire doit être implémenté de la manière suivante :

- Pour un formulaire en méthode POST :

Code JavaScript :

```
// submit de l'Action
// fonction pour valider un formulaire
function valider()
{
  // contrôles de surface renvoyant OK ou NOK
  if(OK) document.forms[0].submit();
}
```

Code HTML d'appel sur une image :

```
<a href="javascript:valider()"><img src="" width="" height=""></a>
```

- Pour un formulaire en méthode GET :

Code JavaScript :

```
// submit de l'Action
// fonction pour valider un formulaire
function valider(url)
{
    // contrôles de surface renvoyant OK ou NOK
    // construction de l'url GET en passant la saisie de l'utilisateur
    url += document.form[0].elements[0].value;       //Première valeur saisie
    //…                                              //Autres valeurs saisies
    if(OK) document.location.replace(url);
}
```

Code HTML d'appel sur une image :

```
<a href="javascript:valider('url')"><img src="" width="" height=""></a>
```

Solution d'implémentation de la validation par l'utilisation de la touche « Entrée » :

La gestion de la touche « Entrée » s'effectue par le code JavaScript suivant (Voir §5.7.3 pour explication) :

```
function fctOnKeyDown(e)
{
```

```
  var key = e ? (e.which ? e.which : e.keyCode) : event.keyCode;
  var elm = e ? e.target : event.srcElement;
  if(key == 13 && elm.form && elm.type!="button")
  return document.onkeydown13(elm);
}
```

5.7.2 GESTION D'APPEL MULTIPLE POUR LA VALIDATION D'UN FORMULAIRE

Si un nouvel appel à la fonction de validation est provoqué par une action de l'utilisateur (Ex : double-click, touche « Entrée » utilisée plusieurs fois) alors que la fonction de validation est déjà en cours d'exécution, il faut annuler cette dernière action.

Pour cela, l'utilisation d'une variable booléenne, nous permet de pallier à cette problématique en implémentant le code suivant :

```
// initialisation
document.onverif = false;

// fonction pour valider un formulaire
function valider()
{
  if(!document.onverif)
  {
    document.onverif = true;
    // contrôles de saisie
    document.onverif = false;
  }
}
```

5.7.3 DESACTIVATION DES ACTIONS UTILISATEUR EN ATTENTE DE L'AFFICHAGE DE LA PAGE SUIVANTE

Nous avons créé une fonction au sein du framework JS technique d'Aubay pour désactiver l'ensemble des actions de clicks sur LINK, BOUTON et la saisie de la touche « Entrée ». Il faut l'exécuter lorsque les tests de validation (contrôles de surface…) sont OK, pour obliger l'attente de l'affichage de la page suivante liée à la requête soumise par le formulaire.

Exemple de fonction de validation :

```
function valider()
{
  // contrôles de surface renvoyant OK ou NOK
  if(OK) disableAllAction();
}
```

5.7.4 CONTROLES DE SURFACE POUR VALIDER UN FORMULAIRE

Pour valider un formulaire, il faut vérifier si la saisie de l'utilisateur est correcte. Pour faciliter cette validation de la saisie, Aubay a développé un framework JS technique proposant un ensemble de contrôle de format de saisie standard.

Ce framework repose au maximum sur les objets intégrés du JavaScript pour une meilleure performance et un code source simplifié. Nous utiliserons ainsi les expressions régulières pour contrôler les formats, l'objet Date pour les dates, Integer pour les entiers et Float pour les montants.

Ce framework permet d'appliquer les règles suivantes par défaut pour les contrôles de saisie :

- Tous les contrôles se font après validation par le client de sa saisie et jamais après la perte du focus d'un élément du formulaire.
- Après affichage du message, le focus est mis sur l'élément du formulaire incriminé.
- Pour les éléments d'un formulaire <input type="text"> de type code, alpha, alphanumérique et texte, aucun reformatage n'est effectué après contrôle.
- Pour les éléments d'un formulaire <input type="text"> de type montant et entier, avant le contrôle, on supprime le + en 1er digit, les 0 avant et après et on remplace le point par une virgule si besoin est. Puis on effectue le contrôle.
- Pour les éléments d'un formulaire <input type="text"> de type date, on reformate si possible sous la forme JJ/MM/AAAA avant contrôle puis on teste la cohérence.

Il est possible à l'aide du framework de modifier ces règles par défaut ou nd'en ajouter de nouvelles suivant les besoins exprimés par la maîtrise d'ouvrage.

De plus, un certain nombre de contraintes techniques doivent être appliquées :

- Il faut utiliser les fonctions du framework pour développer les contrôles de surface plus complexes liées à des règles de gestion banquaire.
- On doit référencer l'ensemble des messages d'erreur liés aux contrôles de surface par le biais d'initialisation de variable.
- Le code des contrôles de surface doit être statique et non généré par le serveur Web. Si besoin de paramétrage fourni par le serveur, passer des paramètres dans l'appel du contrôle de surface ou par le biais d'initialisation de variable.
- Tout contrôle de surface doit être dupliqué sur le serveur pour des raisons de sécurité sous forme de règles de gestion ou de règles techniques.

L'ensemble des fonctionnalités disponibles dans le framework JS Aubay sont nombreuses et permettent de réduire le temps de développement et les risques d'erreur ou d'incompatibilité vis à vis de certain navigateur:

- Fonctions génériques de contrôle de format permettant de vérifier le format d'une saisie :
  - Tranformation d'une chaîne de caractère en entier et vis versa.
  - Tranformation d'une chaîne de caractère en montant et vis versa.
  - Tranformation d'une chaîne de caractère en date et vis versa.
- Fonction générique de contrôle de format permettant de vérifier le format d'une saisie :
  - Format simple pour une chaîne de caractères vis à vis d'une expression régulière.
  - Entier borné ou non (Min et/ou Max).
  - Numéro borné ou non (Min et/ou Max).
  - Montant borné ou non (Min et/ou Max) avec définition du séparateur (« , » ou « . »).
  - Date suivant un format donné et bornée ou non (Min et/ou Max).
  - Autres…
- Fonction générique de contrôle de saisie faisant appel à la fonction de contrôle de format et pouvant vérifier son caractère obligatoire. Si une condition n'est pas vérifiée, un message d'alert paramétré par le développeur est affiché à l'utilisateur.

5.8 GESTION DES EFFETS DYNAMIQUES

5.8.1 CHANGEMENT D'IMAGES

Avant tout changement dynamique d'images, il est nécessaire de pré-charger les images faisant l'objet de manipulation.

Ce pré-chargement s'effectue par le biais d'un tableau javascript d'objet « image » propre au document HTML. Ce tableau est construit lors du chargement de la page en passant en paramètre les paths des images nécessaires. Il permet ainsi d'éviter les appels aux images lors du dynamisme et les désagréments aléatoires d'affichage qui peuvent résulter (Absence d'image, croix rouge de chargement, effets de clignotement…)

Exemple d'implémentation du pré-chargement d'images manipulées par la suite :

```
<head>
<script type="text/javascript">
function MM_preloadImages()
{
  var d=document;
  if(d.images)
  {
    if(!d.MM_p)
      d.MM_p=new Array();
    var i,j=d.MM_p.length,a=MM_preloadImages.arguments;
    for(i=0; i<a.length; i++)
    {
      if (a[i].indexOf("#")!=0)
      {
        d.MM_p[j]=new Image;
        d.MM_p[j++].src=a[i];
      }
    }
  }
}
function init_page()
{
  MM_preloadImages('/images/image1.gif' ,'/images/image2.gif' ,'/images/image3.gif');
}
</script>
</head>
<body onLoad="init_page()">
```

La manipulation de changement d'images consiste à réaffecter l'attribut src par le lien de l'image désirée sur l'emplacement image correspondant. Cette réaffectation s'effectue par la commande document.images["Name de l'image"] sur l'image spécifiée par l'attribut Name. La mise en valeur de l'attribut Name est contraire à la préconisation du §5.4.1, il est donc préférable de doubler cette valeur sur l'attribut ID pour pouvoir effectuer d'autres manipulations (Affichage/Désaffichage, redimensionnement, effet de style…).

De plus, lors de manipulation dynamique multiple sur le document pour la même action client, il peut arriver que le séquencement de réaffichage de la page s'effectue mal (Ex : affichage d'une image vide). Pour pallier ce

dysfonctionnement non aléatoire sur certains navigateurs, il est conseillé de forcer le séquencement du réaffichage de la page liée à une manipulation en temporisant ces actions. Cette temporisation s'effectue par le biais de la commande « setTimeout('Nom de la fonction de manipulation', Délai en milisecondes) » ou un délai d'1 miliseconde suffit pour forcer le séquencement.

5.8.2 AFFICHAGE/DESAFFICHAGE D'UNE ZONE DYNAMIQUE SUIVANT UNE ACTION UTILISATEUR

Pour afficher/désafficher un objet d'un document, deux méthodes peuvent être mises en œuvre selon le DOM W3C :

1. Attribut "visibility" de l'objet renseigné à "show" (affichage) ou à "hidden" (désaffichage).
2. Style "display" de l'objet renseigné à "" (affichage) ou à "none" (désaffichage).

La différence se situe dans l'effet dynamique souhaité. La méthode 1. permet seulement de afficher/désafficher un objet du document sans réafficher le contenu de la page en tenant compte de l'absence de cet objet.

En revanche, la méthode 2 réaffiche le contenu de la page en décalant les autres objets DOM suivant l'état du style display.

Exemple :

Visibility

Voici une zone que l'on affiche/désaffiche :
Dfdqsfdfjkqhfdjqfhjqhfjsqdhfjqhfdjqhdlfkjhqjsdfhql
Dqfqfdljkqshdfjqhfljqhdjfhqdjlfhq
Qfdjhfqjdqfhfljhqfjflhdqljhfjlqfhdjfqhdfjhlqjdlfhfqdjhfjfqdlhq
qdfkjqflhjhfjlfqhfqljfhqjlfqhfqljhfqlfqh
Voici une deuxième zone non manipulée :
Fqdsjfhqjdlfhqdljfjqfdhflkqfdhjlfdqhlqdfhj
Qdfjqdfkfqfjqkqlsfmjqkfdjfkqjfqdmfjqmkqfj
qdfqfjkhdqjdhfjqfdhfljqfdhlqjkfhfjlfqhjfqlhfdqljfhdljqfhdj

Display

Voici une zone que l'on affiche/désaffiche :
Dfdqsfdfjkqhfdjqfhjqhfjsqdhfjqhfdjqhdlfkjhqjsdfhql
Dqfqfdljkqshdfjqhfljqhdjfhqdjlfhq
Qfdjhfqjdqfhfljhqfjflhdqljhfjlqfhdjfqhdfjhlqjdlfhfqdjhfjfqdlhq
qdfkjqflhjhfjlfqhfqljfhqjlfqhfqljhfqlfqh
Voici une deuxième zone non manipulée :
Fqdsjfhqjdlfhqdljfjqfdhflkqfdhjlfdqhlqdfhj
Qdfjqdfkfqfjqkqlsfmjqkfdjfkqjfqdmfjqmkqfj
qdfqfjkhdqjdhfjqfdhfljqfdhlqjkfhfjlfqhjfqlhfdqljfhdljqfhdj

Résultat après affichage/désaffichage :

Visibility

Voici une zone que l'on affiche/Désaffiche :

Voici une deuxième zone non manipulée :
Fqdsjfhqjdlfhqdljfjqfdhflkqfdhjlfdqhlqdfhj
Qdfjqdfkfqfjqkqlsfmjqkfdjfkqjfqdmfjqmkqfj
qdfqfjkhdqjdhfjqfdhfljqfdhlqjkfhfjlfqhjfqlhfdqljfhdljqfhdj

Display

Voici une zone que l'on affiche/Désaffiche :
Voici une deuxième zone non manipulée :
Fqdsjfhqjdlfhqdljfjqfdhflkqfdhjlfdqhlqdfhj
Qdfjqdfkfqfjqkqlsfmjqkfdjfkqjfqdmfjqmkqfj
qdfqfjkhdqjdhfjqfdhfljqfdhlqjkfhfjlfqhjfqlhfdqljfhdljqfhdj

Exemple d'implémentation pour les modules implosables/explosables en utilisant une image de type + et - :

```
function AffichageDesaffichage()
{
  if document.getElementById("Module").style.display == "none")
  {
```

```
    document.getElementById("Module").style.display = "";
    document.getElementById("ExplosionImplosion").src ="/images/moins.gif";
  } else {
    document.getElementById("Module").style.display = "none";
    document.getElementById("ExplosionImplosion").src ="/images/plus.gif";
  }
}
```

5.8.3 MODIFICATION DU CONTENU D'UNE ZONE DYNAMIQUE SUIVANT UNE ACTION UTILISATEUR

Trois principes peuvent être utilisés pour modifier le contenu d'une zone suivant les actions du client :

1. Affichage/Désaffichage de zones dynamiques dont le contenu est déjà pré-formaté et initialisé. (Voir §5.8.2 pour méthode utilisée)
2. Modification du contenu d'une zone dynamique à l'aide de la méthode javascript "innerHTML".
3. Modification du contenu d'une zone dynamqique à l'aide du DOM HTML.

Le premier principe consiste à construire dynamiquement sur le serveur l'ensemble des zones dynamiques (tag DIV) possibles à l'affichage résultant de la page. Ainsi, suivant les actions du client, l'effet dynamique consiste à désafficher les zones dont l'information contenue ne correspond plus à l'action du client et à afficher celles qui correspondent.

Le deuxième et troisième principe consistent eux à construire une seule zone dynamique dans la page et de modifier le contenu de celle-ci par l'information correspondant à l'action du client générée par le serveur sous forme de variable javascript (voir §5.4.3 pour formalisme).

La différenciation entre le second principe et le troième principe se situe dans l'application du DOM Level 0 ou du DOM HTML Level 1. L'application du DOM HTML Level 1 est plus complexe et moins performante voir Annexe J : Comparatif innerHTML vs DOM W3C, elle peut s'avérer utile lors de la mise en place de principe de client riche plutôt que dans le cas de client léger (voir normes liés au client riche).

Ces tois principes ont donc pour résultat le même comportement à l'affichage. Mais, il est préférable d'utiliser le principe 1 lorsque l'information contenue par les zones dynamiques est simple et que le nombre de cas possibles est limité et connu à l'avance. A contrario les deux autres principes s'avèrent plus simples à mettre en œuvre lorsque l'information contenue est complexe en taille et formatage et quand le nombre de cas possibles est grand ou inconnu par avance.

Exemple d'implémentation du principe 2 pour l'affichage de l'adresse d'une personne sélectionnée dans un tableau d'annuaire :

```
// Voir §5.4.3 pour génération serveur de l'ensemble des informations liées à
l'annuaire sous forme tableau d'objets javascript.
function ConstruireInfo(NumLigneTableau)
{
  var TexteAdresse = '<table border="0" cellspacing="0" cellpadding="0">';
  TexteAdresse += '<tr>';
  TexteAdresse += '<td>' + TableauAnnuaire[NumLigneTableau].adresse + '</td>';
  TexteAdresse += '</tr>';
  TexteAdresse += '</table>';
  return TexteAdresse;
}
```

```
function AfficheAdresse(NumLigneSelectionne, elDetailAdresse)
{
  var DetailAdresse = ConstruireAdresse(NumLigneSelectionne);
  elDetailAdresse.innerHTML = DetailAdresse;
  elDetailAdresse.style.display = "";
}
```

Code HTML associé à chaque ligne de tableau :

```
<IMG ALT="Adresse, cliquez-ici" WIDTH="22" HEIGHT="16" BORDER="0"
SRC="/images/select.gif" onClick="AfficheAdresse(30,
document.getElementById('DivAdresse'))">
```

5.8.4 AFFICHAGE D'UNE INFORMATION DANS LA BARRE DE STATUT DU NAVIGATEUR

Il est possible d'afficher une information dans la barre de statut du navigateur à l'aide de l'attribut status de l'objet référencé window.

Mais, l'affichage de cette information dans cette barre pour un message important risque de n'être pas vu par l'utilisateur. Pour mettre en valeur cette information, il est donc nécessaire de faire défiler cette information dans la barre de status en utilisant un timer.

De plus, d'autres affichages dans la barre de status par exemple le lien lorsque le pointer de souris passe dessus peut gêner le bon déroulement de cette mise en valeur et la persistance de l'information pour le temps désiré.

Pour pouvoir mettre en œuvre cette fonctionnalité, il faut donc penser à :

1. Initialiser un temps de pause entre chaque défilement et un temps maximum d'affichage.
2. Stocker le message d'origine à afficher dans une variable et générer chaque nouvel affichage à l'aide du message d'origine et un compteur de défilement.
3. Pour ne pas gêner d'autre affichage dans la barre de statut, vérifier si le contenu est équivalent à l'ancien message généré ou s'il est vide avant de remplacer le contenu de la barre de statut par un nouveau message.
4. A la fin d'un affichage, il faut toujours vider le contenu de la barre de statut.

Une fonction avancée permettant d'initier cet affichage est proposée dans le framework JS technique d'Aubay.

5.8.5 MODIFICATION DU STYLE D'UN ELEMENT DU DOCUMENT

Pour des besoins de mise en valeur suivant une action client, il peut être nécessaire de modifier le style d'un objet du document. Pour cela, il suffit d'interroger le style de cet objet et d'effectuer une réattribution de sa valeur en dynamique soit en lui réaffectant une class (attribut className) soit une valeur sur l'attribut du style (comme pour le display du §5.8.2).

Exemple :

```
document.getElementById("ID de l'objet DOM").className = "Class du CSS à attribuer";
```

Il est préférable d'utiliser cette technique pour une mise en valeur liée à une action utilisateur. Par contre, ce principe est à éviter lors de la gestion de l'affichage lors du chargement de la page. Dans ce cas, déporter ce traitement lors de la génération de la page sur le serveur est préférable.

De plus, il est aussi possible selon DOM W3C d'ajouter ou de modifier la feuille de style ou les styles associés à un document dans un stylesheet à l'aide de la collection rules et des méthodes addRule ou removeRule. Mais, si ces modifications peuvent être utiles, elles ne sont pas encore compatibles avec l'ensemble des navigateurs du marché (Ex : IE 5.X, Netscape 7.0 et Opéra 7.0 non compatibles) et doivent donc être implémentées en connaissance de cause et avec attention.

5.9 GESTION DE FENETRES

5.9.1 GESTION DU MULTI-FENETRAGE (FRAME ET IFRAME)

Le multi-fenêtrage consiste à découper la fenêtre du navigateur en plusieurs cadres par l'intermédiaire de frames ou de l'insertion d'un cadre dans une page par l'intermédiaire d'une iFrame.

La gestion du multi-fenêtrage complexifie la gestion dynamique des pages lorsqu'un événement :

- d'une frame impacte le comportement d'une autre frame.
- d'une page impacte le comportement d'une iFrame fille.
- d'une iFrame impacte le comportement de la page parent.

Cette gestion peut être encore plus complexe lorsqu'un événement d'une frame impacte une iFrame d'une autre frame ou vice versa. Ceci est à proscrire pour limiter la complexité de la maintenance.

Pour le dialogue entre frames, il est préférable de stocker, sur la page parent définissant le FRAMESET, l'ensemble des variables utilisées par les frames et les fonctions JS effectuant la manipulation sur l'une suite à un événement déclenché par l'autre.

Il est donc préconisé de :

- Effectuer l'appel à la fonction de manipulation d'une frame par une autre en préfixant cet appel par "parent."
- Réaliser l'interrogation de l'objet DOM manipulé d'une frame par une autre par la page parent à l'aide du préfixe "parent.frames[N° de la frame]". Il est nécessaire d'utiliser le N° de la frame plutôt que l'id de la frame pour obtenir une compatibilité sur l'ensemble des navigateurs.

Lors de la gestion du dialogue entre une iFrame et sa page parent, l'utilisation de ce principe technique est identique. L'ensemble des variables et des fonctions JS communes au dialogue sera stocké sur la page contenant l'iFrame. Le préfixe "parent." est implémenté par l'iFrame pour tout événement déclencheur répercuté sur le document parent et le préfixe "parent.frames[N° de la frame].window.frames[N° de la iFrame]" est implémenté pour toute manipulation du document parent sur les objets DOM de l'iFrame (voir §5.10 pour exemple d'implémentation).

5.9.2 CREATION D'UNE FENETRE FILLE

La création d'une nouvelle fenêtre fille s'effectue par la méthode javascript window.open(URL, Nom de la fenêtre, Options sur cette fenêtre).

Les options possibles sont :

Option	Description
alwaysLowered	(JavaScript 1.2) If yes, creates a new window that floats below other windows, whether it is active or not. This is a secure feature and must be set in signed scripts

alwaysRaised	(JavaScript 1.2) If yes, creates a new window that floats on top of other windows, whether it is active or not. This is a secure feature and must be set in signed scripts.
dependent	(JavaScript 1.2) If yes, creates a new window as a child of the current window. A dependent window closes when its parent window closes. On Windows platforms, a dependent window does not show on the task bar.
directories	If yes, creates the standard browser directory buttons, such as What's New and What's Cool.
height	(JavaScript 1.0 and 1.1) Specifies the height of the window in pixels.
hotkeys	(JavaScript 1.2) If no (or 0), disables most hotkeys in a new window that has no menu bar. The security and quit hotkeys remain enabled.
innerHeight	(JavaScript 1.2) Specifies the height, in pixels, of the window's content area. To create a window smaller than 100 x 100 pixels, set this feature in a signed script. This feature replaces height, which remains for backwards compatibility.
innerWidth	(JavaScript 1.2) Specifies the width, in pixels, of the window's content area. To create a window smaller than 100 x 100 pixels, set this feature in a signed script. This feature replaces width, which remains for backwards compatibility.
location	If yes, creates a Location entry field.
menubar	If yes, creates the menu at the top of the window.
resizable	If yes, allows a user to resize the window.
screenX	(JavaScript 1.2) Specifies the distance the new window is placed from the left side of the screen. To place a window offscreen, set this feature in a signed scripts
screenY	(JavaScript 1.2) Specifies the distance the new window is placed from the top of the screen. To place a window offscreen, set this feature in a signed scripts.
scrollbars	If yes, creates horizontal and vertical scrollbars when the Document grows larger than the window dimensions.
status	If yes, creates the status bar at the bottom of the window.
titlebar	(JavaScript 1.2) If yes, creates a window with a title bar. To set the titlebar to no, set this feature in a signed script.
toolbar	If yes, creates the standard browser toolbar, with buttons such as Back and Forward.
width	(JavaScript 1.0 and 1.1) Specifies the width of the window in pixels.
z-lock	(JavaScript 1.2) If yes, creates a new window that does not rise above other windows when activated. This is a secure feature and must be set in signed scripts.

Tableau 39 : options possibles lors de la création d'une fenêtre fille

Les options de ce tableau grisées indiquent celles qui peuvent poser des problèmes de compatibilité suivant les environnements OS et les versions de navigateur. Il est préférable de se contenter d'utiliser seulement les autres.

De plus, cette commande window.open crée une fenêtre quand celle-ci n'existe pas et se contente de mettre à jour le contenu de cette fenêtre par l'URL spécifiée quand celle-ci existe déjà. Ainsi, quand l'utilisateur icônise cette fenêtre la mise à jour n'est pas visible car celle-ci n'est pas rappelée.

Deux principes techniques permettent de rappeler la fenêtre lors d'une mise à jour de celle-ci :

1. Fermeture quand elle existe et recréation d'une nouvelle fenêtre avec l'url spécifiée.
2. Demande de mise à jour du contenu de la fenêtre existante avec l'url spécifiée puis passage du focus sur celle-ci pour la rendre visible par la méthode window.focus().

Le deuxième principe a pour inconvénient d'une part de n'être pas géré avec Opéra 7.0 dans l'état de notre étude et d'autre part la fenêtre avec l'ancien contenu surgit avant de voir la mise à jour de son contenu avec l'url spécifiée entraînant parfois une incompréhension de l'utilisateur.

Il est donc préférable d'utiliser le premier principe en sachant que la fermeture sur Mac n'est pas envisageable du fait d'une mauvaise gestion de son pointeur de fenêtre sur cet OS lors d'une nouvelle création. Cet inconvénient sur Mac est sans conséquence du fait que cet OS fournit le focus par défaut à chaque mise à jour du contenu d'une fenêtre. De plus, pour éviter des problèmes de séquencement de commande de fermeture et création sur certain navigateur (Ex : Opera 7.0), il est préférable d'effectuer une temporisation de 1 milliseconde pour éviter tout problème de parallélisme sur la recréation d'une fenêtre après fermeture.

Pour pallier à l'ensemble de ces problèmatiques, Aubay fournit deux fonctions avancées dans le framework JS technique pour ouvrir une nouvelle fenêtre :

- OpenWindow pour créér une nouvelle fenêtre sans s'occuper du surgissement de celle-ci.
- OpenWindowFocus pour créér une nouvelle fenêtre avec surgissement obligatoire.

5.9.3 IMPRESSION

Pour lancer la boîte de dialogue d'impression propre à chaque navigateur, il suffit d'utiliser la méthode javascript window.print(). Malheureusement, cette méthode n'est pas compatible sous les navigateurs Mac et Opéra.

Pour cette raison, le framework JS technique d'Aubay offre une fonction avancée pour alerter les utilisateurs non compatibles en affichant un message conseillant l'utilisation du menu du navigateur pour imprimer le contenu de la fenêtre sélectionnée. Son implémentation suit le principe de détection de ces utilisateurs par leur non-compatibilité sur cette fonction plutôt que par détection de la version d'OS ou du navigateur.

5.9.4 FERMETURE

La fermeture d'une fenêtre s'effectue par la méthode window.close().

5.10 GESTION PARTICULIERE POUR LES IFRAME

5.10.1 AFFICHAGE DU RESULTAT D'UNE RECHERCHE SELON CRITERES PRE-SAISIS

Le premier cas d'utilisation d'une iFrame est l'affichage du résultat d'une recherche à partir d'un formulaire de saisie. Cette iFrame permet donc d'afficher le résultat de la recherche sur le même écran que la saisie de ses critères et offrir ainsi à l'utilisateur la possibilité d'effectuer une autre recherche sans revenir sur l'écran d'un formulaire en ayant perdu sa saisie.

Lors de la validation d'un formulaire (voir §5.7), le contenu de l'iFrame doit donc être mis à jour. Ensuite, pour afficher le résultat de la recherche, l'iFrame doit être affichée et être redimensionnée en hauteur en fonction de la taille de son contenu pour paramétrer si l'iFrame est scrollable ou non.

Lors du chargement de la page, l'iFrame est vide du fait qu'aucun critère n'a encore été saisi par l'utilisateur. Mais, pour éviter un message d'alerte de présence d'éléments sécurisés et non sécurisés sur IE, il est nécessaire de préciser sur l'attribut SRC de l'iFrame une page vide en https (bug référencé par le support Microsoft).

La mise à jour du contenu de l'iFrame s'effectue par l'appel d'une page dont les champs de l'url sont construits à partir des critères saisis sur le formulaire. L'appel de cette url par l'iFrame s'implémente soit :

1. par la réaffectation de l'attribut "src" lié à l'objet DOM iFrame avec pour inconvénient sur certains navigateurs (Ex : IE) de l'intégrer dans l'historique, entraînant à chaque back du navigateur le parcours de l'ensemble des recherches effectuées par l'utilisateur.
2. par la réaffectation de l'attribut de l'objet référencé JS "window.location" lié à l'iFrame (voir §5.9.1 pour l'interroger) en utilisant la méthode référencée JS ".replace(url)" pour éviter toute entrée dans l'historique (voir §5.6.3). Ce principe a pour inconvénient que lorsque l'iFrame est déclarée en no display par son style (non affichée lors du chargement de la page), certains navigateurs (Netscape 7.0 et Opéra 7.0) ne peuvent interroger l'iFrame par un document.frames[] pour réaffecter sa location.

De ce fait, un compromis entre les deux principes techniques cités ci-dessus est nécessaire. A l'aide de la vérification de l'existence du document.frames[0] selon la méthode explicitée dans le §5.4.2, l'exécution de la réaffectation du "src" ou du "window.location" sera pré-conditionnée. Ce compromis n'a aucune incidence sur la gestion de l'historique pour les navigateurs ne pouvant exécuter la réaffectation par un ".replace(url)" car ceux-ci lors de notre étude ne gèrent pas d'historique pour la navigation des iFrames.

L'affichage du résultat de la recherche s'effectue par l'affichage de l'iFrame selon la méthode explicitée dans le §5.8.2 avant la mise à jour de son contenu pour éviter les problèmes liés aux navigateurs ayant des difficultés de gestion sur les iFrames déclarées en no display.

Le redimensionnement de l'iFrame en fonction de son contenu s'implémente par la réaffectation de sa hauteur (attribut "height") en fonction du nombre de lignes présentes dans son contenu et nombre maximum de lignes affichables pour gérer une scroll-bar sur l'iFrame si le nombre de lignes dépasse une taille limite affichable. Le passage de ces paramètres et l'appel du redimensionnement de l'iFrame s'effectuent lors du chargement de la page de contenu de l'iFrame.

Pour gérer l'ensemble de ces contraintes, Aubay propose dans son framework JS technique des fonctions avancées :

- d'initialisation de la taille de l'iFrame et de son scroll en fonction de paramètres de taille maximum.
- d'affichage du résultat de la recherche.
- de construction de l'url de lancement de la recherche en fonction des critères saisis.

5.10.2 GESTION DE LA PAGINATION

Le deuxième cas d'utilisation d'une iFrame est la gestion d'une pagination sur une partie du contenu d'un écran. Cette pagination consiste à découper le résultat d'une liste d'éléments en pages pour limiter la taille de la liste renvoyée au client. Cette limite du nombre d'éléments pour une page de la liste demandée par le client permet d'améliorer les temps de réponse et d'optimiser la taille du paquet en fonction des capacités de réponse son système d'information.

L'utilisation d'iFrame pour gérer cette pagination permet de découper les informations d'un écran en deux catégories :

- les informations persistantes non liées à la liste des éléments demandée par l'utilisateur (Ex : description d'un dossier).
- les informations non persistantes liées à un paquet de la liste des éléments demandée par l'utilisateur (Ex : liste des fiches associées à un dossier).

Ainsi, lors de l'utilisation du pager pour naviguer à travers les paquets de la liste des éléments demandée par l'utilisateur, seul le contenu de l'iFrame est rafraîchie sans rappel des données persistantes. Ceci offre une amélioration des temps de réponse dès la première utilisation du pager et limite les accès à la source de données aux seules données non persistantes.

Lors du chargement de la page, l'iFrame est initialisée par une page statique présente sur le serveur Web sécurisé pour éviter le bug de sécurisation d'IE (voir §5.10.1) et pour afficher un message d'attente à l'utilisateur.

Cette page d'attente présente dans l'iFrame est mise à jour par le contenu du premier paquet de la liste d'éléments (première page de la pagination) en utilisant la deuxième méthode décrite dans le §5.10.1. Contrairement au §5.10.1, il est inutile de gérer le cas de l'iFrame déclarée en style no display car dans ce cas d'utilisation la gestion de l'affichage ou non de l'iFrame n'est pas implémentée.

Par contre, il est possible d'avoir besoin de gérer plusieurs iFrames dans cet écran. Il est donc nécessaire de préciser le nom de l'iFrame pour retrouver le N° de la frame présente dans la page lors de l'implémentation de la mise à jour de son contenu par l'exécution de la commande JS : document.frames[N° de la frame].location.replace(url).

Cette mise à jour s'accompagne par un redimensionnement de l'iFrame en fonction de la taille du paquet renvoyé par le serveur de la même manière décrite dans le §5.10.1.

Pour simplifier l'implémentation de cette pagination, Aubay propose dans son framework JS technique des fonctions avancées de gestion de pagination :

- Gestion de la mise à jour de l'iFrame à partir d'un numéro de page.
- Initialisation du pager pour indiquer la page courante et le nombre maximum de pages.
- Gestion du pager pour proposer d'accéder à la première page, la page suivante, la page précédente et la dernière page.

L'appel des différents paquets (page de l'iFrame) s'effectue par la mise en place de deux indicateurs sous forme de variable JS :

- "page" indique le n° de la page en cours.
- "MaximumPage" indique le nombre maximum de page accessible.

Le calcul de ces deux indicateurs s'implémente sur le serveur Web par l'intermédiaire du nombre d'éléments présents dans la liste et le nombre désiré d'éléments maximum renvoyés par paquet. Le passage de ces deux indicateurs s'effectue lors du chargement de la page contenu de l'iFrame.

Ainsi, le numéro de la page désirée est instancié en fonction du numéro de la page en cours et du type d'action événement réalisé par le client sur le pager (Première page, Page précédente, Page suivante, Dernière page). Ce numéro de page instancié est ensuite passé en paramètre de l'url pour calculer sur le serveur l'indice du premier élément désiré du paquet de données demandé à la source de données.

De plus, ces deux indices permettent de repositionner la zone dynamique indiquant la position du pager (N° de page en cours/Nombre maximum de page) lors du chargement de la page de contenu de l'iFrame (voir §5.8.3 pour implémentation).

5.10.3 AFFICHAGE D'UN DETAIL PAR SELECTION D'UN ITEM PRESENT DANS L'IFRAME

L'affichage d'un détail par sélection d'une action présente dans l'iFrame consiste à afficher une information complémentaire liée à l'action sélectionnés sur la page contenant l'iFrame (page parent à l'iFrame). Cette action peut consister à la sélection d'une ligne lors de la présence d'une liste sous forme de tableau dans l'iFrame pour obtenir plus d'information sur cette ligne dans la page parent à l'iFrame.

Les traitements effectués lors de cette sélection sont :

1. Désélection de la ligne pré-sélectionnée au préalable.
2. Sélection d'une nouvelle ligne par action de l'utilisateur.
3. Mise à jour du contenu de la zone dynamique associée au détail avec les données complémentaires liées à la sélection.

La sélection/désélection d'item consiste en une mise en valeur par un surlignage ou pas de la ligne du tableau et en un changement de l'icône de sélection (image permettant à l'utilisateur d'effectuer l'action de sélection de la ligne par un click de souris). L'interrogation des objets DOM associés à la ligne du tableau s'implémente comme indiqué dans le §5.4.1, le surlignage comme indiqué dans le §5.8.5 et le changement d'images comme indiqué dans le §5.8.1.

La mise à jour du contenu de la zone dynamique s'effectue à l'aide des principes détaillés dans les §5.4.3, §5.8.3 et §5.12.1.

Ainsi, le framework JS d'Aubay offre les fonctions avancées suivantes :

- Sélection/désélection d'une ligne d' un tableau présent dans l'iFrame.
- Génération des informations complémentaires associées à la sélection de l'utilisateur.
- Affichage du détail d'une ligne en associant la génération correspondant à la sélection.

5.11 GESTION DES COOKIES SESSION PAR LE CLIENT

Les cookies offrent un moyen de stocker des informations côté client et de les faire fournir au serveur par le browser en même temps que la requête de la page. Ces cookies sont stockés dans les en-têtes http à l'aide d'un champs Set-Cookie sous la forme :

Set-Cookie : name=VALEUR ; expires=DATE ; path=CHEMIN ; domain=DOMAINE ; secure

L'information name=VALUE est la seule information qui doit impérativement figurer dans le champ Set-Cookie. Il s'agit simplement d'une chaîne de caractères définissant les informations à stocker dans le cookie. Cette chaîne ne peut contenir ni point-virgules, ni virgules, ni espaces.

Nom	Description
expires=Date	Indique la date d'expiration du cookie. Au delà de cette date, le cookie ne sera plus mémorisé par le client ni envoyé au serveur (DATE est stocké sous la forme Sun, JJ-MM-AA HH :MM :SS GMT) La valeur de expires est paramétrée par défaut sur la fin de la session Navigator en cours.
path=Chemin	Indique la partie de l'URL pour laquelle le cookie est valable. Si l'URL coïncide avec path et domain, le cookie est envoyé au serveur dans l'en-tête de la requête. Si path n'est pas paramétré, sa valeur est identique à celle du document qui définit le cookie.
domain=Domaine	Indique la partie correspondant au domaine des URL pour lesquelles le cookie est valable. La valeur par défaut de l'attribut est le domaine du document en cours qui définit le cookie.
Secure	Indique que le cookie ne doit être transmis que par le biais d'un canal protégé (Ex : utilisant le protocole SSL)

Tableau 40 : description des attributs falcultatifs d'un cookie session

De plus, étant donné que le nombre de cookies acceptés n'excède pas 300, que la taille de chaque cookie ne peut dépasser 4000 caractères et que les navigateurs n'acceptent pour la plupart que 20 cookies par domaine, il est

nécessaire de justifier son utilisation pour un besoin non contournable par un autre moyen (contexte du serveur, champs passé dans une url…).

La manipulation et l'interrogation en javascript de ces cookies s'effectuent par la commande document.cookie pour renvoyer la chaîne de l'en-tête http et la réaffecter.

Pour faciliter l'interrogation et la manipulation de cs cookies, Aubay propose dans son framework JS technique les fonctions avancées suivantes :

- Fonction getCookieVal() est une fonction interne appelée par getCookie(). Lorsque le premier caractère de la valeur couple nom-valeur d'un cookie lui est attribué, elle renvoie cette valeur sous forme de chaîne non codée.
- Fonction GetCookie() sert à accéder à la valeur d'un cookie donné. Elle admet comme argument le nom de ce cookie et en renvoie la valeur. Si le cookie n'existe pas, elle renvoie la valeur null.
- Fonction SetCookie() permet de créer un cookie ou de mettre à jour un cookie existant. Elle nécessite deux arguments et peut en admettre d'autres. Elle s'utilise suivant la syntaxe : setCookie(name, value, expires, path, domain, secure) où expires, path, domain et secure sont des paramètres facultatifs et name et value sont des paramètres obligatoires. name, value, path et domain doivent être des objets string. expires doit être communiqué sous forme d'objet Date, et secure doit être une valeur booléenne. L'ordre des arguments est très important. Si vous souhaitez omettre un argument placé au milieu de la série, vous devrez donc lui attribuer la valeur de substitution null alors les valeurs par défaut seront celles exprimées dans le tableau. Lors de la précision de l'insertion de l'attribut secure à true, un test de vérification est effectué pour déterminer si le protocole SSL est utilisé par le document en cours qui définit le cookie. Cette vérification permet d'éviter tout problème de confusion et de capitaliser le code d'applicatif proposant certaines de ses fonctionnalités en accès sécurisé ou non sécurisé.
- Fonction DeleteCookie() sert à supprimer le cookie désigné par un argument name. La procédure de suppression du cookie consiste à la mettre à jour avec une date d'expiration équivalente à la date et l'heure en cours.

Pour une bonne implémentation de ce framework, il est préférable de détruire tout cookie à la fin de son utilisation et de préciser l'attribut secure à true pour éviter toute confusion lors de la même utilisation en accès sécurisé ou non et renforcer ainsi les aspects sécuritaires.

5.12 GENERATION DE CONTENU PAR LE CLIENT

5.12.1 GENERATION D'UNE PARTIE DU CONTENU D'UN DOCUMENT HTML

Il est possible d'avoir pour besoin de générer dans la page un contenu non formaté en HTML fourni sous forme de variables JS exprimées comme indiqué dans le §5.4.3 ou soit sous la forme de flux XML. Ce contenu peut être généré soit par le serveur WEB applicatif pour des données rapatriées de la base transactionnelle ou par un back office lors d'une gestion de contenu informatif.

Deux méthodes sont envisageables :

1. Une zone dynamique (DIV) dont le contenu est initialisée au chargement de la page en implémentant le principe technique décrit dans le §5.8.3.
2. Les expressions JS référencées document.write() ou document.writeln() pour écrire sur le document de la page un texte à l'emplacement de l'appel de ces fonctions précisé par la balise <script>.

Une troisième méthode n'est pas décrite dans le présent document mais fait l'objet d'une normalisation dans le cadre des normes de réalisation Internet d'un client riche :

3. Appliquer les principes du client riche à l'aide des directives propres au DOM W3C HTML en créant l'ensemble des objets DOM nécessaires au contenu généré et en y intégrant les données présentes dans le flux XML.

Ainsi, dans le cadre de réalisation Internet d'un client léger, il est préconisé d'utiliser la première méthode lorsque le contenu est généré sous forme de variables JS car contrairement à la deuxièmre méthode, elle est plus performante en temps de construction de page en évitant au navigateur de parser deux fois l'ensemble du document HTML complet (avant et après génération du contenu).

Les deux méthodes envisageables sont implémentées sous forme de fonction recevant en argument le contenu formaté en HTML. Ceci nécessite donc pour normaliser et capitaliser le formatage HTML du contenu d'écrire une bibliothèque de fonction recevant en argument le contenu non formaté et fournissant en sortie ce contenu formaté en HTML. L'utilisation de ce principe est pratique lorsque l'on désire aussi mutualiser la charte graphique sur des pages statiques pour que lors de sa modification seule cette bibliothèque soit impactée et non l'ensemble des pages associées (principes mis en œuvre dans le cadre du framework ergonomiques).

L'explication et l'illustration de la première méthode sont décrites dans le §5.8.3 à la différence près qu'au lieu de l'appeler lors d'un événement lié à une action du client, le contenu de la zone dynamique est généré lors du chargement de la page sur un événement onLoad.

5.12.2 GENERATION DU CONTENU D'UNE FENETRE SURGISSANTE

La génération par le navigateur client du contenu d'une fenêtre surgissante comme d'ailleurs pour une page complète s'effectue par les fonctions DOM W3C HTML.

Pour créer la fenêtre fille, il est nécessaire d'utiliser les principes définis dans le §5.9.2 en ne précisant en entrée aucune url pour créer un document vide. Les fonctions définies dans le §5.9.2 sont modifiées pour intégrer un événement onLoad sur la création de cette fenêtre déclenchant la génération du contenu de celle-ci.

La génération ne peut pas s'effectuer seulement à l'aide de la première méthode décrite dans le §5.12.1 car elle nécessiterait d'appeler un document possédant une zone dynamique (DIV) dans la fenêtre fille vide de contenu informatif. Or, le contenu du DIV d'une fenêtre fille à partir d'une fenêtre parent ne peut être manipulé directement à l'aide de fonction innerHTML du DOM Level 0.

Il est donc nécessaire de créer entièrement le document de la fenêtre fille à l'aide de fonctions et d'attribut du DOM W3C en plus du innerHTML du DOM Level 0 :

- document.title pour fixer le titre du document.
- document.body.setAttribute pour fixer les attributs du BODY du document.
- document.createElement("DIV") pour créer un élément DOM de type DIV dans le document.
- element.innerHTML pour générer le contenu de la zone dynamique.
- document.body.appendChild pour ajouter au document la zone dynamique et son contenu.

Il serait possible de générer entièrement le contenu du document HTML de la fenêtre fille sans utiliser de DOM Level 0 en implémentant des fonctions DOM W3C HTML pour générer par exemple l'ensemble des tableaux imbriqués liés à la charte graphique. Mais, il est préférable d'utiliser l'innerHTML pour sa facilité de mise en œuvre et sa performance (voir annexe Annexe J : Comparatif innerHTML vs DOM W3C) sauf lors d'un parcours déjà effectué en DOM W3C XML comme dans le cas de la récupération de données présentes dans un flux XML où alors les préconisations de mise en œuvre sont détaillées dans le document des normes de réalisation Internet d'un client riche.

Ainsi, dans le cadre de réalisation Internet d'un client léger, il est préconisé d'utiliser cette méthode lorsque le contenu est généré sous forme de variables JS et d'utiliser le framework ergonomique et les préconisations des normes de réalisation Internet d'un client riche lorsque le contenu est généré sous forme de flux XML.

Attention : cette méthode de génération de contenu de fenêtre fille est intéressante lorsque l'ensemble des données est déjà présente dans la fenêtre parente comme dans le cas d'une version imprimable. Ceci permet d'économiser les échanges client/serveur WEB et le nombre de requêtes à la source de données.

Exemple d'implémentation pour la version imprimable appliquant les principes de l'exemple du §5.8.3 :

Modification du code JavaScript du §5.9.2 pour ajouter l'événement onLoad lors de la création de la fenêtre fille :

```
function OpenWindow(url,nom,options)
{
  msgWindow = window.open(url,nom,options);
  msgWindow.onload = ConstruireWindow();
}
```

Code JavaScript pour générer le contenu du document HTML de la fenêtre fille :

```
function ConstruireWindow()
{
  msgWindow.document.title = "VERSION IMPRIMABLE";
  msgWindow.document.body.setAttribute("bgcolor","#FFFFFF");
  msgWindow.document.body.setAttribute("leftmargin","0");
  msgWindow.document.body.setAttribute("topmargin","0");
  msgWindow.document.body.setAttribute("marginwidth","0");
  msgWindow.document.body.setAttribute("marginheight","0");
  var newDiv = msgWindow.document.createElement("DIV");
  newDiv.id = "Texte";
  newDiv.innerHTML = frames[0].ConstruireVersionImprimable();
  msgWindow.document.body.appendChild(newDiv);
}
```

5.13 GESTION D'ENVOI D'UN MESSAGE ELECTRONIQUE

5.13.1 ENVOI D'UN MESSAGE ELECTRONIQUE A UN DESTINATAIRE

Il est possible d'ouvrir l'outil de messagerie du client en précisant l'adresse du destinataire par le biais de commande mailto:adresse_du_destinataire dans un lien hypertexte.

L'outil de messagerie utilisé est celui précisé en paramétrage du navigateur.

5.13.2 FORMATAGE DU SUJET ET DU CORPS D'UN MESSAGE ELECTRONIQUE

Pour formater le sujet et le corps d'un e-mail, il suffit de spécifier derrière le mailto:adresse_email? les champs subject pour le sujet du message et body pour le corps du message. Ce formatage est possible sur l'ensemble des navigateurs certifiés sauf pour IE 5.0.

Ce paramètre body permet de seulement préciser le texte du message sans pouvoir spécifier un formatage avancé sur ce texte (Police, couleur, souligné, gras…).

Il est possible sur ce corps du message d'y intégrer des caractères spéciaux et y effectuer un formatage d'alignement simple par le biais d'insertion de saut de page, saut de ligne, retour chariot ou tabulation.

Formatage particulier du corps du message (Paramètres du Body) :

\f Saut de page

\n Saut de ligne

\r Retour chariot

\t Tabulation

\' Apostrophe ou guillemet simple

\" Guillemet double

\\ Backslash (\)

\XXX Séquence octale

\xXX Séquence hexadécimale

\uXXXX Séquence Unicode.

Exemple d'implémentation sur une page de test HTML :

```
<head>
<title>Test Mailto</title>
<script type="text/javascript">
function sendEmail(Address, Subject, Body)
{
  var url = "mailto:" + escape(Address) + "?subject=" + escape(Subject) + "&body=" +
escape(Body);
  window.location.href = url;
}
</script>
</head>
<body>
<a href="javascript:sendEmail('speguet@aubay.com','Test mailto','Ceci est un
test\rCeci est un test de saut de ligne')">Test Mailto pré-formaté</a>
</body>
```

5.14 VALIDATION DU CODE AVEC JSLINT ET JAVASCRIPT LINT

5.14.1 PRESENTATION

JavaScript est un langage volontairement permissif sur de nombreux aspects, ce qui introduit le risque de voir apparaître de nombreuses erreurs au sein du code, à commencer par des erreurs de syntaxe.

Pour parer à ces éventualités, le vérificateur historique de code JavaScript, JSLint, a été mis au point par le créateur même du langage, sous la forme d'un formulaire Web.

Un nouveau vérificateur vient de voir le jour, sous le nom de JavaScript Lint. Celui-ci reprend les principes établis par JSLint, mais l'outil est ici construit autour d'une extension de SpiderMonkey, le moteur JavaScript de Firefox, ce qui lui assure une certaine robustesse.

Le pôle d'architecture DSI/PSI préconise la validation des fichiers javascript par l'utilisation de JSLint et JavaScript Lint

5.14.2 FONCTIONNALITES DE JSLINT

JSLint permet de rechercher des erreurs courantes en Javascript :

- **Points-virgule** : Javascript permet de les rendre optionnels grâce à un mécanisme d'insertion automatique de points-virgule. JSLint attend un point-virgule à la fin de chaque instruction sauf pour les instructions *for*, *function*, *if*, *switch*, *try* et *while*.
- **Fin de ligne** : toujours pour se défendre du mécanisme d'insertion automatique de point-virgule, JSLint attend que chaque ligne se termine par une des chaînes suivantes : , . ; : { } ([= < > ? ! + - * / % ~ ^ | & == != <= >= += -= *= /= %= ^= |= &= << >> || && === !== <<= >>= >>> >>>=
- **Virgule** : cet opérateur peut être amené à masquer certaines erreurs de programmation. JSLint attend donc que la virgule soit utilisée comme un séparateur et non un opérateur (hormis dans l'initialisation et l'incrémentation d'une boucle *for*). De même, tous les éléments d'un tableau doivent être renseignés lors de sa déclaration. Une virgule ne doit pas apparaître après le dernier élément sous peine de mauvaise interprétation par certains navigateurs.
- **Blocs requis** : JSLint attend des instructions if et for qu'elles soient entourées d'un bloc {}. Javascript permet d'écrire une instruction if de cette manière :

```
if(condition)
    instruction
```

  Cette forme est connue pour être source d'erreur dans des projets ou de nombreux développeurs travaillent sur le même code. C'est pourquoi JSLint attend l'utilisation d'un bloc :

```
if(condition){
    instruction
}
```

- **Blocs interdits** : dans beaucoup de langages, la portée d'une variable est propre à un bloc. En Javascript, les blocs n'induisent pas une portée. La portée est juste limitée aux fonctions, ce qui prête à confusion même pour les programmeurs expérimentés. JSLint attend des blocs avec les instructions *function*, *if*, *switch*, *while*, *for*, *do* et *try* et pas ailleurs.
- **Expression** : une expression doit être une affectation ou un appel de fonction. Les autres expressions sont considérées comme des erreurs.

var

Javascript permet d'utiliser la définition de variable *var* partout dans une fonction. JSLint est plus strict. Il attend qu'une variable ne soit définie qu'une fois et qu'elle soit déclarée avant d'être utilisée. JSLint ne permet pas de déclarer des paramètres à l'aide de *var*. Il ne permet pas non plus de déclarer les arguments d'un tableau à l'aide de *var*. JSLint ne permet pas enfin d'utiliser *var* pour créer des variables ou paramètres existants dans un bloc d'une autre portée.

switch

Une erreur commune dans une instruction *switch* est d'oublier de placer un *break* après chaque *case*. JSLint vérifie donc que l'instruction avant un *case* est bien l'une de celles-ci : *break*, *case*, *continue*, *return*, *switch* ou *throw*.

with

L'instruction *with* est sensée fournir une abréviation pour accéder aux propriétés des objets. Malheureusement *with* se comporte très mal à l'initialisation de nouvelles propriétés. JSLint interdit donc l'usage de *with* et attend l'instruction *var* à la place.

=

JSLint empêche les assignations dans la condition d'une instruction *if* ou *while*. Car en général, un bloc comme celui-ci :

```
if(a = b){
    ….
}
```

aurait dû être celui-ci :

```
if(a == b){
    ….
}
```

Si l'on veut vraiment faire une affectation, il faut l'entourer de parenthèses :

```
if((a = b)){
    ….
}
```

== et !=

Les opérateurs *==* et *!=* font une coercition (cast) avant la comparaison. Ce qui signifie par exemple que le test *'' == 0* renvoie vrai et entraîne des erreurs de type. Il convient alors d'utiliser les opérateurs === et !== qui ne font pas de coercition.

SI on veut faire une coercition, il faut utiliser la forme abrégée. A la place de :

```
(i != 0)
```

utiliser

```
(i)
```

A la place de

```
(i == 0)
```

utiliser

```
(!i)
```

- **Labels :** Javascript permet à n'importe quelle instruction d'avoir un label. JSLint est plus strict et permet seulement aux instructions interagissant avec *break*, *switch*, *while*, *do* et *for* d'avoir des labels. JSLint attend que les labels soient distincts des variables et paramètres.

- **Code non lu :** JSLint attend qu'une instruction *return*, *break*, *continue* ou *throw* soit suivie par *}*, *case* ou *default*.
- **Confusions sur les opérateurs :** JSLint empêche qu'un + soit suivi de + ou ++ et q'un– soit suivi de – ou --. Il faut utiliser des parenthèses pour éviter la confusion.
- **Opérations ++ et-- :** Les opérations ++ (incrémentation) et – (décrémentation) sont connues pour conduire à de mauvaises pratiques en encourageant des raccourcis dans le code. Ils sont la deuxième source de trous de sécurité et d'exposition aux virus dans les architectures applicatives. C'est pourquoi certains estiment que la non-utilisation de ces opérateurs peut encourager le développement de programmes de meilleure qualité.
- **Eval :** La fonction *eval* (et ses dérivées, *Function*, *setTimeout* et *setInterval*) donne l'accès au compilateur Javascript. Cela peut être parfois utile pour parser du texte JSON (JavaScript Object Notation) mais dans pratiquement tous les autres cas, cela relève d'un code très mauvais.
- **Void :** dans la plupart des langages basés sur le langage C, *void* est un type. En Javascript, *void* est un opérateur préfixe qui renvoie toujours *undefined*. C'est pourquoi JSLint ne permet pas l'utilisation de *void* car il prête à confusion sans être vraiment utile.
- **Expressions régulières :** la syntaxe Javascript pour les expressions régulières surcharge le caractère /. Pour éviter toute ambiguïté, JSLint attend que le caractère précédant une expression régulière soit le caractère *(* ou *=* ou *:* ou *,*. La spécification ECMAScript impose que le caractère / soit échappé dans une expression régulière.
- **Variables non définies :** en Javascript, les variables non-définies sont implicitement considérées comme des variables globales. JSLint prévoit une option permettant de détecter cela. L'utilisation des déclarations */*extern*/* indique des symboles définis dans d'autres modules.
- **Constructeurs :** les constructeurs sont des méthodes préfixées par *new.* Le préfixe *new* crée un nouvel objet basé sur le prototype de la méthode et lie cet objet à cette méthode. Si l'on omet le préfixe *new,* aucun nouvel objet ne sera créé et une liaison sera faite à l'objet global. C'est un sérieux problème. JSLint oblige que le nom des constructeurs commence par une lettre majuscule après le préfixe new. JSLint ne reconnaît pas les déclarations de la forme : *new Number*, *new String*, *new Boolean* JSLint ne reconnaît pas la déclaration *new Object* (utilisation de {} à laplace) JSLint ne reconnaît pas la déclaration *new Array* (utilisation de [] à laplace)
- **Hors contrôle :** JSLint ne contrôle pas l'utilisation des commentaires et des espaces. Il ne se soucie pas du formatage du code. La seule exception est le test de fin de ligne. JSLint ne fait pas d'analyse de flux pour déterminer si une variable est assignée avant d'être utilisée. Ceci est dû au fait que la valeur attribuée par défaut aux variables (*undefined*) est acceptable. JSLint ne fait pas non plus d'analyse globale. Il ne contrôle pas si une fonction utilisée avec *new* est vraiment un constructeur ou si le nom des méthodes est correctement orthographié.
- **HTML :** JSLint est capable de gérer le HTML. Il peut inspecter le code Javascript contenu entre les balises *<scripts>…</script>*. Il inspecte également le contenu du code HTML pour rechercher les problèmes connus pour interférer avec le Javascript :
  - Tous les noms de balise doivent être en minuscule
  - Toutes les balises qui peuvent être fermées doivent l'être (exemple *</p>*)
  - Toutes les balises doivent être correctement imbriquées
  - Le caractère *<* doit être traduit par son code HTML

  JSLint est moins exigeant que le XHTML mais plus strict que ce qui est permis par la plupart des navigateurs.

  JSLint vérifie également les occurrences de la chaîne *</* . Il faut écrire *<\/* à la place. L'antislash supplémentaire est ignoré par le compilateur Javascript mais pas par le parseur HTML. Ce genre d'astuce reste nécessaire.

- **Rapport :** Si la source est telle que JSLint l'attend, il génère un rapport sur les fonctions. Il décrit pour chaque fonction :
  - Le numéro de ligne auquel elle commence
  - Son nom. En cas de fonction anonyme, JSLint « devine » le nom.
  - Les paramètres (arguments inclus si utilisés)
  - Les variables
  - Les variables globales. Un élément inattendu peut être une indication d'erreur.
  - Les labels

5.14.3 FONCTIONNALITES DE JAVASCRIPT LINT

JavaScript Lint reprend la plupart des fonctionnalités de JSLint tout en s'appuyant sur le moteur Javascript de Firefox :

- Points-virgule manquants en fin de ligne
- Accolades sans présence d'instructions if, for, while, etc
- Portion de code jamais exécutée à cause d'une instruction return, throw, continue ou break
- Instructions case dans un switch qui ne sont pas terminées par un break
- Virgules en trop sur les nombres
- Un 0 en trop qui transforme un nombre en base octal
- Commentaire dans un commentaire
- Ambiguïté entre deux lignes adjacentes qui appartiennent à la même instruction
- Instructions vides
- Expressions régulières non précédées par une parenthèse gauche, une affectation, deux points ou une virgule
- Instructions séparées par une virgule au lieu d'un point-virgule
- Utilisation de l'opérateur d'incrémentation ++ ou de décrémentation – sauf pour des instructions simples comme « i++ » ou « --i »
- Utilisation du type void
- Succession de signes plus (ex : x+++y) ou moins (x---y)
- Utilisation d'instructions if, for, while sans accolades.

6 NORME DE DEVELOPPEMENT PLUGINS

6.1 APPLET

6.1.1 INTRODUCTION

Les applets sont des applications écrites en Java déclarées au sein d'une page HTML. Comme tout programme Java, elles s'exécutent au sein d'une machine virtuelle (JVM – Java Virtual Machine).

Le navigateur utilise par défaut sa propre machine virtuelle pour exécuter les applets. Il peut également utiliser une machine virtuelle installée comme plug-in.

La version 1.1 du JDK (Java Development Kit), est la version minimale utilisée sur les navigateurs actuels. Les machines virtuelles livrées avec les nouveaux navigateurs sont maintenant toutes compatibles java 2 (JDK1.2 et +).

6.1.2 RESTRICTION DE SECURITE

Les Applets s'exécutent dans un environnement sécurisé.

- Les Applets ne peuvent ni charger de librairies ni définir de méthodes natives.
    - Les Applets ne doivent utiliser que leur propre code Java ainsi les API Java fournies par l'environnement d'affichage de l'Applet (Navigateur, AppletViewer, …).
    - Chaque environnement d'exécution d'Applet doit fournir au minimum les classes des packages java.*.
- Les Applets ne peuvent pas lire ou écrire des fichiers sur le disque de l'hôte qui l'exécute.
    - Les Applets n'établissent pas de connexion réseau à l'exception de l'hôte à partir duquel elles sont chargées.
- Les Applets ne peuvent pas démarrer de programme sur l'hôte qui les exécute.
- Les Applets ne peuvent pas lire toutes les propriétés système de la JVM de l'hôte qui les exécute
    - Les paramètres que l'Applet ne peut pas voir :
        - java.class.path : CLASSPATH de la JVM
        - java.home : répertoire d'installation de la JVM
        - user.dir : répertoire courant de travail de l'utilisateur
        - user.name : nom de l'utilisateur
    - Les paramètres que l'Applet peut voir :
        - file.separator : le séparateur de fichier ("/")
        - java.class.version : numéro de version de la classe
        - java.vendor : nom du fournisseur de la JVM
        - java.vendor.url : url du fournisseur de la JVM
        - java.version : version de la JVM
        - line.separator : séparateur de ligne
        - os.arch : Operating System architecture
        - os.name : nom du système d'exploitation
        - path.separator : séparateur de chemin (";")
- Les fenêtres ouvertes par une Applet contiennent un avertissement permettant de les distinguer des autres fenêtres.

6.1.3 TAG APPLET

La balise Applet étant, d'une part, dépréciée dans le cadre dans la norme W3C HTML 4.01 et, d'autre part, son usage ne permettant pas d'avoir une gestion fine des machines virtuelles Java, son utilisation n'est donc pas autorisée.

Elle sera avantageusement remplacée par l'utilisation conjointe des balises Object et Embed (cf chapitre 6.1.4)

6.1.4TAGS OBJECT ET EMBED

L'utilisation combinée des tags object et embed permet de déclarer une Applet pour qu'elle s'exécute dans 'une version minimum particulière de la JRE (Java Runtime Environment) en assurant une compatibilité élargie des navigateurs.

Ce fonctionnement permet également de proposer le téléchargement à l'utilisateur si la version de la JVM installée sur le navigateur ne convient pas.

Internet Explorer reconnaît la balise <object> et ignore le contenu de la balise <comment>. La balise <embed> n'est donc pas prise en compte.

Les navigateurs Mozilla ignorent la balise <object> avec un attribut classid de type "clsid:x…x" et interprètent le contenu de la balise <comment>. La balise <embed> est ainsi interprétée.

Prototype :

```
<object
    classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
    [codebase="<URL>#Version=major,minor,micro,update"]
    width="pixels"
    height="pixels"
    [align="alignement"]
    [hspace="pixels"]
    [vspace="pixels"]>
    [name="nomInstanceApplet"]>
    <param name="code" value="NomApplet.class">
    [<param name="codebase" value="codebaseURL">]
    [<param name="type" value="application/x-java-applet;version=x.x">]
    [<param name="name" value=" nomInstanceApplet">]
    [<param name="archive" value="fichierArchive1.jar,fichierArchive2.jar">]
    [<param name="scriptable" value="booléen">
    [<param name="nomParamètre1" value="valeurParamètre1">]
    [<param name="nomParamètre2" value="valeurParamètre2">]
    <comment>
        <embed
            type="application/x-java-applet;version=x.x"
```

```
            width="pixels"
            height="pixels"
            [align="alignement"]
            [hspace="pixels"]
            [vspace="pixels"]
            code="NomApplet.class"
            [codebase="codebaseURL"]
            [name="nomInstanceApplet"]
            [pluginspage="pluginspageURL"]
            <noembed>
                [HTMLSiNavigateurNonCompatibleApplet]
            </noembed>
        </embed>
    </comment>
</object>
```

- **classid**="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"

L'attribut classid identifie la version du plugin Java à utiliser. La valeur indiquée ci-dessus précise qu'il faut utiliser la dernière version installée du plugin Java.

Cet attribut est requis.

- **[codebase**="<URL>#Version=major,minor,micro,update"] (Balise object uniquement)

L'attribut codebase indique que si la version de JRE utilisée par le navigateur est inférieure à la version indiquée, alors l'utilisateur est invité à télécharger cette version à l'URL indiquée.

Ex : codebase="http://java.sun.com/update/1.5.0/jinstall-1_5_0-windows-i586.cab#Version=1,5,0,0"

- **width**="pixels"

L'attribut width spécifie la largeur en pixels de l'Applet au sein de la page HTML.

Cet attribut est requis.

- **height**="pixels"

L'attribut height spécifie la hauteur en pixels de l'Applet au sein de la page HTML.

Cet attribut est requis.

- [**align**="alignement"]

L'attribut align permet de positionner l'alignement de l'Applet. Les valeurs sont les mêmes que pour un tag img.

- [**vspace**="pixels"]

L'attribut vspace spécifie le nombre de pixels au-dessus et en dessous de l'Applet. Ce paramètre a les mêmes effets que sur un tag img.

- [**hspace**="pixels"]

L'attribut hspace spécifie le nombre de pixels au-dessus et en dessous de l'Applet. Ce paramètre a les mêmes effets que sur un tag img.

- [**name**="nomInstanceApplet"]

L'attribut name permet de donner un nom à l'instance de la classe Applet.

Ce nom permet entre autres à du code JavaScript d'appeler les propriétés et méthodes publiques de l'objet Applet comme s'il s'agissait d'un objet JavaScript.

- **param name**="**code**" value="NomApplet.class" (Balise object) et **code**="NomApplet.class" (Balise embed)

La valeur du paramètre code correspond au nom de la classe Applet à instancier.

Cet attribut est requis.

- [**param name**="**codebase**" value="codebaseURL"] (Balise object) et [**codebase**="codebaseURL"] (Balise embed)

Le paramètre codebase spécifie l'URL où se trouvent les fichiers nécessaires à l'Applet y compris la classe spécifiée par le paramètre code.

En l'absence de ce paramètre et du paramètre archive, les fichiers de l'Applet sont recherchés sur l'URL du document courant.

Le répertoire codebase vient s'ajouter au CLASSPATH de l'Applet.

- **[param name**="**type**" value="application/x-java-applet;version=x.x.x"] (Balise object) et **type**="application/x-java-applet;version=x.x.x" (Balise embed)

Le paramètre type précise le type mime de l'applet ainsi que la version minimum requise de la JRE.

Cet attribut est requis pour la balise embed.

Ex : type="application/x-java-applet;version=1.5" nécessite au moins la version 1.5 de la JRE

- **[param name**="**name**" value=" nomInstanceApplet"]

Identique à l'attribut name.

- [**param name**="**archive**" value="fichierArchive1.jar,fichierArchive2.jar"]

Le paramètre archive spécifie un ou plusieurs fichiers d'archive java contenant les classes et les ressources utilisées par l'Applet (y compris l'Applet).

Les fichiers d'archives viennent s'ajouter au CLASSPATH de l'Applet.

L'utilisation de fichiers d'archive permet de réduire considérablement le nombre de requêtes HTTP vers le serveur lorsque l'Applet a besoin de plusieurs autres classes ou ressources pour s'exécuter.

De plus, avoir recours à leur utilisation permet d'éviter des erreurs d'exécution lorsqu'une classe n'a pas pu être chargée à la suite d'une erreur sur un échange HTTP.

- [**param name**="**scriptable**" value="booleen"]

Le paramètre scriptable spécifie si l'applet est scriptable à partir d'une page HTML.

- [<**param name**="nomParamètre1" **value**="valeurParamètre1">]

Les paramètres d'initialisation de l'Applet sont spécifiés au moyen des attributs param. Ces paramètres sont chargés par l'Applet lors de son chargement.

- [**pluginspage**=" pluginspageURL"]

Le paramètre pluginspage spécifie l'URL d'une page HTML permettant le téléchargement de la bonne version du plugin Java si celui qui est installé n'est pas conforme à l'attribut type

Ex :. pluginspage="http://java.sun.com/products/plugin/index.html »

- **<noembed>** TexteHTMLSiNavigateurNonCompatibleApplet **</noembed>**

Le code HTML figurant entre les balises <noembed> et </noembed> est affiché si le navigateur ne supporte pas la balise <embed> ou si le navigateur échoue à lancer la JRE.

L'utilisation systématique de la balise <noembed> permet d'informer l'utilisateur final d'un dysfonctionnement ou d'une inadéquation de son navigateur vis-à-vis des applets. Le texte d'information à présenter est :

Une incompatibilité ou un dysfonctionnement du navigateur a entrainé une erreur d'exécution

Exemple d'Applet déclarée avec les tags <object> et <embed> :

```
<object
    classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
    codebase="http://java.sun.com/update/1.5.0/jinstall-1_5-windows-
i586.cab#Version=1,5,0,0"
    width="170"
    height="150">
    <param name="code" value="Clock.class">
    <param name="type" value="application/x-java-applet;version=1.5">
    <param name="scriptable" value="false">
    <comment>
        <embed
            type="application/x-java-applet;version=1.5"
            code="Clock.class"
            width="170"
            height="150"
            scriptable="false"
            pluginspage="http://java.sun.com/products/plugin/index.html#download">
        <noembed>
        Une incompatibilit&eacute; ou un dysfonctionnement du navigateur a
entrain&eacute; une erreur d'ex&eacute;cution
        </noembed>
```

```
        </embed>
    </comment>
</object>
```

6.1.5 RECOMMANDATIONS

Pour atteindre une ***compatibilité maximale*** avec les navigateurs ciblés, il est nécessaire de ne pas développer les applets dans une version supérieure au ***JDK 5.0*** vis à vis de la compatibilité attendue.

Les connexions HTTP et HTTPS établies directement par les applets peuvent poser des problèmes de cookies de session avec certaines versions des plug-in Java. Il faut donc ***éviter toute connexion sur l'initiative d'une applet*** dans le cadre de déploiement sur Internet.

Pour réduire le nombre d'échanges entre le navigateur et le serveur, il est nécessaire ***d'utiliser des fichiers d'archive plutôt qu'un répertoire sur le serveur pour fournir les classes*** **de l'applet**. Ceci permet également de supprimer les dysfonctionnements induits par le réseau, lorsqu'une des classes utilisées par l'Applet n'a pas été récupérée.

Les Applets doivent être de faible poids pour éviter les lenteurs de téléchargement. Les Applets deviennent rapidement trop volumineuses, les méthodes de développement doivent en tenir compte.

6.2 FLASH

6.2.1 INTRODUCTION

Les applications Flash sont des animations interactives s'exécutant au sein d'une page HTML.

Flash est une technologie de MACROMEDIA. Le plug-in d'exécution des Flashs ainsi que l'environnement de développement sont fournis par l'éditeur.

La facilité de téléchargement et l'intégration par défaut dans IE du plug-in ont permis une très forte pénétration de la technologie Flash.

6.2.2 TAGS OBJECT ET EMBED

Les plug-in flash sont déclarés avec les tags object et embed.

Exemple d'implémentation :

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version
=6,0,40,0" width="550" height="400" id="myMovieName">

<param name="movie" value="myFlashMovie.swf">
<param name="quality" value="high">
<param name="bgcolor" value="#FFFFFF">

<embed src="myFlashMovie.swf" quality="high" bgcolor="#FFFFFF" width="550"
height="400" name="myMovieName" align="" type="application/x-shockwave-flash"
pluginspage="http://www.macromedia.com/go/getflashplayer">
</embed>
</object>
```

Attributs obligatoires

- width

Largeur de la zone d'affichage du flash en pixels ou en pourcentage.

- height

Hauteur de la zone d'affichage du flash en pixels ou en pourcentage.

- src

URL ou l'animation flash peut être chargé pour *tag embed uniquement*.

- pluginspage

URL ou le plugin flash peut être chargé si ce dernier n'est pas encore installé.

- movie

URL où se trouve le fichier de contrôle flashplayer pour *tag object uniquement*.

- classid

Identifie le plugin activeX dans le navigateur pour *tag object uniquement*.

- codebase

Les navigateurs IE se servent de l'attribut codebase pour déterminer la version de Flash à utiliser. Si la version installée sur le poste de travail de l'utilisateur est inférieure à la version contenue dans l'attribut codebase, la version la plus récente est directement installée depuis l'url précisée.

codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=6,0,65,0"

Dans l'exemple ci-dessus, la version requise est la numéro 6 release 65.

Pour *tag object uniquement*.

Attributs optionnels

- name

Identifie l'animation flash sur le navigateur pour permettre les interactions avec JavaScript pour *tag embed uniquement*.

- id

Identifie l'animation flash sur le navigateur pour permettre les interactions avec des langages de script pour *tag object uniquement*.

- swliveconnect (true, false)

Spécifie si le browser doit démarrer Java lorsqu'il lance le lecteur Flash pour la première fois. Cet attribut est à false par défaut. Si JavaScript est utilisé conjointement avec Flash, Java doit être lancé pour permettre aux FSCommand de fonctionner.

- play (true, false)

Spécifie si l'animation flash doit démarrer dès le chargement dans le navigateur effectué. Ce paramètre est à true par défaut.

- loop (true, false)

Spécifie que l'animation doit se répéter indéfiniment ou s'arrêter après la dernière prise. La valeur par défaut est true.

- quality (low, high, autolow, autohigh, best)

low permet de donner la priorité à la rapidité de lecture au détriment de l'apparence (non utilisation de l'anti-aliasing).

Autolow donne la priorité à la vitesse tout en améliorant l'apparence dès que possible. La lecture commence sans l'anti-alias. Si Flash considère que le poste utilisateur peut le supporté, il est automatiquement activé.

Autohigh donne la priorité à la vitesse et à l'apparence par défaut et sacrifie l'apparence si nécessaire. Le mode anti-alias est activé par défaut. Il est désactivé si nécessaire.

Medium donne un niveau intermédiare dans l'anti-alias entre low et high

High donne la priorité à l'apparence. L'anti-alias est toujours actif. L'anti-alias est légèrement dégradé sur les bitmaps lors des animations.

Best fournit la meilleure qualité d'affichage sans tenir compte du temps de lecture.

- bgcolor (#RRGGBB, code RGB en hexadécimale)

Spécifie la couleur du fond de l'animation flash. Cet attribut surcharge l'attribut positionné dans l'animation flash. Ce paramètre est sans impact sur la couleur du fond de la page HTML.

- scale (showall, noborder, exactfit)

showall rend la totalité de l'animation visible dans la zone spécifiée sans engendrer de distorsion. Des bordures peuvent apparaître de chaque côté de l'animation. Cette valeur est positionnée par défaut.

*noborde*r met à l'échelle l'animation pour remplir la zone spécifiée sans distorsion. Certaines partie de l'animation peuvent ne plus être visibles.

exactfit rend l'intégralité de l'image visible dans l'espace spécifié sans bordures. L'image peut présenter des distorsions.

- align (l, t, r, b)

L'animation est toujours centrée par défaut dans la fenêtre du navigateur. Les bords de l'animation sont rognés si la fenêtre est trop petite.

Left, Right, Top et Bottom alignent l'animation sur le bord correspondant de la fenêtre du navigateur en rognant les bords de l'animation si la fenêtre est trop petite.

- salign (l, t, r, b, tl, tr, bl, br)

l, r, t et b alignent respectivement l'animation sur les bords gauche, droit, haut ou bas. L'image est rognée sur les 3 bords non alignés si nécessaires.

tl, tr, bl et br alignent respectivement sur les coins supérieur gauche, supérieur droit, inférieur gauche et inférieur droit. L'image est rognée si nécessaire en maintenant visible le coin aligné.

- base (., répertoire de base ou url)

Spécifie le répertoire de base ou l'URL qui doit être utilisé pour résoudre toutes les demandes de chemin de l'animation.

- menu (true, false)

Affiche un menu complet à l'utilisateur si l'option est activée. Seule la rubrique à propos est affiché dans le cas contraire

6.2.3 RECOMMANDATIONS

Lors de l'utilisation de ***HTTPS***, il faut préfixer les URL de téléchargement du plug-in par https:// afin d'éviter les alertes de sécurité de certaines versions de IE6. Les URL HTTPS ont été créées par MACROMEDIA pour répondre à ce problème.

6.3 PRECONISATIONS POUR L'UTILISATION DE FLASH OU D'UNE APPLET

L'utilisation de plug-in flash ou Applet n'est pas une solution autorisée de fait. Toute utilisation doit faire l'objet d'une étude spécifique.

Les plug-in ont un poids important et une certaine lenteur d'utilisation qui en restreignent l'usage.

Les Applets, compte tenu de la diversité des machines virtuelles, ne proposent pas en standard la même qualité graphique pour un poids équivalent à une animation Flash. Cet écart pourrait être comblé par l'utilisation de JVM Java 2 qui offrent des outils graphiques beaucoup plus évolués en standard. Les Applets conviennent bien pour réaliser de petites animations interactives qui nécessitent une rapidité de chargement et d'initialisation (pourvu de rester raisonnable lors du développement).

Les animations Flash ont un poids et un temps d'initialisation relativement incompressibles. Elles sont cependant bien moins lourdes et beaucoup plus ergonomiques sur des animations interactives complexes.

Dans l'absolu, l'usage de ces plug-ins doit rester réservé à des fonctionnalités "optionnels" ou "de confort" des applications Internet.

ANNEXE A : CARACTERES SPECIFIQUES EN HTML

Voici une liste des caractères spéciaux. Il suffit d'insérer le code du caractère pour l'obtenir dans votre page. Avec cette manipulation, vous garantissez que ces caractères spéciaux seront reconnus correctement par tous les navigateurs cibles de Logitel Net. Cette liste est basée sur le charset utilisé par Logitel Net : ISO 8859-1 (ISO-Latin 1).

Entité HTML	Code Iso	Description	Entité HTML	Code Iso	Description
		Espace insécable	&Ntilde;	&#209;	N tilde (Ñ)
&iexcl;	&#161;	Point d'exclamation inversé (¡)	&Ograve;	&#210;	O accent grave (Ò)
&cent;	&#162;	« cent » (monnaie américaine)	&Oacute;	&#211;	O accent aigu (Ó)
&pound;	&#163;	Livre sterling (£)	&Ocirc;	&#212;	O accent circonflexe (Ô)
&curren;	&#164;	Symbole monétaire (¤)	&Otilde;	&#213;	O tilde (Õ)
&yen;	&#165;	Yen (¥)	&Ouml;	&#214;	O tréma (Ö)
&brvbar;	&#166;	Barre verticale (¦)	&times;	&#215;	Symbole de la multiplication (×)
&sect;	&#167;	Symbole de section (édition)	&Oslash;	&#216;	O barré (Ø)
&uml;	&#168;	Tréma (¨)	&Ugrave;	&#217;	U accent grave (Ù)
&copy;	&#169;	Copyright (©)	&Uacute;	&#218;	U accent aigu (Ú)
&ordf;	&#170;	Indicateur ordinal féminin (ª)	&Ucirc;	&#219;	U accent circonflexe (Û)
&laquo;	&#171;	Guillemet anglais ouvrant (")	&Uuml;	&#220;	U tréma (Ü)
&not;	&#172;	Crochet de négation (¬)	&Yacute;	&#221;	Y accent aigu (Ý)
&shy;	&#173;	Tiret de césure (-)	&THORN;	&#222;	THORN (Islande : Þ)
&reg;	&#174;	Marque déposée (®)	&szlig;	&#223;	Sz ligature (ß)
&macr;	&#175;	Macron (¯)	&agrave;	&#224;	a accent grave (à)
&deg;	&#176;	Degré (°)	&aacute;	&#225;	a accent aigu (á)
&plusmn;	&#177;	Symbole « plus ou moins » (±)	&acirc;	&#226;	a accent circonflexe (â)
&sup2;	&#178;	Exposant 2 (²)	&atilde;	&#227;	a tilde (ã)
&sup3;	&#179;	Exposant 3 (³)	&auml;	&#228;	a tréma (ä)
&acute;	&#180;	Accent aigu (´)	&aring;	&#229;	a anneau (å)
&micro;	&#181;	Lettre grecque mu (µ)	&aelig;	&#230;	e dans l'a (æ)

&para;	&#182;	Symbole « paragraphe » (§)	&ccedil;	&#231;	c cédille (ç)
&middot;	&#183;	Point médiant (·)	&egrave;	&#232;	e accent grave (è)
&cedil;	&#184;	Cédille	&eacute;	&#233;	e accent aigu (é)
&sup1;	&#185;	Exposant 1 (¹)	&acirc;	&#234;	e accent circonflexe (ê)
&ordm;	&#186;	Indicateur ordinal masculin (º)	&euml;	&#235;	e tréma (ë)
&raquo;	&#187;	Guillemet anglais fermant (")	&igrave;	&#236;	i accent grave (ì)
&frac14;	&#188;	Un quart (¼)	&iacute;	&#237;	i accent aigu (í)
&frac12;	&#189;	Un demi (½)	&icirc;	&#238;	i accent circonflexe (î)
&frac34;	&#190;	Trois quart (¾)	&iuml;	&#239;	i tréma (ï)
&iquest;	&#191;	Point d'interrogation inversé (¿)	&eth;	&#240;	eth (Islande : ð)
&Agrave;	&#192;	A accent grave (À)	&ntilde;	&#241;	n tilde (ñ)
&Aacute;	&#193;	A accent aigu (Á)	&ograve;	&#242;	o accent grave (ò)
&Acirc;	&#194;	A accent circonflexe (Â)	&oacute;	&#243;	o accent aigu (ó)
&Atilde;	&#195;	A tilde (Ã)	&ocirc;	&#244;	o accent circonflexe (ô)
&Auml;	&#196;	A tréma (Ä)	&otilde;	&#245;	o tilde (õ)
&Aring;	&#197;	A anneau (Å)	&ouml;	&#246;	o tréma (ö)
&Aelig;	&#198;	E dans l'A (Æ)	&divide;	&#247;	Symbole de la division (÷)
&Ccedil;	&#199;	C cédille (Ç)	&oslash;	&#248;	o barré (ø)
&Egrave;	&#200;	E accent grave (È)	&ugrave;	&#249;	u accent grave (ù)
&Eacute;	&#201;	E accent aigu (É)	&uacute;	&#250;	u accent aigu (ú)
&Ecirc;	&#202;	E accent circonflexe (Ê)	&uicirc;	&#251;	u accent circonflexe (û)
&Euml;	&#203;	E tréma (Ë)	&uuml;	&#252;	u tréma (ü)
&Igrave;	&#204;	I accent grave (Ì)	&yacute;	&#253;	y accent aigu (ý)
&Iacute;	&#205;	I accent aigu (Í)	&thorn;	&#254;	thorm (Islande : þ)
&Icirc;	&#206;	I accent circonflexe (Î)	&yuml;	&#255;	y tréma (ÿ)
&Iuml;	&#207;	I tréma (Ï)	&oelig;	&#338;	oe
&ETH;	&#208;	Eth majuscule (Islande : Ð)	&Oelig;	&#339;	OE
			&euro;	&#8364;	euro (€)

ANNEXE B : REFERENCES CSS

Le tableau ci-dessous indique les directives du langage CSS et les versions de leur introduction.

Les lignes surlignées en rouge interdisent leur implémentation vis à vis de leur compatibilité et de leur fiabilité.

Les lignes surlignées en orange demandent une autorisation précise avant toute utilisation du fait de leur problème de compatibilité.

Nom	Syntaxe	Description	Version CSS
Type : Texts & Fonts			
color	x { color:#rrggbb }	Sets the color of a text	CSS1
font	style="font:x "	A shorthand property for setting all of the properties for a font in one declaration	CSS1
font-family	style="font-family:x "	A prioritized list of font family names and/or generic family names for an element	CSS1
font-size	style="font-size:x "	Sets the size of a font	CSS1
font-size-adjust	style="font-size-adjust:x"	Specifies an aspect value for an element that will preserve the x-height of the first-choice font	CSS2
font-stretch	style="font-stretch:x"	Condenses or expands the current font-family	CSS2
font-style	style="font-style:x "	Sets the style of the font	CSS1
font-variant	style="font-variant:x"	Displays text in a small-caps font or a normal font	CSS1
font-weight	x{font-weight:X}	Sets the weight of a font	CSS1
letter-spacing	{ letter-spacing : sSpacing }	Increase or decrease the space between characters	CSS1
line-break	{ line-break : sBreak }	Sets or retrieves line-breaking rules for Japanese text.	This property is part of a proposed addition to CSS
line-height	{ line-height : sHeight }	Sets the distance between lines	CSS1
quotes		Sets the type of quotation marks	CSS2

Nom	Syntaxe	Description	Version CSS
ruby-align	{ ruby-align : sRubyAlign }	Sets or retrieves the position of the ruby text specified by the rt object.	CSS3
ruby-overhang	{ ruby-overhang : sRubyOverhang }	Sets or retrieves the position of the ruby text specified by the rt object.	CSS3
ruby-position	{ ruby-position : sRubyPlacement }	Sets or retrieves the position of the ruby text specified by the rt object.	CSS3
text-align	{ text-align : sAlign }	Sets or retrieves whether the text in the object is left-aligned, right-aligned, centered, or justified	CSS1
text-autospace	{ text-autospace : sIdeograph }	Sets or retrieves the autospacing and narrow space width adjustment of text.	This property is part of a proposed addition to CSS
text-decoration	{ text-decoration : sDecoration }	Sets or retrieves a value that indicates whether the text in the object has blink, line-through, overline, or underline decorations	CSS1
text-indent	{ text-indent : sIndent }	Sets or retrieves the indentation of the first line of text in the object	CSS1
text-justify	{ text-justify : sAlign }	Sets or retrieves the type of alignment used to justify text in the object.	This property is part of a proposed addition to CSS
text-kashida-space	{ text-kashida-space : vKashida }	Sets or retrieves the ratio of kashida expansion to white space expansion when justifying lines of text in the object.	This property is part of a proposed addition to CSS
text-transform	{ text-transform : sTransform }	Sets or retrieves the rendering of the text in the object	CSS1
text-underline-position	{ text-underline-position : sPosition }	Sets or retrieves the position of the underline decoration that is set through the textDecoration property of the object.	This property is a Microsoft extension to CSS
unicode-bidi	{ unicode-bidi : sEmbedLevel }	Sets or retrieves the level of embedding with respect to the bidirectional algorithm.	CSS2
white-space	{ white-space : sWrap }	Sets how white space inside an element is handled	CSS1
word-break	{ word-break : sBreak }	Sets or retrieves line-breaking behavior within words, particularly where multiple languages appear in the object.	This property is part of a proposed addition to CSS
word-spacing	{ word-spacing : sSpacing }	Sets or retrieves the amount of additional space between words in the object	CSS1
word-wrap	{ word-wrap : sWrap }	Sets or retrieves whether to break words when the content exceeds the boundaries of its container.	This property is a Microsoft extension to CSS

Nom	Syntaxe	Description	Version CSS
writing-mode	{ writing-mode : sFlow }	Sets or retrieves the direction and flow of the content in the object.	This property is part of a proposed addition to CSS
Type : Positionning			
bottom	{ bottom : sBottom }	Sets how far the bottom edge of an element is above/below the bottom edge of the parent element	CSS2
height	{ height : sHeight }	Sets the height of an element	CSS1
left	{ left : sPosition }	Sets how far the left edge of an element is to the right/left of the left edge of the parent element	CSS2
right	{ right : sPosition }	Sets how far the right edge of an element is to the left/right of the right edge of the parent element	CSS2
top	{ top : sTop }	Sets how far the top edge of an element is above/below the top edge of the parent element	CSS2
width	{ width : sWidth }	Sets the width of an element	CSS1
pixelBottom	object.style.pixelBottom [= iBottom]	Sets or retrieves the bottom position of the object	There is no public standard that applies to this property
pixelHeight	object.style.pixelHeight [= iHeight]	Sets or retrieves the height of the object	There is no public standard that applies to this property
pixelLeft	object.style.pixelLeft [= iLeft]	Sets or retrieves the left position of the object	There is no public standard that applies to this property
pixelRight	object.style.pixelRight [= iRight]	Sets or retrieves the right position of the object	There is no public standard that applies to this property
pixelTop	object.style.pixelTop [= iTop]	Sets or retrieves the top position of the object	There is no public standard that applies to this property
pixelWidth	object.style.pixelWidth [= iWidth]	Sets or retrieves the width of the object	There is no public standard that applies to this property
posBottom	object.style.posBottom [= iBottom]	Sets or retrieves the bottom position of the object in the units specified by the bottom attribute	There is no public standard that applies to this property
posHeight	object.style.posHeight [= iHeight]	Sets or retrieves the height of the object in the units specified by the height attribute	There is no public standard that applies to this property

Nom	Syntaxe	Description	Version CSS
posLeft	object.style.posLeft [= iLeft]	Sets or retrieves the left position of the object in the units specified by the left attribute	There is no public standard that applies to this property
posRight	object.style.posRight [= iRight]	Sets or retrieves the right position of the object in the units specified by the right attribute	There is no public standard that applies to this property
posTop	object.style.posTop [= iTop]	Sets or retrieves the top position of the object in the units specified by the top attribute	There is no public standard that applies to this property
posWidth	object.style.posWidth [= iWidth]	Sets or retrieves the width of the object in the units specified by the width attribute	There is no public standard that applies to this property
position	{ position : sPosition }	Places an element in a static, relative, absolute or fixed position	CSS2
z-index	{ z-index : vOrder }	Sets the stack order of an element	CSS2
Type : Border & Edges			
border	{ border : sBorder }	A shorthand property for setting all of the properties for the four borders in one declaration	CSS1
border-bottom	{ border-bottom : sBottom }	A shorthand property for setting all of the properties for the bottom border in one declaration	CSS1
border-bottom-color	{ border-bottom-color : sColor }	Sets the color of the bottom border	CSS2
border-bottom-style	{ border-bottom-style : sStyle }	Sets the style of the bottom border	CSS2
border-bottom-width	{ border-bottom-width : sWidth }	Sets the width of the bottom border	CSS1
border-color	{ border-color : sColor }	Sets the color of the four borders, can have from one to four colors	CSS1
border-left	{ border-left : sLeft }	A shorthand property for setting all of the properties for the left border in one declaration	CSS1
border-left-color	{ border-left-color : sColor }	Sets the color of the left border	CSS2
border-left-style	{ border-left-style : sStyle }	Sets the style of the left border	CSS2
border-left-width	{ border-left-width : sWidth }	Sets the width of the left border	CSS1
border-right	{ border-right : sRight }	A shorthand property for setting all of the properties for the right border in one declaration	CSS1
border-right-color	{ border-right-color : sColor }	Sets the color of the right border	CSS2
border-right-style	{ border-right-style : sStyle }	Sets the style of the right border	CSS2

Nom	Syntaxe	Description	Version CSS
border-right-width	{ border-right-width : sWidth }	Sets the width of the right border	CSS1
border-style	{ border-style : sStyle }	Sets the style of the four borders, can have from one to four styles	CSS1
border-top	{ border-top : sTop }	A shorthand property for setting all of the properties for the top border in one declaration	CSS1
border-top-color	{ border-top-color : sColor }	Sets the color of the top border	CSS2
border-top-style	{ border-top-style : sStyle }	Sets the style of the top border	CSS2
border-top-width	{ border-top-width : sWidth }	Sets the width of the top border	CSS1
border-width	{ border-width : sWidth }	A shorthand property for setting the width of the four borders in one declaration, can have from one to four values	CSS1
margin	{ margin : sMargin }	A shorthand property for setting the margin properties in one declaration	CSS1
margin-bottom	{ margin-bottom : sHeight }	Sets the bottom margin of an element	CSS1
margin-left	{ margin-left : sWidth }	Sets the left margin of an element	CSS1
margin-right	{ margin-right : sWidth }	Sets the right margin of an element	CSS1
margin-top	{ margin-top : sHeight }	Sets the top margin of an element	CSS1
outline	{ outline : x }	A shorthand property for setting all the outline properties in one declaration	CSS2
outline-color	{ outline-color : x }	Sets the color of the outline around an element	CSS2
outline-style	{ outline-style : x }	Sets the style of the outline around an element	CSS2
outline-width	{ outline-width : x }	Sets the width of the outline around an element	CSS2
padding	{ padding : sPadding }	Sets or retrieves the amount of space to insert between the object and its margin or, if there is a border, between the object and its border	CSS1
padding-bottom	{ padding-bottom : sPadding }	Sets the bottom padding of an element	CSS1
padding-left	{ padding-left : sPadding }	Sets the left padding of an element	CSS1
padding-right	{ padding-right : sPadding }	Sets the right padding of an element	CSS1
padding-top	{ padding-top : sPadding }	Sets the top padding of an element	CSS1
Type : Tables			
border-collapse	{ border-collapse : sCollapse }	Sets the border model of a table	CSS2
border-spacing	{border-spacing: x }	Sets the distance between the borders of adjacent cells (only for the "separated borders" model)	CSS2

Nom	Syntaxe	Description	Version CSS
		"separated borders" model)	
caption-side	{caption-side: x}	Sets the position of the caption according to the table	CSS2
empty-cells	{ empty-cells: x}	Sets whether cells with no visible content should have borders or not (only for the "separated borders" model)	CSS2
table-layout	{ table-layout : sLayout }	Sets the algorithm used to lay out the table	CSS2
Type : Lists			
list-style	{ list-style : sStyle }	A shorthand property for setting all of the properties for a list in one declaration	CSS1
list-style-image	{ list-style-image : sLocation }	Sets an image as the list-item marker	CSS1
list-style-position	{ list-style-position : sPosition }	Sets where the list-item marker is placed in the list	CSS1
list-style-type	{ list-style-type : sType }	Sets the type of the list-item marker	CSS1
Type : Background			
background	{ background : sBackground }	Sets or retrieves up to five separate background properties of the object	CSS1
background-attachment	{ background-attachment : sAttachment }	Sets whether a background image is fixed or scrolls with the rest of the page	CSS1
background-color	{ background-color : sColor }	Sets the background color of an element	CSS1
background-image	{ background-image : sLocation }	Sets an image as the background	CSS1
background-position	{ background-position : sPosition }	Sets the starting position of a background image	CSS1
background-position-x	{ background-position-x : iPositionX }	Sets or retrieves the x-coordinate of the background-position property	This property is a Microsoft extension to CSS
background-position-y	{ background-position-y : iPositionY }	Sets or retrieves the y-coordinate of the background-position property	This property is a Microsoft extension to CSS
background-repeat	{ background-repeat : sRepeat }	Sets if/how a background image will be repeated	CSS1
Type : Inline Displays & Layout			

Nom	Syntaxe	Description	Version CSS
Clear	{ clear : sClear }	Sets the sides of an element where other floating elements are not allowed	CSS1
Clip	{ clip : sClip }	Sets the shape of an element. The element is clipped into this shape, and displayed	CSS2
clip-bottom	[sBottom =] currentStyle.clipBottom	Retrieves the bottom coordinate of the object clipping region	There is no public standard that applies to this property
clip-left	[sLeft =] currentStyle.clipLeft	Retrieves the left coordinate of the object clipping region	There is no public standard that applies to this property
clip-right	[sRight =] currentStyle.clipRight	Retrieves the right coordinate of the object clipping region	There is no public standard that applies to this property
clip-top	[sTop =] currentStyle.clipTop	Retrieves the top coordinate of the object clipping region	There is no public standard that applies to this property
content	{ content: x }	Generates content in a document. Used with the :before and :after pseudo-elements	CSS2
counter-increment	{counter-increment: x}	Sets how much the counter increments on each occurrence of a selector	CSS2
counter-reset	{counter-reset x}	Sets the value the counter is set to on each occurrence of a selector	CSS2
cursor	{ cursor : sCursor }	Specifies the type of cursor to be displayed	CSS2
direction	{ direction : sDirection }	Sets the text direction	CSS2
display	{ display : sDisplay }	Sets how/if an element is displayed	CSS1
Filter	{ filter : sFilter }	Sets or retrieves the filter or collection of filters applied to the object.	This property is part of a proposed addition to CSS
Float	{ float : sFloat }	Sets where an image or a text will appear in another element	CSS1
layout-grid	{ layout-grid : sLayout }	Sets or retrieves the composite document grid properties that specify the layout of text characters.	This property is part of a proposed addition to CSS
layout-grid-char	{ layout-grid-char : sCharSize }	Sets or retrieves the size of the character grid used for rendering the text content of an element.	This property is part of a proposed addition to CSS
layout-grid-line	{ layout-grid-line : sLineSpace }	Sets or retrieves the gridline value used for rendering the text content of an element.	This property is part of a proposed addition to CSS

Nom	Syntaxe	Description	Version CSS
layout-grid-mode	{ layout-grid-mode : sMode }	Sets or retrieves whether the text layout grid uses two dimensions.	This property is part of a proposed addition to CSS
layout-grid-type	{ layout-grid-type : sType }	Sets or retrieves the type of grid used for rendering the text content of an element.	This property is part of a proposed addition to CSS
marker-offset	{marker-offset: x}	Cette propriété indique la distance entre les plus proches bords de bordures d'une boîte de marqueurs et est associée au bloc principal	CSS2
marks	marks{ X }	Sets what sort of marks should be rendered outside the page box	CSS2
max-height	x { max-height:X; }	Sets the maximum height of an element	CSS2
max-width	x { max-width:X; }	Sets the maximum width of an element	CSS2
min-height	x { min-height:X; }	Sets the minimum height of an element	CSS2
min-width	x { min-width:X; }	Sets the minimum width of an element	CSS2
overflow	{ overflow : sOverflow }	Sets what happens if the content of an element overflow its area	CSS2
overflow-x	{ overflow-x : sOverflow }	Sets or retrieves how to manage the content of the object when the content exceeds the width of the object.	This property is part of a proposed addition to CSS
overflow-y	{ overflow-y : sOverflow }	Sets or retrieves how to manage the content of the object when the content exceeds the height of the object	This property is part of a proposed addition to CSS
styleFloat	object.style.styleFloat [= sFloat]	Sets or retrieves on which side of the object the text will flow	CSS1
vertical-align	{ vertical-align : sAlign }	Sets the vertical alignment of an element	CSS1
visibility	{ visibility : sVisibility }	Sets if an element should be visible or invisible	CSS2
width	style="position:X; width:Y;"	Sets the width of an element	CSS1
zoom	{ zoom : vMagnification }	Sets or retrieves the magnification scale of the object.	This property is a Microsoft extension to CSS
Type : Printing			
orphans	x { orphans:X; }	Sets the minimum number of lines for a paragraph that must be left at the bottom of a page	CSS2
page		Sets a page type to use when displaying an element	CSS2
page-break-after	{ page-break-after : sBreak }	Sets the page-breaking behavior after an element	CSS2
page-break-before	{ page-break-before : sBreak }	Sets the page-breaking behavior before an element	CSS2

Nom	Syntaxe	Description	Version CSS
page-break-inside	{page-break-inside: x}	Sets the page-breaking behavior inside an element	CSS2
size	@page { size:x y }	Sets the orientation and size of a page	
widows	x { widows:X; }	Sets the minimum number of lines for a paragraph that must be left at the top of a page	CSS2
Type : Miscellaneous			
accelerator	{ ACCELERATOR : sIsAccessible }	Sets or retrieves a string that indicates whether the object contains an accelerator key	There is no public standard that applies to this property
behavior	<MARQUEE BEHAVIOR = sScroll... >	Sets or retrieves how the text scrolls in the marquee.	This property is part of a proposed addition to CSS
behavior	{ behavior : sBehavior }	Sets or retrieves the location of the Introduction to DHTML Behaviors	This property is part of a proposed addition to CSS
cssText	object.cssText [= sTxt]	Sets or retrieves the persisted representation of the style rule	There is no public standard that applies to this property
imeMode	{ ime-mode : sMode }	Sets or retrieves the state of an Input Method Editor (IME).	This property is part of a proposed addition to CSS
Type : Scrollbars			
scrollbar-3dlight-Color	{ scrollbar-3dlight-color : vColor }	Sets or retrieves the color of the top and left edges of the scroll box and scroll arrows of a scroll bar	This property is a Microsoft extension to CSS
scrollbar-arrow-color	{ scrollbar-arrow-color : vColor }	Sets or retrieves the color of the arrow elements of a scroll arrow	This property is a Microsoft extension to CSS
scrollbar-base-color	{ scrollbar-base-color : vColor }	Sets or retrieves the color of the main elements of a scroll bar, which include the scroll box, track, and scroll arrows	This property is a Microsoft extension to CSS
scrollbar-darkshadow-color	{ scrollbar-darkshadow-color : vColor }	Sets or retrieves the color of the gutter of a scroll bar	This property is a Microsoft extension to CSS
scrollbar-face-color	{ scrollbar-face-color : vColor }	Sets or retrieves the color of the scroll box and scroll arrows of a scroll bar	This property is a Microsoft extension to CSS
scrollbar-highlight-color	{ scrollbar-highlight-color : vColor }	Sets or retrieves the color of the top and left edges of the scroll box and scroll arrows of a scroll bar	This property is a Microsoft extension to CSS

Nom	Syntaxe	Description	Version CSS
scrollbar-shadow-color	{ scrollbar-shadow-color : vColor }	Sets or retrieves the color of the bottom and right edges of the scroll box and scroll arrows of a scroll bar	This property is a Microsoft extension to CSS
scrollbar-track-color	{ scrollbar-track-color : vColor }	Sets or retrieves the color of the track element of a scroll bar	There is no public standard that applies to this property
Type : Pseudo-Classes & Other Properties			
:active	[A]:active { sRules }	Sets the style of an a element when the link is engaged or active	CSS1
@charset	@charset sCharacterSet	Sets the character set for an external style sheet	CSS2
:first-letter	:first-letter { sRules }	Applies one or more styles to the first letter of the object	CSS1
:first-line	:first-line { sRules }	Applies one or more styles to the first line of the object	CSS1
@font-face	@font-face { sFace }	Sets a font to embed in the HTML document	CSS2
:hover	[A]:hover { sRules }	Sets the style of an a element when the user hovers the mouse pointer over the link	CSS2
@import	@import [url] (sUrl) ;	Imports an external style sheet	CSS1
!important	{ sRule!important }	Increases the weight or importance of a particular rule	CSS1
:link	[A]:link { sRules }	Sets the style of an a element when the link has not been visited recently	CSS1
@media	@media sMediaType { sRules }	Sets the media types for a set of rules in a styleSheet object	CSS2
@page	@page sPageSelector { sRules }	Sets the dimensions, orientation, and margins of a page box in a styleSheet	CSS1
:visited	[A]:visited { sRules }	Sets the style of an a element when the link has been visited recently	CSS1

ANNEXE C : REFERENCES XHTML

Attributs génériques

Le tableau ci-dessous indique les attributs génériques propres à l'ensemble des balises du langage XHTML (Core Attributes) à l'exception des balises suivantes : base, head, html, meta, param, script, style, et title.

Core Attribute	Description	Value
class	The class of the element	class_rule or style_rule
id	id for the element	id_name A unique
style	An inline style definition	style_definition
title	A text to display in a tool tip	tooltip_text

Attributs spécifiques à chaque balise

Le tableau ci-dessous indique d'une part les balise du langage XHTML et les attributs spécifiques associés et d'autre part les versions de leur introduction sur les principaux navigateurs du marché.

Les lignes surlignées en bleu indiquent les balises XHTML.

Les lignes surlignées en vert indiquent les attributs requis pour les balises.

Les lignes surlignées en rouge interdisent l'implémentation de ces attributs ou de ces balises vis à vis de leur compatibilité et de leur fiabilité.

Les lignes surlignées en orange demandent une autorisation précise avant toute implémentation de ces attributs car leur implémentation doit normalement être déportée dans une feuille de style ou être remplacée par une autre implémentation préconisée dans les normes.

Légende :

NN: cette colonne indique la version de Netscape à partir de laquelle la balise est supportée.

IE: même chose pour Internet Explorer.

DTD: indique la DTD(**XHTML 1.0 DTD)** qui autorise l'utilisation de la balise : S=Strict, T=Transitional et F=Frameset

Tag/Attribute	Description	Value	NN	IE	DTD
<!--...-->	**Defines a comment**		**3.0**	**3.0**	**STF**
<a>	**Defines an anchor**		**3.0**	**3.0**	**STF**
charset	Specifies the character encoding of the target URL	character_encoding			STF
coords	Specifies the coordinates appropriate to the shape attribute to define a region of an image for image maps	if shape="rect" then coords="left,top,right,bottom" if shape="circ" then coords="centerx,centery,radius" if shape="poly" then coords="x1,y1,x2,y2,..,xn,yn"			STF
href	The target URL of the link	URL			STF
hreflang	Specifies the base language of the target URL	language_code			STF
name	Names an anchor. Use this attribute to create a bookmark in a document. In future versions of XHTML the name attribute will be replaced by the id attribute!!	section_name			STF
rel	Specifies the relationship between the current document and the target URL	alternate designates stylesheet start next prev contents index glossary copyright chapter section subsection appendix help			STF

Tag/Attribute	Description	Value	NN	IE	DTD
		bookmark			
rev	Specifies the relationship between the target URL and the current document	alternate designates stylesheet start next prev contents index glossary copyright chapter section subsection appendix help bookmark			STF
shape	Defines the type of region to be defined for mapping in the current area tag. Used with the coords attribute.	rect rectangle circ circle poly polygon			STF
target	Where to open the target URL. _blank - the target URL will open in a new window _self - the target URL will open in the same frame as it was clicked _parent - the target URL will open in the parent frameset _top - the target URL will open in the full body of the window	_blank _parent _self _top			TF
type	Specifies the MIME (Multipurpose Internet Mail Extensions) type of the target URL	mime_type			STF
<abbr>	**Defines an abbreviation**		**6.2**		**STF**
title	Additional information to be displayed in tool tip	text			

Tag/Attribute	Description	Value	NN	IE	DTD
<acronym>	**Defines an acronym**		**6.2**	**4.0**	**STF**
title	Additional information to be displayed in tool tip	text			
<address>	**Defines an address element**		**4.0**	**4.0**	**STF**
<applet>	**Defines an applet**		**3.0**	**3.0**	**TF**
height	Defines the height of the applet	pixels			
width	Defines the width of the object	pixels			
align	Defines the text alignment around the applet	left right top bottom middle baseline texttop absmiddle absbottom			
alt	An alternate text to be displayed if the browser support applets but cannot run this applet	text			
archive	A URL to the applet when it is stored in a Java Archive or ZIP file	URL			
code	A URL that points to the class of the applet	URL			
codebase	Indicates the base URL of the applet if the code attribute is relative	URL			
hspace	Defines the horizontal spacing around the applet	pixels			
name	Defines a unique name for the applet (to use in scripts)	unique_name			
object	Defines the name of the resource that contains a serialized representation of the applet	name			
title	Additional information to be displayed in tool tip	text			
vspace	Defines the vertical spacing around the applet	pixels			
<area />	**Defines an area inside an image map**		**3.0**	**3.0**	**STF**
alt	Specifies an alternate text for the area	text			
coords	Specifies the coordinates for the clickable area	if shape="rect" then			

Tag/Attribute	Description	Value	NN	IE	DTD
		coords="left,top,right,bottom" if shape="circ" then coords="centerx,centery,radius" if shape="poly" then coords="x1,y1,x2,y2,..,xn,yn"			
href	Specifies the target URL of the area	URL			
nohref	Excludes an area from the image map	true false			
shape	Defines the shape of the area	rect rectangle circ circle poly polygon			
target	Where to open the target URL. _blank - the target URL will open in a new window _self - the target URL will open in the same frame as it was clicked _parent - the target URL will open in the parent frameset _top - the target URL will open in the full body of the window	_blank _parent _self _top			TF
<b>	**Defines bold text**		**3.0**	**3.0**	**STF**
<base />	**Defines a base URL for all the links in a page**		**3.0**	**3.0**	**STF**
href	Specifies the URL to use as the base URL for links in the page	URL			
target	Where to open all the links on the page. This attribute can be overridden by using the target attribute in each link. _blank - all the links will open in new windows _self - all the links will open in the same frame they where clicked _parent - all the links will open in the parent frameset _top - all the links will open in the full body of the window	_blank _parent _self _top			TF

Tag/Attribute	Description	Value	NN	IE	DTD
<basefont />	**Defines a base font**		**3.0**	**3.0**	**TF**
color	Specifies the text color. Deprecated. Use styles instead.	rgb(x,x,x) #xxxxxx colorname			
face	Specifies the font to use. Deprecated. Use styles instead.	list_of_fontnames			
size	Font size for font elements. Deprecated. Use styles instead.	default_text_size (a number from 1 to 7)			
<bdo>	**Defines the direction of text display**		**6.2**	**5.0**	**STF**
dir	Defines the text direction	ltr rtl			
<big>	**Defines big text**		**3.0**	**3.0**	**STF**
<blockquote>	**Defines a long quotation**		**3.0**	**3.0**	**STF**
cite	URL of the quote, if it is taken from the web	URL			
<body>	**Defines the body element**		**3.0**	**3.0**	**STF**
alink	Specifies the color of the active links in the document. Deprecated. Use styles instead.	rgb(x,x,x) #xxxxxx colorname			TF
background	An image to use as the background. Deprecated. Use styles instead.	file_name			TF
bgcolor	The background color of the document. Deprecated. Use styles instead.	rgb(x,x,x) #xxxxxx colorname			TF
link	Specifies the color of all the links in the document. Deprecated. Use styles instead.	rgb(x,x,x) #xxxxxx colorname			TF
text	Specifies the color of the text in the document. Deprecated. Use styles instead.	rgb(x,x,x) #xxxxxx colorname			TF
vlink	Specifies the color of the visited links in the document. Deprecated. Use styles instead.	rgb(x,x,x) #xxxxxx			TF

Tag/Attribute	Description	Value	NN	IE	DTD
		colorname			
** **	**Inserts a single line break**		**3.0**	**3.0**	**STF**
<button>	**Defines a push button**		**6.2**	**4.0**	**STF**
disabled	Disables the button	disabled			
name	Specifies a unique name for the button	button_name			
type	Defines the type of button	button reset submit			
value	Specifies an initial value for the button. The value can be changed by a script	some_value			
<caption>	**Defines a table caption**		**3.0**	**3.0**	**STF**
align	How to align the caption. Deprecated. Use styles instead.	left right top bottom			TF
** **	**Defines centered text**		**3.0**	**3.0**	**STF**
<cite>	**Defines a citation**		**3.0**	**3.0**	**STF**
<code>	**Defines computer code text**		**3.0**	**3.0**	**STF**
<col>	**Defines attributes for table columns**			**3.0**	**STF**
align	Defines the horizontal alignment of the content in the table cell, in the column	right left center justify char			
char	Defines a character to use to align text on (use with align="char")	character			
charoff	Defines an alignment offset to the first character to align on, as set with char	pixels %			
span	Defines the number of columns the <col> should span	number			
valign	Defines the vertical alignment of the content in the table cell, in the column	top			

Tag/Attribute	Description	Value	NN	IE	DTD
		middle bottom baseline			
width	Defines the width of the column. Note: Overrides the width set in <colgroup>	% pixels relative_length			
<colgroup>	**Defines groups of table columns**			**3.0**	**STF**
align	Defines the horizontal alignment of the contents in the column group	right left center justify char			
char	Defines a character to use to align text on (use with align="char")	character			
charoff	Defines an alignment offset to the first character to align on, as set with char	pixels %			
span	Defines the number of columns the <colgroup> should span	number			
valign	Defines the vertical alignment of the contents in the column group	top middle bottom baseline			
width	Defines the width of the column group. Note: The width attribute can be overridden by settings in <col>!	% pixels relative_length			
<dd>	**Defines a definition description**		**3.0**	**3.0**	**STF**
<del>	**Defines deleted text**		**6.2**	**4.0**	**STF**
cite	Defines a URL to another document which explains why the text was deleted or inserted	URL			
datetime	Defines the date and time the text was deleted	YYYYMMDD			
<dir>	**Defines a directory list**		**3.0**	**3.0**	**TF**
compact	Deprecated. Use styles instead	compact_rendering			

Tag/Attribute	Description	Value	NN	IE	DTD
<dfn>	**Defines a definition term**			**3.0**	**STF**
** **	**Defines a section in a document**		**3.0**	**3.0**	**STF**
align	How to align the text in the div element. Deprecated. Use styles instead.	left right top bottom			TF
<dl>	**Defines a definition list**		**3.0**	**3.0**	**STF**
<dt>	**Defines a definition term**		**3.0**	**3.0**	**STF**
****	**Defines emphasized text**		**3.0**	**3.0**	**STF**
<fieldset>	**Defines a fieldset**		**6.2**	**4.0**	**STF**
<font>	**Defines the font face, size, and color of text**		**3.0**	**3.0**	**TF**
color	Defines the color of the text in the font element. Deprecated. Use styles instead	rgb(x,x,x) #xxxxxx colorname			
face	Defines the font of the text in the font element. Deprecated. Use styles instead	list_of_fontnames			
size	Defines the size of the text in the font element. Deprecated. Use styles instead	A number from 1 to 7. If basefont is specified you can specify a number from -6 to 6			
<form>	**Defines a form**		**3.0**	**3.0**	**STF**
action	URL	A URL that defines where to send the data when the submit button is pushed			
accept	list of contenttypes	A comma separated list of content types that the server that process this form will handle correctly			
accept-charset	charset_list	A comma separated list of possible character sets for the form data. The default value is			

Tag/Attribute	Description	Value	NN	IE	DTD
		"unknown"			
enctype	mimetype	The mime type used to encode the content of the form			
method	get post	The HTTP method for sending data to the action URL. Default is get. method="get": This method sends the form contents in the URL: URL?name=value&name=value. Note: If the form values contains non-ASCII characters or exceeds 100 characters you MUST use method="post". method="post": This method sends the form contents in the body of the request. Note: Most browsers are unable to bookmark post requests.			
name	form_name	Defines a unique name for the form			TF
target	_blank _self _parent _top	Where to open the target URL. _blank - the target URL will open in a new window _self - the target URL will open in the same frame as it was clicked _parent - the target URL will open in the parent frameset _top - the target URL will open in the full body of the window			TF

Tag/Attribute	Description	Value	NN	IE	DTD
<frame>	**Defines a sub window (a frame)**		**3.0**	**3.0**	**F**
frameborder	Specifies whether or not to display border around the frame	0 1			
longdesc	A URL to a long description of the frame contents. Use it for browsers that do not support frames	URL			
marginheight	Defines the top and bottom margins in the frame	pixels			
marginwidth	Defines the left and right margins in the frame	pixels			
name	Defines a unique name for the frame (to use in scripts)	frame_name			
noresize	When set to noresize the user cannot resize the frame	noresize			
scrolling	Determines scrollbar action	yes no auto			
src	Defines the URL of the file to show in the frame	URL			
<frameset>	**Defines a set of frames**		**3.0**	**3.0**	**F**
cols	Defines the number and size of columns in a frameset	pixels % *			
rows	Defines the number and size of rows in a frameset	pixels % *			
<h1> to <h6>	**Defines header 1 to header 6**		**3.0**	**3.0**	**STF**
align	Specifies the alignment of the text in the header. Deprecated. Use styles instead	left center right justify			TF
<head>	**Defines information about the document**		**3.0**	**3.0**	**STF**
profile	A space separated list of URL's that contains meta data information about the page	URL			
<hr />	**Defines a horizontal rule**		**3.0**	**3.0**	**STF**

Tag/Attribute	Description	Value	NN	IE	DTD
align	Specifies the alignment of the horizontal rule. Deprecated. Use styles instead	center left right			TF
noshade	When set to true the rule should render in a solid color, when set to false the rule should render in a two-color "groove". Deprecated. Use styles instead	true false			TF
size	Specifies the thickness (height) of the horizontal rule. Deprecated. Use styles instead	pixels %			TF
width	Specifies the width of the horizontal rule Deprecated. Use styles instead	pixels %			TF
<html>	**Defines an html document**		**3.0**	**3.0**	**STF**
xmlns	Defines the XML namespace attribute	http://www.w3.org/1999/xhtml			
<i>	**Defines italic text**		**3.0**	**3.0**	**STF**
<iframe>	**Defines an inline sub window (frame)**		**6.0**	**4.0**	**TF**
align	Specifies how to align the iframe according to the surrounding text	left right top middle bottom			
frameborder	Specifies whether or not to display a frame border	1 0			
height	Defines the height of the iframe	pixels %			
longdesc	A URL to a long description of the frame contents	URL			
marginheight	Defines the top and bottom margins of the iframe	pixels			
marginwidth	Defines the left and right margins of the iframe	pixels			
name	Specifies a unique name of the iframe (to use in scripts)	frame_name			
scrolling	Define scroll bars	yes no			

Tag/Attribute	Description	Value	NN	IE	DTD
		auto			
src	The URL of the document to show in the iframe	URL			
width	Defines the width of the iframe	pixels %			
<img />	**Defines an image**		**3.0**	**3.0**	**STF**
alt	Defines a short description of the image	text			TF
src	The URL of the image to display	URL			TF
align	Specifies how to align the image according to surrounding text. Deprecated. Use styles instead	top bottom middle left right			
border	Defines a border around an image. Deprecated. Use styles instead	pixels			TF
height	Defines the height of an image	pixels %			
hspace	Defines white space on the left and right side of the image. Deprecated. Use styles instead	pixels			
ismap	Defines the image as a server-side image map	URL			
longdesc	A URL to a document that contains a long description of the image	URL			
usemap	Defines the image as a client-side image map. Look at the <map> and <area> tags to figure out how it works	URL			
vspace	Defines white space on the top and bottom of the image. Deprecated. Use styles instead	pixels			TF
width	Sets the width of an image	pixels %			
<input />	**Defines an input field**		**3.0**	**3.0**	**STF**
accept	A comma-separated list of MIME types that indicates the MIME type of the file transfer. Note: Only used with type="file"	list_of_mime_types			
align	Defines the alignment of text following the image.	left right			TF

Tag/Attribute	Description	Value	NN	IE	DTD
	Note: Only used with type="image"	top texttop middle absmiddle baseline bottom absbottom			
alt	Defines an alternate text for the image. Note: Only used with type="image"	text			
checked	Indicates that the input element should be checked when it first loads. Note: Used with type="checkbox" and type="radio"	checked			
disabled	Disables the input element when it first loads so that the user can not write text in it, or select it. Note: Cannot be used with type="hidden"	disabled			
maxlength	Defines the maximum number of characters allowed in a text field. Note: Only used with type="text"	number			
name	Defines a unique name for the input element. Note: This attribute is required with type="button", type="checkbox", type="file", type="hidden", type="image", type="password", type="text", and type="radio"	field_name			
readonly	Indicates that the value of this field cannot be modified. Note: Only used with type="text"	readonly			
size	Defines the size of the input element. Note: Cannot be used with type="hidden"	number_of_char			
src	Defines the URL of the image to display. Note: Only used with type="image"	URL			
type	Indicates the type of the input element. The default value is "text" Note: This is not a required attribute, but we think you should include it. If omitted, IE 5.5 will still display a text field, but Netscape 4.7 will not.	button checkbox file hidden			

Tag/Attribute	Description	Value	NN	IE	DTD
		hidden image password radio reset submit text			
value	For buttons, reset buttons and submit buttons: Defines the text on the button. For image buttons: Defines the symbolic result of the field passed to a script. For checkboxes and radio buttons: Defines the result of the input element when clicked. The result is sent to the form's action URL. For hidden, password, and text fields: Defines the default value of the element. Note: Cannot be used with type="file" Note: This attribute is required with type="checkbox" and type="radio"	value			
<ins>	**Defines inserted text**		**6.2**	**4.0**	**STF**
cite	A URL to another document which explains why the text was inserted	URL			
datetime	Defines the date and time when the text was inserted	YYYYMMDD			
<isindex>			**3.0**	**3.0**	**TF**
<kbd>	**Defines keyboard text**		**3.0**	**3.0**	**STF**
<label>	**Defines a label for a form control**		**6.2**	**4.0**	**STF**
for	Defines which form element the label is for. Set to an ID of a form element. Note: If this attribute is not specified, the label is associated with its contents.	id_of_another_field			
<legend>	**Defines a title in a fieldset**		**6.2**	**4.0**	**STF**
** **	**Defines a list item**		**3.0**	**3.0**	**STF**
type	Specifies the type of the list. Deprecated. Use styles instead				TF
value	Deprecated. Use styles instead				TF
<link>	**Defines a resource reference**		**4.0**	**3.0**	**STF**

Tag/Attribute	Description	Value	NN	IE	DTD
charset	Defines the character encoding of the target URL. Default value is "ISO-8859-1"	charset			
href	The target URL of the resource	URL			
hreflang	Defines the base language of the target URL	language_code			
media	Specifies on what device the document will be displayed	all braille print projection screen speech			
rel	Defines the relationship between the current document and the targeted document	alternate appendix bookmark chapter contents copyright glossary help home index next prev section start stylesheet subsection			
rev	Defines the relationship between the targeted document and the current document	alternate appendix bookmark chapter contents copyright glossary			

Tag/Attribute	Description	Value	NN	IE	DTD
		help home index next prev section start stylesheet subsection			
target	Where to open the target URL. _blank - the target URL will open in a new window _self - the target URL will open in the same frame as it was clicked _parent - the target URL will open in the parent frameset _top - the target URL will open in the full body of the window	_blank _self _top _parent			TF
type	Specifies the MIME type of the target URL	MIME_type like: text/css text/javascript image/gif			
<map>	**Defines an image map**		**3.0**	**3.0**	**STF**
id	Defines a unique name for the map tag	unique_name			
name	Defines a unique name for the map tag (for backwards compability).	unique_name			
<menu>	**Defines a menu list**		**3.0**	**3.0**	**TF**
compact	Deprecated. Use styles instead	compact_rendering			
<meta>	**Defines meta information**		**3.0**	**3.0**	**STF**
content	Defines meta information to be associated with http-equiv or name	some_text			
http-equiv	Connects the content attribute to an HTTP header	content-type expires refresh set-cookie			

Tag/Attribute	Description	Value	NN	IE	DTD
name	Connects the content attribute to a name	author description keywords generator revised others			
scheme	Defines a format to be used to interpret the value of the content attribute	some_text			
<noframes>	**Defines a noframe section**		**3.0**	**3.0**	**TF**
<noscript>	**Defines a noscript section**		**3.0**	**3.0**	**STF**
<object>	**Defines an embedded object**		**6.2**	**3.0**	**STF**
align	Defines the text alignment around the object	left right top bottom			TF
archive	A space separated list of URL's to archives. The archives contains resources relevant to the object	URL			
border	Defines a border around the object	pixels			TF
classid	Defines a class ID value as set in the Windows Registry or a URL	class ID			
codebase	Defines where to find the code for the object	URL			
codetype	The internet media type of the code referred to by the classid attribute	MIME type			
data	Defines a URL that refers to the object's data	URL			
declare	Defines that the object should only be declared, not created or instantiated until needed	declare			
height	Defines the height of the object	pixels			
hspace	Defines the horizontal spacing around the object	pixels			TF
name	Defines a unique name for the object (to use in scripts)	unique_name			
standby	Defines a text to display while the object is loading	text			
type	Defines the MIME type of data specified in the data attribute	MIME_type			

Tag/Attribute	Description	Value	NN	IE	DTD
usemap	Specifies a URL of a client-side image map to be used with the object	URL			
vspace	Defines the vertical spacing around the object	pixels			TF
width	Defines the width of the object	left right top bottom			
** **	**Defines an ordered list**		**3.0**	**3.0**	**STF**
compact	Deprecated. Use styles instead	compact_rendering			TF
start	Specifies the number to start on. Deprecated. Use styles instead	start_on_number			TF
type	Specifies the type of the list. Deprecated. Use styles instead	A a I i 1			TF
<optgroup>	**Defines an option group**		**6.0**		**STF**
label	Defines the label for the option group	text_label			
disabled	Disables the option-group when it first loads	disabled			
<option>	**Defines an option in a drop-down list**		**3.0**	**3.0**	**STF**
disabled	Specifies that the option should be disabled when it first loads	disabled			
label	Defines a label to use when using <optgroup>	text			
selected	Specifies that the option should appear selected (will be displayed first in the list)	selected			
value	Defines the the value of the option to be sent to the server	text			
<p>	**Defines a paragraph**		**3.0**	**3.0**	**STF**
align	left right center justify				TF
<param>	**Defines a parameter for an object**		**3.0**	**3.0**	**STF**

Tag/Attribute	Description	Value	NN	IE	DTD
name	Defines a unique name for the parameter	unique_name			
type	Specifies the internet media type for the parameter	MIME type			
value	Specifies the value of the parameter	value			
valuetype	Specifies the MIME type of the value	data ref object			
<pre>	**Defines preformatted text**		**3.0**	**3.0**	**STF**
width	Defines the maximum number of characters per line (usually 40, 80, or 132)	number			TF
<q>	**Defines a short quotation**		**6.2**		**STF**
cite	Defines a citation for the quotation	citation			
<s>	**Defines strikethrough text**		**3.0**	**3.0**	**TF**
<samp>	**Defines sample computer code**		**3.0**	**3.0**	**STF**
<script>	**Defines a script**		**3.0**	**3.0**	**STF**
type	Indicates the MIME type of the script	text/ecmascript text/javascript text/jscript text/vbscript text/vbs text/xml			
charset	Defines the character encoding used in script	charset			
defer	Indicates that the script is not going to generate any document content. The browser can continue parsing and drawing the page	defer			
language	Specifies the scripting language. Deprecated. Use the type attribute instead.	javascript livescript vbscript other			TF
src	Defines a URL to a file that contains the script (instead of inserting the script into your HTML document, you can refer to a file that contains the script)	URL			

Tag/Attribute	Description	Value	NN	IE	DTD
<select>	**Defines a selectable list**		**3.0**	**3.0**	**STF**
disabled	When set, it disables the drop-down list	disabled			
multiple	When set, it specifies that multiple items can be selected at a time	multiple			
name	Defines a unique name for the drop-down list	unique_name			
size	Definer the number of visible items in the drop-down list	number			
<small>	**Defines small text**		**3.0**	**3.0**	**STF**
****	**Defines a section in a document**		**4.0**	**3.0**	**STF**
<strike>	**Defines strikethrough text**		**3.0**	**3.0**	**TF**
****	**Defines strong text**		**3.0**	**3.0**	**STF**
<style>	**Defines a style definition**		**4.0**	**3.0**	**STF**
type	Defines the content-type	text/css			
media	The destination medium for the style information	screen tty tv projection handheld print braille aural all			
<sub>	**Defines subscripted text**		**3.0**	**3.0**	**STF**
<sup>	**Defines superscripted text**		**3.0**	**3.0**	**STF**
<table>	**Defines a table**		**3.0**	**3.0**	**STF**
align	Aligns the table. Deprecated. Use styles instead.	left center right			TF
bgcolor	Specifies the background color of the table. Deprecated. Use styles instead.	rgb(x,x,x) #xxxxxx			TF

Tag/Attribute	Description	Value	NN	IE	DTD
		colorname			
border	Specifies the border width. Tip: Set border="0" to display tables with no borders!	pixels			
cellpadding	Specifies the space between the cell walls and contents	pixels %			
cellspacing	Specifies the space between cells	pixels %			
frame	Specifies how the outer borders should be displayed. Note: Must be used in conjunction with the "border" attribute!	void above below hsides lhs rhs vsides box border			
rules	Specifies the horizontal/vertical divider lines. Note: Must be used in conjunction with the "border" attribute!	none groups rows cols all			
summary	Specifies a summary of the table for speech-synthesizing/non-visual browsers	text			
width	Specifies the width of the table	% pixels			
<tbody>	**Defines a table body**		**6.2**	**4.0**	**STF**
align	Defines the text alignment in cells	right left center justify char			

Tag/Attribute	Description	Value	NN	IE	DTD
char	Specifies which character to align text on. Note: Only used if align="char"!	character			
charoff	Specifies the alignment offset to the first character to align on. Note: Only used if align="char"!	pixels %			
valign	Specifies the vertical text alignment in cells	top middle bottom baseline			
<td>	**Defines a table cell**		**3.0**	**3.0**	**STF**
abbr	Specifies an abbreviated version of the content in a cell	abbr_text			
align	Specifies the horizontal alignment of cell content	left right center justify char			
axis	Defines a name for a cell	category_names			
bgcolor	Specifies the background color of the table cell. Deprecated. Use styles instead.	rgb(x,x,x) #xxxxxx colorname			TF
char	Specifies which character to align text on. Note: Only used if align="char"!	character			
charoff	Specifies the alignment offset to the first character to align on. Note: Only used if align="char"!	pixels %			
colspan	Indicates the number of columns this cell should span	number			
headers	A space-separated list of cell IDs that supply header information for the cell. This attribute allows text-only browsers to render the header information for a given cell	header_cells'_id			
height	Specifies the height of the table cell. Deprecated. Use styles instead.	pixels			TF
nowrap	Whether to disable or enable automatic text wrapping in this cell. Deprecated. Use styles	nowrap			TF

Tag/Attribute	Description	Value	NN	IE	DTD
	instead.				
rowspan	Indicates the number of rows this cell should span	number			
scope	Specifies if this cell provides header information for the rest of the row that contains it (row), or for the rest of the column (col), or for the rest of the row group that contains it (rowgroup), or for the rest of the column group that contains it	col colgroup row rowgroup			
valign	Specifies the vertical alignment of cell content	top middle bottom baseline			
width	Specifies the width of the table cell. Deprecated. Use styles instead.	pixels %			TF
<textarea>	**Defines a text area**		**3.0**	**3.0**	**STF**
cols	Specifies the number of columns visible in the text-area	number			
rows	Specifies the number of rows visible in the text-area	number			
disabled	Disables the text-area when it is first displayed	disabled			
name	Specifies a name for the text-area	name_of_textarea			
readonly	Indicates that the user cannot modify the content in the text-area	readonly			
<tfoot>	**Defines a table footer**		**6.2**	**4.0**	**STF**
align	Defines the text alignment in cells	right left center justify char			
char	Specifies which character to align text on. Note: Only used if align="char"!	character			
charoff	Specifies the alignment offset to the first character to align on. Note: Only used if align="char"!	pixels %			

Tag/Attribute	Description	Value	NN	IE	DTD
valign	Specifies the vertical text alignment in cells	top middle bottom baseline			
<th>	**Defines a table header. Use tag <thead> instead**		**3.0**	**3.0**	**STF**
abbr	Specifies an abbreviated version of the content in a cell	abbr_text			
align	Specifies the horizontal alignment of cell content	left right center justify char			
axis	Defines a name for a cell	category_names			
bgcolor	Specifies the background color of the table cell. Deprecated. Use styles instead.	rgb(x,x,x) #xxxxxx colorname			TF
char	Specifies which character to align text on. Note: Only used if align="char"!	character			
charoff	Specifies the alignment offset to the first character to align on. Note: Only used if align="char"!	pixels %			
colspan	Indicates the number of columns this cell should span	number			
headers	A space-separated list of cell IDs that supply header information for the cell. This attribute allows text-only browsers to render the header information for a given cell	header_cells'_id			
height	Specifies the height of the table cell. Deprecated. Use styles instead.	pixels			TF
nowrap	Whether to disable or enable automatic text wrapping in this cell. Deprecated. Use styles instead.	nowrap			TF
rowspan	Indicates the number of rows this cell should span	number			
scope	Specifies if this cell provides header information for the rest of the row that contains it (row), or for the rest of the column (col), or for the rest of the row group that contains it (rowgroup), or for the rest of the column group that contains it	col colgroup row			

Tag/Attribute	Description	Value	NN	IE	DTD
	(rowgroup), or for the rest of the column group that contains it	rowgroup			
valign	Specifies the vertical alignment of cell content	top middle bottom baseline			
width	Specifies the width of the table cell. Deprecated. Use styles instead.	pixels %			TF
<thead>	**Defines a table header**		**6.2**	**4.0**	**STF**
align	Defines the text alignment in cells	right left center justify char			
char	Specifies which character to align text on. Note: Only used if align="char"!	character			
charoff	Specifies the alignment offset to the first character to align on. Note: Only used if align="char"!	pixels %			
valign	Specifies the vertical text alignment in cells	top middle bottom baseline			
<title>	**Defines the document title**		**3.0**	**3.0**	**STF**
<tr>	**Defines a table row**		**3.0**	**3.0**	**STF**
align	Defines the text alignment in cells	right left center justify char			
bgcolor	Specifies the background color of the table cell. Deprecated. Use styles instead.	rgb(x,x,x) #xxxxxx			TF

Tag/Attribute	Description	Value	NN	IE	DTD
		colorname			
char	Specifies which character to align text on. Note: Only used if align="char"!	character			
charoff	Specifies the alignment offset to the first character to align on. Note: Only used if align="char"!	pixels %			
valign	Specifies the vertical text alignment in cells	top middle bottom baseline			
<tt>	**Defines teletype text**		**3.0**	**3.0**	**STF**
<u>	Defines underlined text		3.0	3.0	TF
** **	**Defines an unordered list**		**3.0**	**3.0**	**STF**
compact	Deprecated. Use styles instead	compact_rendering			TF
type	Specifies the type of the list. Deprecated. Use styles instead	disc square circle			TF
<var>	**Defines a variable**		**3.0**	**3.0**	**STF**
<xmp>	**Defines preformatted text. Use <pre> instead**		**3.0**	**3.0**	

ANNEXE D : REFERENCES JSCRIPT

Le tableau ci-dessous indique les directives du langage JScript et les versions de leur introduction.

Les lignes surlignées en rouge interdisent leur implémentation vis à vis de leur compatibilité et de leur fiabilité.

Les lignes surlignées en orange demandent une autorisation précise avant toute utilisation du fait de leur problème de compatibilité.

1 JSCRIPT

Language Element	1.0	2.0	3.0	4.0	5.0	5.5
0...n Property						x
$1...$9 Properties			x			
abs Method	x					
acos Method	x					
ActiveXObject Object			x			
Addition Assignment Operator (+=)	x					
Addition Operator (+)	x					
anchor Method	x					
apply Method						x
arguments Object	x					
arguments Property		x				
Array Object		x				
asin Method	x					
Assignment Operator (=)	x					
atan Method	x					
atan2 Method	x					
atEnd Method			x			
big Method	x					
Bitwise AND Assignment Operator (&=)	x					
Bitwise AND Operator (&)	x					
Bitwise Left Shift Operator (<<)	x					
Bitwise NOT Operator (~)	x					
Bitwise OR Assignment Operator (|=)	x					
Bitwise OR Operator (|)	x					
Bitwise Right Shift Operator (>>)	x					
Bitwise XOR Assignment Operator (^=)	x					
Bitwise XOR Operator (^)	x					
blink Method	x					
bold Method	x					
Boolean Object		x				

Language Element	1.0	2.0	3.0	4.0	5.0	5.5
break Statement	x					
call Method						x
callee Property						x
caller Property		x				
catch Statement					x	
@cc_on Statement			x			
ceil Method	x					
charAt Method	x					
charCodeAt Method						x
Comma Operator (,)	x					
// (Single-line Comment Statement)	x					
/*..*/ (Multiline Comment Statement)	x					
Comparison Operators	x					
compile Method			x			
concat Method (Array)			x			
concat Method (String)			x			
Conditional Compilation			x			
Conditional Compilation Variables			x			
Conditional (Ternary) Operator (?:)	x					
constructor Property		x				
continue Statement	x					
cos Method	x					
Data Type Conversion			x			
Date Object	x					
Debug Object			x			
debugger Statement			x			
decodeURI Method						x
DecodeURIComponent Method						x
Decrement Operator (--)	x					
delete Operator			x			
description Property					x	
dimensions Method			x			
Division Assignment Operator (/=)	x					
Division Operator (/)	x					
do...while Statement			x			
E Property	x					
encodeURI Method						x
encodeURI Component Method						x
Enumerator Object			x			
Equality Operator (==)	x					

Language Element	1.0	2.0	3.0	4.0	5.0	5.5
Error Object					x	
escape Method	x					
eval Method	x					
exec Method			x			
exp Method	x					
finally Statement					x	
fixed Method	x					
floor Method	x					
fontcolor Method	x					
fontsize Method	x					
for Statement	x					
for...in Statement					x	
fromCharCode Method			x			
Function Object		x				
function Statement	x					
getDate Method	x					
getDay Method	x					
getFullYear Method			x			
getHours Method	x					
getItem Method			x			
getMilliseconds Method			x			
getMinutes Method	x					
getMonth Method	x					
GetObject Function			x			
getSeconds Method	x					
getTime Method	x					
getTimezoneOffset Method	x					
getUTCDate Method			x			
getUTCDay Method			x			
getUTCFullYear Method			x			
getUTCHours Method			x			
getUTCMilliseconds Method			x			
getUTCMinutes Method			x			
getUTCMonth Method			x			
GetUTCSeconds Method			x			
getVarDate Method			x			
getYear Method	x					
Global Object			x			
global Property						x
Greater than Operator (>)	x					

Language Element	1.0	2.0	3.0	4.0	5.0	5.5
Greater than or equal to Operator (>=)	x					
hasOwnProperty Method						x
Identity Operator (===)	x					
@if Statement			x			
if...else Statement	x					
ignoreCase Property						x
In Operator	x					
Increment Operator (++)	x					
index Property			x			
indexOf Method	x					
Inequality Operator (!=)	x					
Infinity Property			x			
input Property ($_)			x			
instanceof Operator					x	
isFinite Method			x			
isNaN Method	x					
IsPrototypeOf Method						x
italics Method	x					
item Method			x			
join Method		x				
Labeled Statement			x			
lastIndex Property			x			
lastIndexOf Method	x					
lastMatch Property ($&)						x
lastParen Property ($+)						x
lbound Method			x			
leftContext Property ($')						x
Left Shift Assignment Operator (<<=)	x					
length Property (Arguments)						x
length Property (Array)		x				
length Property (Function)		x				
length Property (String)	x					
Less than Operator (<)	x					
Less than or equal to Operator (<=)	x					
link Method	x					
LN2 Property	x					
LN10 Property	x					
localeCompare Method						x
log Method	x					
LOG2E Property	x					

Language Element	1.0	2.0	3.0	4.0	5.0	5.5
LOG10E Property	x					
Logical AND Operator (&&)	x					
Logical NOT Operator (!)	x					
Logical OR Operator (||)	x					
match Method			x			
Math Object	x					
max Method	x					
MAX_VALUE Property		x				
message Property						x
min Method	x					
MIN_VALUE Property		x				
Modulus Assignment Operator (%=)	x					
Modulus Operator (%)	x					
moveFirst Method			x			
moveNext Method			x			
multiline Property						x
Multiplication Assignment Operator (*=)	x					
Multiplication Operator (*)	x					
name Property						x
NaN Property (Global)			x			
NaN Property (Number)		x				
NEGATIVE_INFINITY Property		x				
new Operator	x					
Nonidentity Operator (!==)	x					
Number Object		x				
number Property					x	
Object Object			x			
Operator Precedence	x					
parse Method	x					
parseFloat Method	x					
parseInt Method	x					
PI Property	x					
pop Method						x
POSITIVE_INFINITY Property		x				
pow Method	x					
prototype Property		x				
propertyIsEnumerable Property						x
push Method						x
random Method	x					
RegExp Object			x			

Language Element	1.0	2.0	3.0	4.0	5.0	5.5
Regular Expression Object			x			
Regular Expression Syntax			x			
replace Method	x					
return Statement	x					
reverse Method		x				
rightContext Property ($')						x
Right Shift Assignment Operator (>>=)	x					
round Method	x					
ScriptEngine Function		x				
ScriptEngineBuildVersion Function		x				
ScriptEngineMajorVersion Function		x				
ScriptEngineMinorVersion Function		x				
search Method			x			
@set Statement			x			
setDate Method	x					
setFullYear Method			x			
setHours Method	x					
setMilliseconds Method			x			
setMinutes Method	x					
setMonth Method	x					
setSeconds Method	x					
setTime Method	x					
setUTCDate Method			x			
setUTCFullYear Method			x			
setUTCHours Method			x			
setUTCMilliseconds Method			x			
setUTCMinutes Method			x			
setUTCMonth Method			x			
setUTCSeconds Method			x			
setYear Method	x					
shift Method						x
sin Method	x					
slice Method (Array)			x			
slice Method (String)			x			
small Method	x					
sort Method		x				
source Property			x			
splice Method						x
split Method			x			
sqrt Method	x					

Language Element	1.0	2.0	3.0	4.0	5.0	5.5
SQRT1_2 Property	x					
SQRT2 Property	x					
strike Method	x					
String Object	x					
sub Method	x					
substr Method			x			
substring Method	x					
Subtraction Assignment Operator (-=)	x					
Subtraction Operator (-)	x					
sup Method	x					
switch Statement			x			
tan Method	x					
test Method			x			
this Statement	x					
throw Statement					x	
toArray Method			x			
toDateString Method						x
toExponential Method						x
toFixed Method						x
toGMTString Method	x					
toLocaleDateString Method						x
toLocaleLowercase Method						x
toLocaleString Method	x					
toLocaleTimeString Method						x
toLocaleUppercase Method						x
toLowerCase Method	x					
toPrecision Method						x
toString Method		x				
toTimeString Method						x
toUpperCase Method	x					
toUTCString Method			x			
try Statement					x	
typeof Operator	x					
ubound Method			x			
Unary Negation Operator (-)	x					
undefined Property						x
unescape Method	x					
unshift Method						x
Unsigned Right Shift Assignment Operator (>>>=)	x					

Language Element	1.0	2.0	3.0	4.0	5.0	5.5
Unsigned Right Shift Operator (>>>)	x					
UTC Method	x					
valueOf Method		x				
var Statement	x					
VBArray Object			x			
void Operator		x				
while Statement	x					
with Statement	x					
write Method			x			
writeln Method			x			

2 JSCRIPT VS ECMASCRIPT

Le tableau ci-dessous indique les directives respectant les préconisations de ECMA :

Category	Feature/Keyword
Array Handling	Array join, length, reverse, sort
Assignments	Assign (=) Compound Assign (OP=)
Booleans	Boolean
Comments	/*...*/ or //
Constants/Literals	NaN null true, false Infinity undefined
Control flow	Break continue for for...in if...else return while

Category	Feature/Keyword
Dates and Time	Date getDate, getDay, getFullYear, getHours, getMilliseconds, getMinutes, getMonth, getSeconds, getTime, getTimezoneOffset, getYear, getUTCDate, getUTCDay, getUTCFullYear, getUTCHours, getUTCMilliseconds, getUTCMinutes, getUTCMonth, getUTCSeconds, setDate, setFullYear, setHours, setMilliseconds, setMinutes, setMonth, setSeconds, setTime, setYear, setUTCDate, setUTCFullYear, setUTCHours, setUTCMilliseconds, setUTCMinutes, setUTCMonth, setUTCSeconds, toGMTString, toLocaleString, toUTCString, parse, UTC
Declarations	Function new this var with
Function Creation	Function arguments, length
Global Methods	Global escape, unescape eval isFinite, isNaN parseInt, parseFloat
Math	Math abs, acos, asin, atan, atan2, ceil, cos, exp, floor, log, max, min, pow, random, round, sin, sqrt, tan, E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2, SQRT2
Numbers	Number MAX_VALUE, MIN_VALUE NaN NEGATIVE_INFINITY, POSITIVE_INFINITY
Object Creation	Object new constructor, prototype, instanceof, toString, valueOf

Category	Feature/Keyword
Operators	Addition (+), Subtraction (-) Modulus arithmetic (%) Multiplication (*), Division (/) Negation (-) Equality (==), Inequality (!=) Less Than (<), Less Than or Equal To (<=) Greater Than (>) Greater Than or Equal To (>=) Logical And(&&), Or (||), Not (!) Bitwise And (&), Or (|), Not (~), Xor (^) Bitwise Left Shift (<<), Shift Right (>>) Unsigned Shift Right (>>>) Conditional (?:) Comma (,) delete, typeof, void Decrement (--), Increment (++)
Objects	Array Boolean Date Function Global Math Number Object String
Strings	String charAt, charCodeAt, fromCharCode indexOf, lastIndexOf split toLowerCase, toUpperCase length

Le tableau ci-dessous indique les directives ne respectant pas les préconisations de ECMA :

Category	Feature/Keyword
Array Handling	concat, slice VBArray dimensions, getItem, lbound, toArray, ubound
Conditional Compilation	@cc_on @if Statement @set Statement Conditional Compilation Variables
Control flow	do...while Labeled switch
Dates and Time	getVarDate

Category	Feature/Keyword
Enumeration	Enumerator atEnd, item, moveFirst, moveNext
Error Handling	Error description, number throw, try...catch
Function Creation	caller
Operators	Identity (===), Nonidentity (!==)
Objects	Enumerator RegExp Regular Expression VBArray ActiveXObject GetObject
Regular Expressions and Pattern Matching	RegExp index, input, lastIndex, $1...$9, source, compile, exec, test Regular Expression Syntax
Script Engine Identification	ScriptEngine ScriptEngineBuildVersion ScriptEngineMajorVersion ScriptEngineMinorVersion
Strings	concat, slice match, replace, search anchor, big, blink, bold, fixed, fontcolor, fontsize, italics, link, small, strike, sub, sup

ANNEXE E : REFERENCES JAVASCRIPT

Le tableau ci-dessous indique les directives du langage JavaScript et les versions de leur introduction.

Les lignes surlignées en rouge interdisent leur implémentation sur Logitel Net vis à vis de leur compatibilité sur les navigateurs ciblés.

Les lignes surlignées en orange demandent une autorisation précise avant toute utilisation du fait de leur problème de compatibilité.

1 OBJETS/PROPRIETES/METHODES DU JAVASCRIPT

Objet	Type	Nom	Syntaxe	Description	Version JS
Globals	Core property	Infinity		A numeric value representing infinity.	JavaScript 1.3 (In previous versions, Infinity was defined only as a property of the Number object)
Globals	Core property	NaN		A value representing Not-A-Number.	JavaScript 1.3
Globals	Core property	undefined		The value undefined.	JavaScript 1.3
Globals	Core property	decodeURI	decodeURI(encodedURI)	Decodes a URI which has been encoded with encodeURI.	JavaScript 1.5
Globals	Core property	decodeURIComponent	decodeURIComponent(encodedURI)	Decodes a URI which has been encoded with encodeURIComponent	JavaScript 1.5
Globals	Core property	encodeURI	encodeURI(uri)	Computes a new version of a complete URI replacing each instance of certain characters with escape sequences representing the UTF-8 encoding of the characters.	JavaScript 1.5

Objet	Type	Nom	Syntaxe	Description	Version JS
Globals	Core property	encodeURIComponent	encodeURIComponent(uri)	Computes a new version of components of a URI replacing each instance of certain characters with escape sequences representing the UTF-8 encoding of the characters.	JavaScript 1.5
Globals	Core function	escape	escape("string")	Returns the hexadecimal encoding of an argument in the ISO Latin-1 character set; used to create strings to add to a URL.	JavaScript 1.0
Globals	Core function	eval	eval(string)	Evaluates a string of JavaScript code without reference to a particular object.	JavaScript 1.0 JavaScript 1.4: eval cannot be called indirectly
Globals	Core function	isFinite	isFinite(number)	Evaluates an argument to determine whether it is a finite number.	JavaScript 1.3
Globals	Core function	isNaN	isNaN(testValue)	Evaluates an argument to determine if it is not a number.	JavaScript 1.0: Unix only JavaScript 1.1: all platforms
Globals	Core function	Number	Number(obj)	Converts an object to a number.	JavaScript 1.2
Globals	Core function	parseFloat	parseFloat(string)	Parses a string argument and returns a floating-point number.	JavaScript 1.0: If the first character of the string specified in parseFloat(string) cannot be converted to a number, returns NaN on Solaris and Irix and 0 on all other platforms. JavaScript 1.1: Returns NaN on all platforms if the first character of the string specified in parseFloat(string) cannot be converted to a number.

Objet	Type	Nom	Syntaxe	Description	Version JS
Globals	Core function	parseInt	parseInt(string[, radix])	Parses a string argument and returns an integer.	JavaScript 1.0: If the first character of the string specified in parseInt(string) cannot be converted to a number, returns NaN on Solaris and Irix and 0 on all other platforms. JavaScript 1.1: Returns NaN on all platforms if the first character of the string specified in parseInt(string) cannot be converted to a number.
Globals	Core function	String	String(obj)	Converts an object to a string.	JavaScript 1.2
Globals	Core function	taint	taint(dataElementName)	Adds tainting to a data element or script.	JavaScript 1.1 JavaScript 1.2: removed
Globals	Core function	unescape	unescape(string)	Returns the ASCII string for the specified hexadecimal encoding value.	JavaScript 1.0
Globals	Core function	untaint	untaint([dataElementName])	Removes tainting from a data element or script.	JavaScript 1.1 JavaScript 1.2: removed
Array	Core object		new Array(arrayLength) new Array(element0, element1, ..., elementN)	Lets you work with arrays.	JavaScript 1.1 JavaScript 1.3: added toSource method; changed length property; changed push and splice methods.
Array	Property	constructor		Specifies the function that creates an object's prototype.	JavaScript 1.1
Array	Property	index		For an array created by a regular expression match, the zero-based index of the match in the string.	JavaScript 1.2
Array	Property	input		For an array created by a regular expression match, reflects the original string against which the regular expression was matched.	JavaScript 1.2
Array	Property	length		Reflects the number of elements in an array	JavaScript 1.1 JavaScript 1.3: length is an unsigned, 32-bit integer with a value less than 232.

Objet	Type	Nom	Syntaxe	Description	Version JS
Array	Property	prototype		Allows the addition of properties to all objects.	JavaScript 1.1
Array	Method	concat	concat(arrayName2, arrayName3, ..., arrayNameN)	Joins two arrays and returns a new array.	JavaScript 1.2
Array	Method	join	join(separator)	Joins all elements of an array into a string.	JavaScript 1.1
Array	Method	pop	pop()	Removes the last element from an array and returns that element.	JavaScript 1.2
Array	Method	push	push(element1, ..., elementN)	Adds one or more elements to the end of an array and returns the new length of the array.	JavaScript 1.2 JavaScript 1.3: push returns the new length of the array rather than the last element added to the array.
Array	Method	reverse	reverse()	Transposes the elements of an array: the first array element becomes the last and the last becomes the first.	JavaScript 1.1
Array	Method	shift	shift()	Removes the first element from an array and returns that element	JavaScript 1.2
Array	Method	slice	slice(begin[,end])	Extracts a section of an array and returns a new array.	JavaScript 1.2
Array	Method	sort	sort(compareFunction)	Sorts the elements of an array.	JavaScript 1.1 JavaScript 1.2: modified behavior.
Array	Method	splice	splice(index, howMany, [element1][, ..., elementN])	Adds and/or removes elements from an array.	JavaScript 1.2 JavaScript 1.3: returns an array containing the removed elements
Array	Method	toSource	toSource()	Returns an array literal representing the specified array; you can use this value to create a new array. Overrides the Object.toSource method.	JavaScript 1.3
Array	Method	toString	toString()	Returns a string representing the array and its elements. Overrides the Object.toString method.	JavaScript 1.1
Array	Method	unshift	arrayName.unshift(element1 ,..., elementN)	Adds one or more elements to the front of an array and returns the new length of the array.	JavaScript 1.2

Objet	Type	Nom	Syntaxe	Description	Version JS
Array	Method	valueOf	valueOf()	Returns the primitive value of the array. Overrides the Object.valueOf method.	JavaScript 1.1
Boolean	Core object		new Boolean(value)	The Boolean object is an object wrapper for a boolean value.	JavaScript 1.1 JavaScript 1.3: added toSource method
Boolean	Property	constructor		Specifies the function that creates an object's prototype.	JavaScript 1.1
Boolean	Property	prototype		Defines a property that is shared by all Boolean objects.	JavaScript 1.1
Boolean	Method	toSource	toSource()	Returns an object literal representing the specified Boolean object; you can use this value to create a new object. Overrides the Object.toSource method.	JavaScript 1.3
Boolean	Method	toString	toString()	Returns a string representing the specified object. Overrides the Object.toString method.	JavaScript 1.1
Boolean	Method	valueOf	valueOf()	Returns the primitive value of a Boolean object. Overrides the Object.valueOf method.	JavaScript 1.1
Date	Core object		new Date() new Date(milliseconds) new Date(dateString) new Date(yr_num, mo_num, day_num [, hr_num, min_num, sec_num, ms_num])	Lets you work with dates and times.	JavaScript 1.0 JavaScript 1.1: added prototype property JavaScript 1.3: removed platform dependencies to provide a uniform behavior across platforms; added ms_num parameter to Date constructor; added getFullYear, setFullYear, getMilliseconds, setMilliseconds, toSource, and UTC methods (such as getUTCDate and setUTCDate).
Date	Property	constructor		Specifies the function that creates an object's prototype.	JavaScript 1.1
Date	Property	prototype		Allows the addition of properties to a Date object.	JavaScript 1.1
Date	Method	getDate	getDate()	Returns the day of the month for the specified date according to local time.	JavaScript 1.0
Date	Method	getDay	getDay()	Returns the day of the week for the specified date according to local time.	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
Date	Method	getFullYear	getFullYear()	Returns the year of the specified date according to local time.	JavaScript 1.3
Date	Method	getHours	getHours()	Returns the hour in the specified date according to local time.	JavaScript 1.0
Date	Method	getMilliseconds	getMilliseconds()	Returns the milliseconds in the specified date according to local time.	JavaScript 1.3
Date	Method	getMinutes	getMinutes()	Returns the minutes in the specified date according to local time.	JavaScript 1.0
Date	Method	getMonth	getMonth()	Returns the month in the specified date according to local time.	JavaScript 1.0
Date	Method	getSeconds	getSeconds()	Returns the seconds in the specified date according to local time.	JavaScript 1.0
Date	Method	getTime	getTime()	Returns the numeric value corresponding to the time for the specified date according to local time.	JavaScript 1.0
Date	Method	getTimezoneOffset	getTimezoneOffset()	Returns the time-zone offset in minutes for the current locale.	JavaScript 1.0
Date	Method	getUTCDate	getUTCDate()	Returns the day (date) of the month in the specified date according to universal time.	JavaScript 1.3
Date	Method	getUTCDay	getUTCDay()	Returns the day of the week in the specified date according to universal time.	JavaScript 1.3
Date	Method	getUTCFullYear	getUTCFullYear()	Returns the year in the specified date according to universal time.	JavaScript 1.3
Date	Method	getUTCHours	getUTCHours()	Returns the hours in the specified date according to universal time.	JavaScript 1.3
Date	Method	getUTCMilliseconds	getUTCMilliSeconds()	Returns the milliseconds in the specified date according to universal time.	JavaScript 1.3
Date	Method	getUTCMinutes	getUTCMinutes()	Returns the minutes in the specified date according to universal time.	JavaScript 1.3
Date	Method	getUTCMonth	getUTCMonth()	Returns the month according in the specified date according to universal time.	JavaScript 1.3

Objet	Type	Nom	Syntaxe	Description	Version JS
Date	Method	getUTCSeconds	getUTCSeconds()	Returns the seconds in the specified date according to universal time.	JavaScript 1.3
Date	Method	getYear	getYear()	Returns the year in the specified date according to local time.	JavaScript 1.0 JavaScript 1.3: deprecated; also, getYear returns the year minus 1900 regardless of the year specified
Date	Method	parse	Date.parse(dateString)	Returns the number of milliseconds in a date string since January 1, 1970, 00:00:00, local time.	JavaScript 1.0
Date	Method	setDate	setDate(dayValue)	Sets the day of the month for a specified date according to local time.	JavaScript 1.0
Date	Method	setFullYear	setFullYear(yearValue[, monthValue, dayValue])	Sets the full year for a specified date according to local time.	JavaScript 1.3
Date	Method	setHours	setHours(hoursValue[, minutesValue, secondsValue, msValue])	Sets the hours for a specified date according to local time.	JavaScript 1.0 JavaScript 1.3: Added minutesValue, secondsValue, and msValue parameters
Date	Method	setMilliseconds	setMilliseconds(millisecondsValue)	Sets the milliseconds for a specified date according to local time.	JavaScript 1.3
Date	Method	setMinutes	setMinutes(minutesValue[, secondsValue, msValue])	Sets the minutes for a specified date according to local time.	JavaScript 1.0 JavaScript 1.3: Added secondsValue and msValue parameters
Date	Method	setMonth	setMonth(monthValue[, dayValue])	Sets the month for a specified date according to local time.	JavaScript 1.0 JavaScript 1.3: Added dayValue parameter
Date	Method	setSeconds	setSeconds(secondsValue[, msValue])	Sets the seconds for a specified date according to local time.	JavaScript 1.0 JavaScript 1.3: Added msValue parameter
Date	Method	setTime	setTime(timevalue)	Sets the value of a Date object according to local time.	JavaScript 1.0
Date	Method	setUTCDate	setUTCDate(dayValue)	Sets the day of the month for a specified date according to universal time.	JavaScript 1.3
Date	Method	setUTCFullYear	setUTCFullYear(yearValue[, monthValue, dayValue])	Sets the full year for a specified date according to universal time.	JavaScript 1.3

Objet	Type	Nom	Syntaxe	Description	Version JS
Date	Method	setUTCHours	setUTCHour(hoursValue[, minutesValue, secondsValue, msValue])	Sets the hour for a specified date according to universal time.	JavaScript 1.3
Date	Method	setUTCMilliseconds	setUTCMilliseconds(millisecondsValue)	Sets the milliseconds for a specified date according to universal time.	JavaScript 1.3
Date	Method	setUTCMinutes	setUTCMinutes(minutesValue[, secondsValue, msValue])	Sets the minutes for a specified date according to universal time.	JavaScript 1.3
Date	Method	setUTCMonth	setUTCMonth(monthValue[, dayValue])	Sets the month for a specified date according to universal time.	JavaScript 1.3
Date	Method	setUTCSeconds	setUTCSeconds(secondsValue[, msValue])	Sets the seconds for a specified date according to universal time.	JavaScript 1.3
Date	Method	setYear	setYear(yearValue)	Sets the year for a specified date according to local time.	JavaScript 1.0 Deprecated in JavaScript 1.3
Date	Method	toGMTString	toGMTString()	Converts a date to a string, using the Internet GMT conventions.	JavaScript 1.0 Deprecated in JavaScript 1.3
Date	Method	toLocaleString	toLocaleString()	Converts a date to a string, using the current locale's conventions.	JavaScript 1.0
Date	Method	toSource	toSource()	Returns an object literal representing the specified Date object; you can use this value to create a new object. Overrides the Object.toSource method.	JavaScript 1.3
Date	Method	toString	toString()	Returns a string representing the specified Date object. Overrides the Object.toString method.	JavaScript 1.1
Date	Method	toUTCString	toUTCString()	Converts a date to a string, using the universal time convention.	JavaScript 1.3
Date	Method	UTC	Date.UTC(year, month, day[, hrs, min, sec, ms])	Returns the number of milliseconds in a Date object since January 1, 1970, 00:00:00, universal time.	JavaScript 1.0 JavaScript 1.3: added ms parameter
Date	Method	valueOf	valueOf()	Returns the primitive value of a Date object. Overrides the Object.valueOf method.	JavaScript 1.1

Objet	Type	Nom	Syntaxe	Description	Version JS
Function	Core object			Specifies a string of JavaScript code to be compiled as a function.	JavaScript 1.1 JavaScript 1.2: added arity, arguments.callee properties; added ability to nest functions. JavaScript 1.3: added apply, call, and toSource methods; deprecated arguments.caller property. JavaScript 1.4: deprecated arguments, arguments.callee, arguments.length, and arity properties (arguments remains a variable local to a function rather than a property of Function).

Function	Property	arguments	new Function ([arg1[, arg2[, ... argN]],] functionBody)
			function name([param[, param[, ... param]]]) {
			statements
			}
				An array corresponding to the arguments passed to a function.	JavaScript 1.1
					JavaScript 1.2: added arguments.callee property.
					JavaScript 1.3: deprecated arguments.caller property; removed support for argument names and local variable names as properties of the arguments array.
					JavaScript 1.4: deprecated arguments, arguments.callee, and arguments.length as properties of Function; retained arguments as a local variable of a function and arguments.callee and arguments.length as properties of this variable.

Function	Property	arguments.callee		Specifies the function body of the currently executing function.	JavaScript 1.2 JavaScript 1.4: Deprecated callee as a property of Function.arguments, retained it as a property of a function's local arguments variable.
Function	Property	arguments.caller		Specifies the name of the function that invoked the currently executing function. (Deprecated)	JavaScript 1.1 Deprecated in JavaScript 1.3

Objet	Type	Nom	Syntaxe	Description	Version JS
Function	Property	arguments.length		Specifies the number of arguments passed to the function.	JavaScript 1.1 JavaScript 1.4: Deprecated length as a property of Function.arguments, retained it as a property of a function's local arguments variable.
Function	Property	arity		Specifies the number of arguments expected by the function.	JavaScript 1.2 Deprecated in JavaScript 1.4.
Function	Property	constructor		Specifies the function that creates an object's prototype.	JavaScript 1.1
Function	Property	length		Specifies the number of arguments expected by the function.	JavaScript 1.1
Function	Property	prototype		Allows the addition of properties to a Function object.	JavaScript 1.1
Function	Method	apply	apply(thisArg[, argArray])	Allows you to apply a method of another object in the context of a different object (the calling object).	JavaScript 1.3
Function	Method	call	call(thisArg[, arg1[, arg2[, ...]]])	Allows you to call (execute) a method of another object in the context of a different object (the calling object).	JavaScript 1.3
Function	Method	toSource	toSource()	Returns a string representing the source code of the function. Overrides the Object.toSource method.	JavaScript 1.3
Function	Method	toString	toString()	Returns a string representing the source code of the function. Overrides the Object.toString method.	JavaScript 1.1
Function	Method	valueOf	valueOf()	Returns a string representing the source code of the function. Overrides the Object.valueOf method.	JavaScript 1.1
Math				A built-in object that has properties and methods for mathematical constants and functions. For example, the Math object's PI property has the value of pi.	JavaScript 1.0
Math	Property	E		Euler's constant and the base of natural logarithms, approximately 2.718.	JavaScript 1.0
Math	Property	LN10		Natural logarithm of 10, approximately 2.302.	JavaScript 1.0
Math	Property	LN2		Natural logarithm of 2, approximately 0.693.	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
Math	Property	LOG10E		Base 10 logarithm of E (approximately 0.434).	JavaScript 1.0
Math	Property	LOG2E		Base 2 logarithm of E (approximately 1.442).	JavaScript 1.0
Math	Property	PI		Ratio of the circumference of a circle to its diameter, approximately 3.14159.	JavaScript 1.0
Math	Property	SQRT1_2		Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.	JavaScript 1.0
Math	Property	SQRT2		Square root of 2, approximately 1.414.	JavaScript 1.0
Math	Method	abs	abs(x)	Returns the absolute value of a number.	JavaScript 1.0
Math	Method	acos	acos(x)	Returns the arccosine (in radians) of a number.	JavaScript 1.0
Math	Method	asin	asin(x)	Returns the arcsine (in radians) of a number.	JavaScript 1.0
Math	Method	atan	atan(x)	Returns the arctangent (in radians) of a number.	JavaScript 1.0
Math	Method	atan2	atan2(y, x)	Returns the arctangent of the quotient of its arguments.	JavaScript 1.0
Math	Method	ceil	ceil(x)	Returns the smallest integer greater than or equal to a number.	JavaScript 1.0
Math	Method	cos	cos(x)	Returns the cosine of a number.	JavaScript 1.0
Math	Method	exp	exp(x)	Returns Enumber, where number is the argument, and E is Euler's constant, the base of the natural logarithms.	JavaScript 1.0
Math	Method	floor	floor(x)	Returns the largest integer less than or equal to a number.	JavaScript 1.0
Math	Method	log	log(x)	Returns the natural logarithm (base E) of a number.	JavaScript 1.0
Math	Method	max	max(x,y)	Returns the greater of two numbers.	JavaScript 1.0
Math	Method	min	min(x,y)	Returns the lesser of two numbers.	JavaScript 1.0
Math	Method	pow	pow(x,y)	Returns base to the exponent power, that is, baseexponent.	JavaScript 1.0
Math	Method	random	random()	Returns a pseudo-random number between 0 and 1.	JavaScript 1.0 : Unix only JavaScript 1.1 : all platforms
Math	Method	round	round(x)	Returns the value of a number rounded to the nearest integer.	JavaScript 1.0
Math	Method	sin	sin(x)	Returns the sine of a number.	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
Math	Method	sqrt	sqrt(x)	Returns the square root of a number.	JavaScript 1.0
Math	Method	tan	tan(x)	Returns the tangent of a number.	JavaScript 1.0
Number	Core object		new Number(value)	Lets you work with numeric values. The Number object is an object wrapper for primitive numeric values.	JavaScript 1.1 JavaScript 1.2: modified behavior of Number constructor. JavaScript 1.3: added toSource method. JavaScript 1.5: added toExponential, toFixed, and toPrecision methods.
Number	Property	constructor		Specifies the function that creates an object's prototype.	JavaScript 1.1
Number	Property	MAX_VALUE		The largest representable number.	JavaScript 1.1
Number	Property	MIN_VALUE		The smallest representable number.	JavaScript 1.1
Number	Property	NaN		Special "not a number" value.	JavaScript 1.1
Number	Property	NEGATIVE_INFINITY		Special value representing negative infinity; returned on overflow.	JavaScript 1.1
Number	Property	POSITIVE_INFINITY		Special value representing infinity; returned on overflow.	JavaScript 1.1
Number	Property	prototype		Allows the addition of properties to a Number object.	JavaScript 1.1
Number	Method	toExponential	toExponential([fractionDigits])	Returns a string representing the Number object in exponential notation.	JavaScript 1.5
Number	Method	toFixed	toFixed([fractionDigits])	Returns a string representing the Number object in fixed-point notation.	JavaScript 1.5
Number	Method	toPrecision	toPrecision([precision])	Returns a string representing the Number object to the specified precision.	JavaScript 1.5
Number	Method	toSource	toSource()	Returns an object literal representing the specified Number object; you can use this value to create a new object. Overrides the Object.toSource method.	JavaScript 1.3
Number	Method	toString	toString() toString(radix)	Returns a string representing the specified object. Overrides the Object.toString method.	JavaScript 1.1

Objet	Type	Nom	Syntaxe	Description	Version JS
Number	Method	valueOf	valueOf()	Returns the primitive value of the specified object. Overrides the Object.valueOf method.	JavaScript 1.1
Object	Core object		new Object()		JavaScript 1.0: toString method. JavaScript 1.1: added eval and valueOf methods; constructor property. JavaScript 1.2: deprecated eval method. JavaScript 1.3: added toSource method. JavaScript 1.4: removed eval method.
Object	Property	constructor		Specifies the function that creates an object's prototype.	JavaScript 1.1
Object	Property	prototype		Allows the addition of properties to all objects.	JavaScript 1.1
Object	Method	eval	eval(string)	Deprecated. Evaluates a string of JavaScript code in the context of the specified object.	JavaScript 1.1 JavaScript 1.2: deprecated as method of objects; retained as top-level function. JavaScript 1.4: removed as method of objects.
Object	Method	toSource	toSource()	Returns an object literal representing the specified object; you can use this value to create a new object.	JavaScript 1.3
Object	Method	toString	toString()	Returns a string representing the specified object.	JavaScript 1.0
Object	Method	unwatch	unwatch(prop)	Removes a watchpoint from a property of the object.	JavaScript 1.2
Object	Method	valueOf	valueOf()	Returns the primitive value of the specified object.	JavaScript 1.1
Object	Method	watch	watch(prop, handler)	Adds a watchpoint to a property of the object.	JavaScript 1.2

Objet	Type	Nom	Syntaxe	Description	Version JS
RegExp	Core object		/pattern/flags new RegExp("pattern"[, "flags"])	A regular expression object contains the pattern of a regular expression. It has properties and methods for using that regular expression to find and replace matches in strings.In addition to the properties of an individual regular expression object that you create using the RegExp constructor function, the predefined RegExp object has static properties that are set whenever any regular expression is used.	JavaScript 1.2 JavaScript 1.3: added toSource method JavaScript 1.5: added m flag, non-greedy modifier, non-capturing parentheses, lookahead assertions. ECMA 262, Edition 3
RegExp	Property	$'		See rightContext.	
RegExp	Property	$&		See lastMatch.	
RegExp	Property	$*		See multiline.	
RegExp	Property	$_		See input.	
RegExp	Property	$`		See leftContext.	
RegExp	Property	$+		See lastParen.	
RegExp	Property	$1, ..., $9		Parenthesized substring matches, if any.	JavaScript 1.2
RegExp	Property	constructor		Specifies the function that creates an object's prototype.	JavaScript 1.1
RegExp	Property	global		Whether or not to test the regular expression against all possible matches in a string, or only against the first.	JavaScript 1.2
RegExp	Property	ignoreCase		Whether or not to ignore case while attempting a match in a string.	JavaScript 1.2
RegExp	Property	input		The string against which a regular expression is matched.	JavaScript 1.2
RegExp	Property	lastIndex		The index at which to start the next match.	JavaScript 1.2
RegExp	Property	lastMatch		The last matched characters.	JavaScript 1.2
RegExp	Property	lastParen		The last parenthesized substring match, if any.	JavaScript 1.2
RegExp	Property	leftContext		The substring preceding the most recent match.	JavaScript 1.2
RegExp	Property	multiline		Whether or not to search in strings across multiple lines.	JavaScript 1.2
RegExp	Property	prototype		Allows the addition of properties to all objects.	JavaScript 1.1
RegExp	Property	rightContext		The substring following the most recent match.	JavaScript 1.2

Objet	Type	Nom	Syntaxe	Description	Version JS
RegExp	Property	source		The text of the pattern.	JavaScript 1.2
RegExp	Method	compile	regexp.compile(pattern[, flags])	Compiles a regular expression object.	JavaScript 1.2
RegExp	Method	exec	regexp.exec([str]) regexp([str])	Executes a search for a match in its string parameter.	JavaScript 1.2
RegExp	Method	test	regexp.test([str])	Tests for a match in its string parameter.	JavaScript 1.2
RegExp	Method	toSource	toSource()	Returns an object literal representing the specified object; you can use this value to create a new object. Overrides the Object.toSource method.	JavaScript 1.3
RegExp	Method	toString	toString()	Returns a string representing the specified object. Overrides the Object.toString method.	JavaScript 1.1
RegExp	Method	valueOf	valueOf()	Returns the primitive value of the specified object. Overrides the Object.valueOf method.	JavaScript 1.1
String	Core object		new String(string)	An object representing a series of characters in a string.	JavaScript 1.0: Create a String object only by quoting characters. JavaScript 1.1: added String constructor; added prototype property; added split method; added ability to pass strings among scripts in different windows or frames (in previous releases, you had to add an empty string to another window's string to refer to it) JavaScript 1.2: added concat, match, replace, search, slice, and substr methods. JavaScript 1.3: added toSource method; changed charCodeAt, fromCharCode, and replace methods
String	Property	constructor		Specifies the function that creates an object's prototype.	JavaScript 1.1

Objet	Type	Nom	Syntaxe	Description	Version JS
String	Property	length		Reflects the length of the string.	JavaScript 1.0
String	Property	prototype		Allows the addition of properties to a String object.	JavaScript 1.1
String	Method	anchor	anchor(nameAttribute)	Creates an HTML anchor that is used as a hypertext target.	JavaScript 1.0
String	Method	big	big()	Causes a string to be displayed in a big font as if it were in a BIG tag.	JavaScript 1.0
String	Method	blink	blink()	Causes a string to blink as if it were in a BLINK tag.	JavaScript 1.0
String	Method	bold	bold()	Causes a string to be displayed as if it were in a B tag.	JavaScript 1.0
String	Method	charAt	charAt(index)	Returns the character at the specified index.	JavaScript 1.0
String	Method	charCodeAt	charCodeAt([index])	Returns a number indicating the Unicode value of the character at the given index.	JavaScript 1.2 JavaScript 1.3: returns a Unicode value rather than an ISO-Latin-1 value
String	Method	concat	concat(string2, string3[, ..., stringN])	Combines the text of two strings and returns a new string.	JavaScript 1.2
String	Method	fixed	fixed()	Causes a string to be displayed in fixed-pitch font as if it were in a TT tag.	JavaScript 1.0
String	Method	fontcolor	fontcolor(color)	Causes a string to be displayed in the specified color as if it were in a <FONT COLOR=color> tag.	JavaScript 1.0
String	Method	fontsize	fontsize(size)	Causes a string to be displayed in the specified font size as if it were in a <FONT SIZE=size> tag.	JavaScript 1.0
String	Method	fromCharCode	fromCharCode(num1, ..., numN)	Returns a string created by using the specified sequence of Unicode values.	JavaScript 1.2 JavaScript 1.3: uses a Unicode value rather than an ISO-Latin-1 value
String	Method	indexOf	indexOf(searchValue[, fromIndex])	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.	JavaScript 1.0
String	Method	italics	italics()	Causes a string to be italic, as if it were in an I tag.	JavaScript 1.0
String	Method	lastIndexOf	lastIndexOf(searchValue[, fromIndex])	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
String	Method	link	link(hrefAttribute)	Creates an HTML hypertext link that requests another URL.	JavaScript 1.0
String	Method	match	match(regexp)	Used to match a regular expression against a string.	JavaScript 1.2
String	Method	replace	replace(regexp, newSubStr) replace(regexp, function)	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.	JavaScript 1.2 JavaScript 1.3: supports the nesting of a function in place of the second argument
String	Method	search	search(regexp)	Executes the search for a match between a regular expression and a specified string.	JavaScript 1.2
String	Method	slice	slice(beginslice[, endSlice])	Extracts a section of a string and returns a new string.	JavaScript 1.0
String	Method	small	small()	Causes a string to be displayed in a small font, as if it were in a SMALL tag.	JavaScript 1.0
String	Method	split	split([separator][, limit])	Splits a String object into an array of strings by separating the string into substrings.	JavaScript 1.1
String	Method	strike	strike()	Causes a string to be displayed as struck-out text, as if it were in a STRIKE tag.	JavaScript 1.0
String	Method	sub	sub()	Causes a string to be displayed as a subscript, as if it were in a SUB tag.	JavaScript 1.0
String	Method	substr	substr(start[, length])	Returns the characters in a string beginning at the specified location through the specified number of characters.	JavaScript 1.0
String	Method	substring	substring(indexA, indexB)	Returns the characters in a string between two indexes into the string.	JavaScript 1.0
String	Method	sup	sup()	Causes a string to be displayed as a superscript, as if it were in a SUP tag.	JavaScript 1.0
String	Method	toLowerCase	toLowerCase()	Returns the calling string value converted to lowercase.	JavaScript 1.0
String	Method	toSource	toSource()	Returns an object literal representing the specified object; you can use this value to create a new object. Overrides the Object.toSource method.	JavaScript 1.3
String	Method	toString	toString()	Returns a string representing the specified object. Overrides the Object.toString method.	JavaScript 1.1

Objet	Type	Nom	Syntaxe	Description	Version JS
String	Method	toUpperCase	toUpperCase()	Returns the calling string value converted to uppercase.	JavaScript 1.0
String	Method	valueOf	valueOf()	Returns the primitive value of the specified object. Overrides the Object.valueOf method.	JavaScript 1.1

2 « STATEMENTS » DU JAVASCRIPT

Nom	Syntaxe	Description	Version JS
break	break	Terminates the current while or for loop and transfers program control to the statement following the terminated loop.	JavaScript 1.0
comment	// comment text /* multiple line comment text */	Notations by the author to explain what a script does. Comments are ignored by the interpreter.	JavaScript 1.0
const	const constname [= value] [..., constname [= value]]	Declares a global constant, optionally initializing it to a value.	JavaScript 1.5
continue	continue	Terminates execution of the block of statements in a while or for loop, and continues execution of the loop with the next iteration.	JavaScript 1.0
do...while	do statements while (condition);	Executes the specified statements until the test condition evaluates to false. Statements execute at least once.	JavaScript 1.2
export	export name1, name2, ..., nameN export *	Allows a signed script to provide properties, functions, and objects to other signed or unsigned scripts.	JavaScript 1.2

for	for ([initial-expression]; [condition]; [increment-expression]) {
	  statements
	}
		Creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a block of statements executed in the loop.	JavaScript 1.0

for...in	for (variable in object) {
	  statements
	}
		Iterates a specified variable over all the properties of an object. For each distinct property, JavaScript executes the specified statements.	JavaScript 1.0

Nom	Syntaxe	Description	Version JS

function	function name([param] [, param] [..., param]) {
	  statements
	}
		Declares a function with the specified parameters. Acceptable parameters include strings, numbers, and objects.	JavaScript 1.0
			JavaScript 1.5: added conditional function declarations (Netscape extension).

if...else	if (condition) {
	  statements1
	}
	[else {
	  statements2
	}]
		Executes a set of statements if a specified condition is true. If the condition is false, another set of statements can be executed.	JavaScript 1.0

import	import objectName.name1, objectName.name2, ..., objectName.nameN import objectName.*	Allows a script to import properties, functions, and objects from a signed script that has exported the information.	JavaScript 1.2
label	label : statements	Provides an identifier that can be used with break or continue to indicate where the program should continue execution.	JavaScript 1.2
return	return expression	Specifies the value to be returned by a function.	JavaScript 1.0

switch	switch (expression){
	  case label :
	  statements;
	  break;
	  case label :
	  statements;
	  break;
	  ...
	  default : statements;
	}
		Allows a program to evaluate an expression and attempt to match the expression's value to a case label.	JavaScript 1.2

throw	throw expression;	Throws a user-defined exception.	JavaScript 1.4

Nom	Syntaxe	Description	Version JS

try...catch	try {
	statements
	}
	[catch (exception_var if expression)
	{statements}] . . .
	[catch (exception_var) {statements}]
	[finally {statements}]
		Marks a block of statements to try, and specifies a response should an exception be thrown.	JavaScript 1.4
			JavaScript 1.5: added multiple catch clauses (Netscape extension).

Var	var varname [= value] [..., varname [= value]]	Declares a variable, optionally initializing it to a value.	JavaScript 1.0

While	while (condition) {
	statements
	}
		Creates a loop that evaluates an expression, and if it is true, executes a block of statements. The loop then repeats, as long as the specified condition is true.	JavaScript 1.0

With	with (object){
	statements
	}
		Establishes the default object for a set of statements.	JavaScript 1.0

3 OPERATEURS DU JAVASCRIPT

Type	Nom	Description	Version JS
Arithmetic Operators	+	(Addition) Adds 2 numbers.	JavaScript 1.0
Arithmetic Operators	++	(Increment) Adds one to a variable representing a number (returning either the new or old value of the variable)	JavaScript 1.0
Arithmetic Operators	-	(Unary negation, subtraction) As a unary operator, negates the value of its argument. As a binary operator, subtracts 2 numbers.	JavaScript 1.0
Arithmetic Operators	--	(Decrement) Subtracts one from a variable representing a number (returning either the new or old value of the variable)	JavaScript 1.0
Arithmetic Operators	*	(Multiplication) Multiplies 2 numbers.	JavaScript 1.0
Arithmetic Operators	/	(Division) Divides 2 numbers.	JavaScript 1.0
Arithmetic Operators	%	(Modulus) Computes the integer remainder of dividing 2 numbers.	JavaScript 1.0

Type	Nom	Description	Version JS
String Operators	+	(String addition) Concatenates 2 strings.	JavaScript 1.0
String Operators	+=	Concatenates 2 strings and assigns the result to the first operand.	JavaScript 1.0
Logical Operators	&&	(Logical AND) Returns the first operand if it can be converted to false; otherwise, returns the second operand. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.	JavaScript 1.0
Logical Operators	||	(Logical OR) Returns the first operand if it can be converted to true; otherwise, returns the second operand. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns false.	JavaScript 1.0
Logical Operators	!	(Logical NOT) Returns false if its single operand can be converted to true; otherwise, returns true.	JavaScript 1.0
Bitwise Operators	&	(Bitwise AND) Returns a one in each bit position if bits of both operands are ones.	JavaScript 1.0
Bitwise Operators	^	(Bitwise XOR) Returns a one in a bit position if bits of one but not both operands are one.	JavaScript 1.0
Bitwise Operators	|	(Bitwise OR) Returns a one in a bit if bits of either operand is one.	JavaScript 1.0
Bitwise Operators	~	(Bitwise NOT) Flips the bits of its operand.	JavaScript 1.0
Bitwise Operators	<<	(Left shift) Shifts its first operand in binary representation the number of bits to the left specified in the second operand, shifting in zeros from the right.	JavaScript 1.0
Bitwise Operators	>>	(Sign-propagating right shift) Shifts the first operand in binary representation the number of bits to the right specified in the second operand, discarding bits shifted off.	JavaScript 1.0
Bitwise Operators	>>>	(Zero-fill right shift) Shifts the first operand in binary representation the number of bits to the right specified in the second operand, discarding bits shifted off, and shifting in zeros from the left.	JavaScript 1.0
Assignment Operators	=	Assigns the value of the second operand to the first operand.	JavaScript 1.0
Assignment Operators	+=	Adds 2 numbers and assigns the result to the first.	JavaScript 1.0
Assignment Operators	-=	Subtracts 2 numbers and assigns the result to the first.	JavaScript 1.0
Assignment Operators	*=	Multiplies 2 numbers and assigns the result to the first.	JavaScript 1.0
Assignment Operators	/=	Divides 2 numbers and assigns the result to the first.	JavaScript 1.0
Assignment Operators	%=	Computes the modulus of 2 numbers and assigns the result to the first.	JavaScript 1.0
Assignment Operators	&=	Performs a bitwise AND and assigns the result to the first operand.	JavaScript 1.0
Assignment Operators	^=	Performs a bitwise XOR and assigns the result to the first operand.	JavaScript 1.0
Assignment Operators	|=	Performs a bitwise OR and assigns the result to the first operand.	JavaScript 1.0

Type	Nom	Description	Version JS
Assignment Operators	<<=	Performs a left shift and assigns the result to the first operand.	JavaScript 1.0
Assignment Operators	>>=	Performs a sign-propagating right shift and assigns the result to the first operand.	JavaScript 1.0
Assignment Operators	>>>=	Performs a zero-fill right shift and assigns the result to the first operand.	JavaScript 1.0
Comparison Operators	==	Returns true if the operands are equal.	JavaScript 1.0
Comparison Operators	!=	Returns true if the operands are not equal.	JavaScript 1.0
Comparison Operators	===	Returns true if the operands are equal and of the same type.	JavaScript 1.3
Comparison Operators	!==	Returns true if the operands are not equal and/or not of the same type.	JavaScript 1.3
Comparison Operators	>	Returns true if the left operand is greater than the right operand.	JavaScript 1.0
Comparison Operators	>=	Returns true if the left operand is greater than or equal to the right operand.	JavaScript 1.0
Comparison Operators	<	Returns true if the left operand is less than the right operand.	JavaScript 1.0
Comparison Operators	<=	Returns true if the left operand is less than or equal to the right operand.	JavaScript 1.0
Special Operators	?:	Performs a simple "if...then...else"	JavaScript 1.0
Special Operators	,	Evaluates two expressions and returns the result of the second expression.	JavaScript 1.0
Special Operators	delete	Deletes an object, an object's property, or an element at a specified index in an array.	JavaScript 1.2
Special Operators	new	Creates an instance of a user-defined object type or of one of the built-in object types.	JavaScript 1.0
Special Operators	this	Keyword that you can use to refer to the current object.	JavaScript 1.0
Special Operators	typeof	Returns a string indicating the type of the unevaluated operand.	JavaScript 1.1
Special Operators	void	Specifies an expression to be evaluated without returning a value.	JavaScript 1.1

4 « EXPRESSIONS REGULIERES » DU JAVASCRIPT

Caractère	Description

Caractère	Description
\	For characters that are usually treated literally, indicates that the next character is special and not to be interpreted literally. For example, /b/ matches the character 'b'. By placing a backslash in front of b, that is by using /\b/, the character becomes special to mean match a word boundary. -or- For characters that are usually treated specially, indicates that the next character is not special and should be interpreted literally. For example, * is a special character that means 0 or more occurrences of the preceding character should be matched; for example, /a*/ means match 0 or more a's. To match * literally, precede the it with a backslash; for example, /a*/ matches 'a*'.
^	Matches beginning of input or line. For example, /^A/ does not match the 'A' in "an A," but does match it in "An A."
$	Matches end of input or line. For example, /t$/ does not match the 't' in "eater", but does match it in "eat"
*	Matches the preceding character 0 or more times. For example, /bo*/ matches 'boooo' in "A ghost booooed" and 'b' in "A bird warbled", but nothing in "A goat grunted".
+	Matches the preceding character 1 or more times. Equivalent to {1,}. For example, /a+/ matches the 'a' in "candy" and all the a's in "caaaaaaandy."
?	Matches the preceding character 0 or 1 time. For example, /e?le?/ matches the 'el' in "angel" and the 'le' in "angle."
.	(The decimal point) matches any single character except the newline character. For example, /.n/ matches 'an' and 'on' in "nay, an apple is on the tree", but not 'nay'.
(x)	Matches 'x' and remembers the match. For example, /(foo)/ matches and remembers 'foo' in "foo bar." The matched substring can be recalled from the resulting array's elements [1], ..., [n], or from the predefined RegExp object's properties $1, ..., $9.
x|y	Matches either 'x' or 'y'. For example, /green|red/ matches 'green' in "green apple" and 'red' in "red apple."
{n}	Where n is a positive integer. Matches exactly n occurrences of the preceding character. For example, /a{2}/ doesn't match the 'a' in "candy," but it matches all of the a's in "caandy," and the first two a's in "caaandy."
{n,}	Where n is a positive integer. Matches at least n occurrences of the preceding character. For example, /a{2,} doesn't match the 'a' in "candy", but matches all of the a's in "caandy" and in "caaaaaaandy."

Caractère	Description
{n,m}	Where n and m are positive integers. Matches at least n and at most m occurrences of the preceding character. For example, /a{1,3}/ matches nothing in "cndy", the 'a' in "candy," the first two a's in "caandy," and the first three a's in "caaaaaaandy" Notice that when matching "caaaaaaandy", the match is "aaa", even though the original string had more a's in it.
[xyz]	A character set. Matches any one of the enclosed characters. You can specify a range of characters by using a hyphen. For example, [abcd] is the same as [a-c]. They match the 'b' in "brisket" and the 'c' in "ache".
[^xyz]	A negated or complemented character set. That is, it matches anything that is not enclosed in the brackets. You can specify a range of characters by using a hyphen. For example, [^abc] is the same as [^a-c]. They initially match 'r' in "brisket" and 'h' in "chop."
[\b]	Matches a backspace. (Not to be confused with \b.)
\b	Matches a word boundary, such as a space. (Not to be confused with [\b].) For example, /\bn\w/ matches the 'no' in "noonday";/\wy\b/ matches the 'ly' in "possibly yesterday."
\B	Matches a non-word boundary. For example, /\w\Bn/ matches 'on' in "noonday", and /y\B\w/ matches 'ye' in "possibly yesterday."
\cX	Where X is a control character. Matches a control character in a string. For example, /\cM/ matches control-M in a string.
\d	Matches a digit character. Equivalent to [0-9]. For example, /\d/ or /[0-9]/ matches '2' in "B2 is the suite number."
\D	Matches any non-digit character. Equivalent to [^0-9]. For example, /\D/ or /[^0-9]/ matches 'B' in "B2 is the suite number."
\f	Matches a form-feed.
\n	Matches a linefeed.
\r	Matches a carriage return.
\s	Matches a single white space character, including space, tab, form feed, line feed. Equivalent to [\f\n\r\t\v]. For example, /\s\w*/ matches ' bar' in "foo bar."
\S	Matches a single character other than white space. Equivalent to [^ \f\n\r\t\v]. For example, /\S/\w* matches 'foo' in "foo bar."
\t	Matches a tab
\v	Matches a vertical tab.

Caractère	Description
\w	Matches any alphanumeric character including the underscore. Equivalent to [A-Za-z0-9_]. For example, /\w/ matches 'a' in "apple," '5' in "$5.28," and '3' in "3D."
\W	Matches any non-word character. Equivalent to [^A-Za-z0-9_]. For example, /\W/ or /[^$A-Za-z0-9_]/ matches '%' in "50%."
\n	Where n is a positive integer. A back reference to the last substring matching the n parenthetical in the regular expression (counting left parentheses). For example, /apple(,)\sorange\1/ matches 'apple, orange', in "apple, orange, cherry, peach." A more complete example follows this table.
\ooctal \xhex	Where \ooctal is an octal escape value or \xhex is a hexadecimal escape value. Allows you to embed ASCII codes into regular expressions.

ANNEXE F : REFERENCES DOM LEVEL 0

Le tableau ci-dessous indique les directives du langage JavaScript propres au DOM Level 0 et les versions de leur introduction.

Les lignes surlignées en rouge interdisent leur implémentation sur Logitel Net vis à vis de leur compatibilité et de leur fiabilité.

Les lignes surlignées en orange demandent une implémentation de détection ou des précautions d'utilisation du fait de leur problème de compatibilité (voir Norme de réalisation JS du présent document pour explications).

Objet	Type	Nom	Syntaxe	Description	Version JS
Anchor	Object			A place in a document that is the target of a hypertext link.	JavaScript 1.0
Anchor	Property	None			
Anchor	Method	None			
Button	Object			A push button on an HTML form.	JavaScript 1.0 JavaScript 1.1: added type property; added onBlur and onFocus event handlers; added blur and focus methods. JavaScript 1.2: added handleEvent method.
Button	Property	form		Specifies the form containing the Button object.	JavaScript 1.0
Button	Property	name		Reflects the NAME attribute.	JavaScript 1.0
Button	Property	type		Reflects the TYPE attribute.	JavaScript 1.1
Button	Property	value		Reflects the VALUE attribute.	JavaScript 1.0
Button	Method	blur	blur()	Removes focus from the button.	JavaScript 1.0
Button	Method	click	click()	Simulates a mouse-click on the button.	JavaScript 1.0
Button	Method	focus	focus()	Gives focus to the button.	JavaScript 1.0
Button	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2

Objet	Type	Nom	Syntaxe	Description	Version JS
Checkbox	Object			A checkbox on an HTML form. A checkbox is a toggle switch that lets the user set a value on or off.	JavaScript 1.0 JavaScript 1.1: added type property; added onBlur and onFocus event handlers; added blur and focus methods. JavaScript 1.2: added handleEvent method.
Checkbox	Property	checked		Boolean property that reflects the current state of the checkbox.	JavaScript 1.0
Checkbox	Property	defaultChecked		Boolean property that reflects the CHECKED attribute.	JavaScript 1.0
Checkbox	Property	form		Specifies the form containing the Checkbox object.	JavaScript 1.0
Checkbox	Property	name		Reflects the NAME attribute.	JavaScript 1.0
Checkbox	Property	type		Reflects the TYPE attribute.	JavaScript 1.1
Checkbox	Property	value		Reflects the TYPE attribute.	JavaScript 1.0
Checkbox	Method	blur	blur()	Removes focus from the checkbox.	JavaScript 1.0
Checkbox	Method	click	click()	Simulates a mouse-click on the checkbox.	JavaScript 1.0
Checkbox	Method	focus	focus()	Gives focus to the checkbox.	JavaScript 1.0
Checkbox	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
document	Object			Contains information about the current document, and provides methods for displaying HTML output to the user.	JavaScript 1.0 JavaScript 1.1: added onBlur and onFocus syntax; added applets, domain, embeds, forms, formName, images, and plugins properties. JavaScript 1.2: added layers property; added captureEvents, getSelection, handleEvent, releaseEvents, and routeEvent methods.
document	Property	alinkColor		A string that specifies the ALINK attribute.	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
document	Property	anchors	anchors[]	An array containing an entry for each anchor in the document.	JavaScript 1.0
document	Property	applets	applets[]	An array containing an entry for each applet in the document.	JavaScript 1.1
document	Property	bgColor		A string that specifies the BGCOLOR attribute.	JavaScript 1.0
document	Property	cookie		Specifies a cookie.	JavaScript 1.0
document	Property	domain		Specifies the domain name of the server that served a document.	JavaScript 1.1
document	Property	embeds	embeds[]	An array containing an entry for each plug-in in the document.	JavaScript 1.1
document	Property	fgColor		A string that specifies the TEXT attribute.	JavaScript 1.0
document	Property	formName		A separate property for each named form in the document.	JavaScript 1.1
document	Property	forms	forms[]	An array a containing an entry for each form in the document.	JavaScript 1.1
document	Property	images	images[]	An array containing an entry for each image in the document.	JavaScript 1.1
document	Property	lastModified		A string that specifies the date the document was last modified.	JavaScript 1.0
document	Property	layers	layers[]	Array containing an entry for each layer within the document.	JavaScript 1.2
document	Property	linkColor		A string that specifies the LINK attribute.	JavaScript 1.0
document	Property	links	links[]	An array containing an entry for each link in the document.	JavaScript 1.0
document	Property	plugins	plugins[]	An array containing an entry for each plug-in in the document.	JavaScript 1.1
document	Property	referrer		A string that specifies the URL of the calling document.	JavaScript 1.0
document	Property	title		A string that specifies the contents of the TITLE tag.	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
document	Property	URL		A string that specifies the complete URL of a document.	JavaScript 1.0
document	Property	vlinkColor		A string that specifies the VLINK attribute.	JavaScript 1.0
document	Method	captureEvents	captureEvents(eventType)	Sets the document to capture all events of the specified type.	JavaScript 1.2
document	Method	close	close()	Closes an output stream and forces data to display.	JavaScript 1.0
document	Method	getSelection	getSelection()	Returns a string containing the text of the current selection.	JavaScript 1.2
document	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
document	Method	open	open([mimeType, [replace]])	Opens a stream to collect the output of write or writeln methods.	JavaScript 1.0 JavaScript 1.1: added "replace" parameter; document.open() or document.open("text/html") clears the current document if it has finished loading
document	Method	releaseEvents	releaseEvents(eventType)	Sets the window or document to release captured events of the specified type, sending the event to objects further along the event hierarchy.	JavaScript 1.2
document	Method	routeEvent	routeEvent(event)	Passes a captured event along the normal event hierarchy.	JavaScript 1.2
document	Method	write	write(expr1[, ...,exprN])	Writes one or more HTML expressions to a document in the specified window.	JavaScript 1.0
document	Method	writeln	writeln(expr1[, ... exprN])	Writes one or more HTML expressions to a document in the specified window and follows them with a newline character.	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
FileUpload	Object			A file upload element on an HTML form. A file upload element lets the user supply a file as input.	JavaScript 1.0 JavaScript 1.1: added type property JavaScript 1.2: added handleEvent method.
FileUpload	Property	form		Specifies the form containing the FileUpload object.	JavaScript 1.0
FileUpload	Property	name		Reflects the NAME attribute.	JavaScript 1.0
FileUpload	Property	type		Reflects the TYPE attribute.	JavaScript 1.1
FileUpload	Property	value		Reflects the current value of the file upload element's field; this corresponds to the name of the file to upload.	JavaScript 1.0
FileUpload	Method	blur	blur()	Removes focus from the object.	JavaScript 1.0
FileUpload	Method	focus	focus()	Selects the input area of the file upload field.	JavaScript 1.0
FileUpload	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
FileUpload	Method	select	select()	Selects the input area of the file upload field.	JavaScript 1.0
Form	Object			Lets users input text and make choices from Form elements such as checkboxes, radio buttons, and selection lists. You can also use a form to post data to a server.	JavaScript 1.0 JavaScript 1.1: added reset method. JavaScript 1.2: added handleEvent method.
Form	Property	action		Reflects the ACTION attribute.	JavaScript 1.0
Form	Property	elements	elements[]	An array reflecting all the elements in a form.	JavaScript 1.0
Form	Property	encoding		Reflects the ENCTYPE attribute.	JavaScript 1.0
Form	Property	length		Reflects the number of elements on a form.	JavaScript 1.0
Form	Property	method		Reflects the METHOD attribute.	JavaScript 1.0
Form	Property	name		Reflects the NAME attribute.	JavaScript 1.0
Form	Property	target		Reflects the TARGET attribute.	JavaScript 1.0
Form	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
Form	Method	reset	reset()	Simulates a mouse click on a reset button for the calling form.	JavaScript 1.1

Objet	Type	Nom	Syntaxe	Description	Version JS
Form	Method	submit	submit()	Submits a form.	JavaScript 1.0
Hidden	Object			A Text object that is suppressed from form display on an HTML form. A Hidden object is used for passing name/value pairs when a form submits.	JavaScript 1.0 JavaScript 1.1: added type property
Hidden	Property	form		Specifies the form containing the Hidden object.	JavaScript 1.0
Hidden	Property	name		Reflects the NAME attribute.	JavaScript 1.0
Hidden	Property	type		Reflects the TYPE attribute.	JavaScript 1.1
Hidden	Property	value		Reflects the current value of the Hidden object.	JavaScript 1.0
History	Object			Contains an array of information on the URLs that the client has visited within a window. This information is stored in a history list and is accessible through the browser's Go menu.	JavaScript 1.0 JavaScript 1.1: added current, next, and previous properties.
History	Property	current		Retrieves the URL of the current history entry.	JavaScript 1.1
History	Property	length		Reflects the number of entries in the history list.	JavaScript 1.0
History	Property	next		Retrieves the URL of the next history entry.	JavaScript 1.1
History	Property	previous		Retrieves the URL of the previous history entry.	JavaScript 1.1
History	Method	back	back()	Loads the previous URL in the history list.	JavaScript 1.0
History	Method	forward	forward()	Loads the next URL in the history list.	JavaScript 1.0
History	Method	go	go(delta) go(location)	Loads a URL from the history list.	JavaScript 1.0
Image	Object		new Image([width,] [height])	An image on an HTML form.	JavaScript 1.1 JavaScript 1.2: added handleEvent method
Image	Property	border		Reflects the BORDER attribute.	JavaScript 1.1
Image	Property	complete		Boolean value indicating whether the web browser has completed its attempt to load the image.	JavaScript 1.1

Objet	Type	Nom	Syntaxe	Description	Version JS
Image	Property	height		Reflects the HEIGHT attribute.	JavaScript 1.1
Image	Property	hspace		Reflects the HSPACE attribute.	JavaScript 1.1
Image	Property	lowsrc		Reflects the LOWSRC attribute.	JavaScript 1.1
Image	Property	name		Reflects the NAME attribute.	JavaScript 1.1
Image	Property	src		Reflects the SRC attribute.	JavaScript 1.1
Image	Property	vspace		Reflects the VSPACE attribute.	JavaScript 1.1
Image	Property	width		Reflects the WIDTH attribute.	JavaScript 1.1
Image	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
Layer	Object			Corresponds to a layer in an HTML page and provides a means for manipulating that layer.	JavaScript 1.2
Layer	Property	above		The layer object above this one in z-order, among all layers in the document or the enclosing window object if this layer is topmost.	JavaScript 1.2
Layer	Property	background		The image to use as the background for the layer's canvas.	JavaScript 1.2
Layer	Property	below		The layer object below this one in z-order, among all layers in the document or null if this layer is at the bottom.	JavaScript 1.2
Layer	Property	bgColor		The color to use as a solid background color for the layer's canvas.	JavaScript 1.2
Layer	Property	clip.bottom		The bottom edge of the clipping rectangle (the part of the layer that is visible.)	JavaScript 1.2
Layer	Property	clip.height		The height of the clipping rectangle (the part of the layer that is visible.)	JavaScript 1.2
Layer	Property	clip.left		The left edge of the clipping rectangle (the part of the layer that is visible.)	JavaScript 1.2
Layer	Property	clip.right		The right edge of the clipping rectangle (the part of the layer that is visible.)	JavaScript 1.2

Objet	Type	Nom	Syntaxe	Description	Version JS
Layer	Property	clip.top		The top edge of the clipping rectangle (the part of the layer that is visible.)	JavaScript 1.2
Layer	Property	clip.width		The width of the clipping rectangle (the part of the layer that is visible.)	JavaScript 1.2
Layer	Property	document		The layer's associated document.	JavaScript 1.2
Layer	Property	left		The horizontal position of the layer's left edge, in pixels, relative to the origin of its parent layer.	JavaScript 1.2
Layer	Property	name		A string specifying the name assigned to the layer through the ID attribute in the LAYER tag.	JavaScript 1.2
Layer	Property	pageX		The horizontal position of the layer, in pixels, relative to the page.	JavaScript 1.2
Layer	Property	pageY		The vertical position of the layer, in pixels, relative to the page.	JavaScript 1.2
Layer	Property	parentLayer		The layer object that contains this layer, or the enclosing window object if this layer is not nested in another layer.	JavaScript 1.2
Layer	Property	siblingAbove		The layer object above this one in z-order, among all layers that share the same parent layer, or null if the layer has no sibling above.	JavaScript 1.2
Layer	Property	siblingBelow		The layer object below this one in z-order, among all layers that share the same parent layer, or null if layer is at the bottom.	JavaScript 1.2
Layer	Property	src		A string specifying the URL of the layer's content.	JavaScript 1.2
Layer	Property	top		The vertical position of the layer's top edge, in pixels, relative to the origin of its parent layer.	JavaScript 1.2
Layer	Property	visibility		Whether or not the layer is visible.	JavaScript 1.2
Layer	Property	zIndex		The relative z-order of this layer with respect to its siblings.	JavaScript 1.2
Layer	Method	captureEvents	captureEvents(eventType)	Sets the window or document to capture all events of the specified type.	JavaScript 1.2

Objet	Type	Nom	Syntaxe	Description	Version JS
Layer	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
Layer	Method	load	load(sourcestring, width)	Changes the source of a layer to the contents of the specified file, and simultaneously changes the width at which the layer's HTML contents will be wrapped.	JavaScript 1.2
Layer	Method	moveAbove	moveAbove(aLayer)	Stacks this layer above the layer specified in the argument, without changing either layer's horizontal or vertical position.	JavaScript 1.2
Layer	Method	moveBelow	moveBelow(aLayer)	Stacks this layer below the specified layer, without changing either layer's horizontal or vertical position.	JavaScript 1.2
Layer	Method	moveBy	moveBy(horizontal, vertical)	Changes the layer position by applying the specified deltas, measured in pixels.	JavaScript 1.2
Layer	Method	moveTo	moveTo(x-coordinate, y-coordinate)	Moves the top-left corner of the window to the specified screen coordinates.	JavaScript 1.2
Layer	Method	moveToAbsolute	moveToAbsolute(x, y)	Changes the layer position to the specified pixel coordinates within the page (instead of the containing layer.)	JavaScript 1.2
Layer	Method	releaseEvents	releaseEvents(eventType)	Sets the layer to release captured events of the specified type, sending the event to objects further along the event hierarchy.	JavaScript 1.2
Layer	Method	resizeBy	resizeBy(width, height)	Resizes the layer by the specified height and width values (in pixels).	JavaScript 1.2
Layer	Method	resizeTo	resizeTo(width, height)	Resizes the layer to have the specified height and width values (in pixels).	JavaScript 1.2
Layer	Method	routeEvent	routeEvent(event)	Passes a captured event along the normal event hierarchy.	JavaScript 1.2

Objet	Type	Nom	Syntaxe	Description	Version JS
Link	Object			A piece of text, an image, or an area of an image identified as a hypertext link. When the user clicks the link text, image, or area, the link hypertext reference is loaded into its target window. Area objects are a type of Link object.	JavaScript 1.0 JavaScript 1.1: added onMouseOut event handler; added Area objects; links array contains areas created with <AREA HREF="..."> JavaScript 1.2: added handleEvent method
Link	Property	hash		Specifies an anchor name in the URL.	JavaScript 1.0
Link	Property	host		Specifies the host and domain name, or IP address, of a network host.	JavaScript 1.0
Link	Property	hostname		Specifies the host:port portion of the URL.	JavaScript 1.0
Link	Property	href		Specifies the entire URL.	JavaScript 1.0
Link	Property	pathname		Specifies the URL-path portion of the URL.	JavaScript 1.0
Link	Property	port		Specifies the communications port that the server uses.	JavaScript 1.0
Link	Property	protocol		Specifies the beginning of the URL, including the colon.	JavaScript 1.0
Link	Property	search		Specifies a query string.	JavaScript 1.0
Link	Property	target		Reflects the TARGET attribute.	JavaScript 1.0
Link	Property	text		A string containing the content of the corresponding A tag.	JavaScript 1.2
Link	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
Location	Object			Contains information on the current URL.	JavaScript 1.0 JavaScript 1.1: added reload, replace methods
Location	Property	hash		Specifies an anchor name in the URL.	JavaScript 1.0
Location	Property	host		Specifies the host and domain name, or IP address, of a network host.	JavaScript 1.0
Location	Property	hostname		Specifies the host:port portion of the URL.	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
Location	Property	href		Specifies the entire URL.	JavaScript 1.0
Location	Property	pathname		Specifies the URL-path portion of the URL.	JavaScript 1.0
Location	Property	port		Specifies the communications port that the server uses.	JavaScript 1.0
Location	Property	protocol		Specifies the beginning of the URL, including the colon.	JavaScript 1.0
Location	Property	search		Specifies a query.	JavaScript 1.0
Location	Method	reload	reload([forceGet])	Forces a reload of the window's current document.	JavaScript 1.1
Location	Method	replace	replace(URL)	Loads the specified URL over the current history entry.	JavaScript 1.1
MimeType	Object			A MIME type (Multipart Internet Mail Extension) supported by the client.	JavaScript 1.1
MimeType	Property	description		A description of the MIME type.	JavaScript 1.1
MimeType	Property	enabledPlugin		Reference to the Plugin object configured for the MIME type.	JavaScript 1.1
MimeType	Property	suffixes		A string listing possible filename extensions for the MIME type, for example "mpeg, mpg, mpe, mpv, vbs, mpegv".	JavaScript 1.1
MimeType	Property	type		The name of the MIME type, for example "video/mpeg" or "audio/x-wav".	JavaScript 1.1
navigator	Object			Contains information about the version of Navigator in use.	JavaScript 1.0 JavaScript 1.1: added mimeTypes and plugins properties; added javaEnabled and taintEnabled methods. JavaScript 1.2: added language and platform properties; added preference method.

Objet	Type	Nom	Syntaxe	Description	Version JS
navigator	Property	appCodeName		Specifies the code name of the browser.	JavaScript 1.0
navigator	Property	appName		Specifies the name of the browser.	JavaScript 1.0
navigator	Property	appVersion		Specifies version information for the Navigator.	JavaScript 1.0
navigator	Property	cookieEnabled		Specifies if Navigator supported cookies	JavaScript 1.2
navigator	Property	language		Indicates the translation of the Navigator being used.	JavaScript 1.2
navigator	Property	mimeTypes		An array of all MIME types supported by the client.	JavaScript 1.1
navigator	Property	platform		Indicates the machine type for which the Navigator was compiled.	JavaScript 1.2
navigator	Property	plugins		An array of all plug-ins currently installed on the client.	JavaScript 1.1
navigator	Property	userAgent		Specifies the user-agent header.	JavaScript 1.0
navigator	Method	javaEnabled	javaEnabled()	Tests whether Java is enabled.	JavaScript 1.1
navigator	Method	plugins.refresh	navigator.plugins.refresh(true|false)	Makes newly installed plug-ins available and optionally reloads open documents that contain plug-ins.	JavaScript 1.0
navigator	Method	preference	preference(prefName[, setValue])	Allows a signed script to get and set certain Navigator preferences.	JavaScript 1.2
navigator	Method	taintEnabled	taintEnabled()	Specifies whether data tainting is enabled.	JavaScript 1.1 JavaScript 1.2: removed
Option	Object			An option in a selection list.	JavaScript 1.0 JavaScript 1.1: added defaultSelected property; text property can be changed to change the text of an option
Option	Property	defaultSelected		Specifies the initial selection state of the option	JavaScript 1.1
Option	Property	selected		Specifies the current selection state of the option	JavaScript 1.0
Option	Property	text		Specifies the text for the option	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
Option	Property	value		Specifies the value that is returned to the server when the option is selected and the form is submitted	JavaScript 1.0
Password	Object			A text field on an HTML form that conceals its value by displaying asterisks (*). When the user enters text into the field, asterisks (*) hide entries from view.	JavaScript 1.0 JavaScript 1.1: added type property; added onBlur and onFocus event handlers JavaScript 1.2: added handleEvent method.
Password	Property	defaultValue		Reflects the VALUE attribute.	JavaScript 1.0
Password	Property	form		Specifies the form containing the Password object.	JavaScript 1.0
Password	Property	name		Reflects the NAME attribute.	JavaScript 1.0
Password	Property	type		Reflects the TYPE attribute.	JavaScript 1.1
Password	Property	value		Reflects the current value of the Password object's field.	JavaScript 1.0
Password	Method	blur	blur()	Removes focus from the object.	JavaScript 1.0
Password	Method	focus	focus()	Gives focus to the object.	JavaScript 1.0
Password	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
Password	Method	select	select()	Selects the input area of the object.	JavaScript 1.0
Plugin	Object			A plug-in module installed on the client.	JavaScript 1.1
Plugin	Property	description		A description of the plug-in.	JavaScript 1.1
Plugin		filename		Name of the plug-in file on disk.	JavaScript 1.1
Plugin		length		Number of elements in the plug-in's array of MimeType objects.	JavaScript 1.1
Plugin		name		Name of the plug-in.	JavaScript 1.1

Objet	Type	Nom	Syntaxe	Description	Version JS
Radio	Object			An individual radio button in a set of radio buttons on an HTML form. The user can use a set of radio buttons to choose one item from a list.	JavaScript 1.0 JavaScript 1.1: added type property; added blur and focus methods. JavaScript 1.2: added handleEvent method.
Radio	Property	checked		Lets you programmatically select a radio button (property of the individual button).	JavaScript 1.0
Radio	Property	defaultChecked		Reflects the CHECKED attribute (property of the individual button).	JavaScript 1.0
Radio	Property	form		Specifies the form containing the Radio object (property of the array of buttons).	JavaScript 1.0
Radio	Property	name		Reflects the NAME attribute (property of the array of buttons).	JavaScript 1.0
Radio	Property	type		Reflects the TYPE attribute (property of the array of buttons).	JavaScript 1.1
Radio	Property	value		Reflects the VALUE attribute (property of the array of buttons).	JavaScript 1.0
Radio	Method	blur	blur()	Removes focus from the radio button.	JavaScript 1.0
Radio	Method	click	click()	Simulates a mouse-click on the radio button.	JavaScript 1.0
Radio	Method	focus	focus()	Gives focus to the radio button.	JavaScript 1.0
Radio	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
Reset	Object			A reset button on an HTML form. A reset button resets all elements in a form to their defaults.	JavaScript 1.0 JavaScript 1.1: added type property; added onBlur and onFocus event handlers; added blur and focus methods JavaScript 1.2: added handleEvent method
Reset	Property	form		Specifies the form containing the Reset object.	JavaScript 1.0
Reset	Property	name		Reflects the NAME attribute.	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
Reset	Property	type		Reflects the TYPE attribute.	JavaScript 1.1
Reset	Property	value		Reflects the VALUE attribute.	JavaScript 1.0
Reset	Method	blur	blur()	Removes focus from the reset button.	JavaScript 1.0
Reset	Method	click	click()	Simulates a mouse-click on the reset button.	JavaScript 1.0
Reset	Method	focus	focus()	Gives focus to the reset button.	JavaScript 1.0
Reset	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
screen	Object			Contains properties describing the display screen and colors.	JavaScript 1.2
screen	Property	availHeight		Specifies the height of the screen, in pixels, minus permanent or semipermanent user interface features displayed by the operating system, such as the Taskbar on Windows.	JavaScript 1.2
screen	Property	availWidth		Specifies the width of the screen, in pixels, minus permanent or semipermanent user interface features displayed by the operating system, such as the Taskbar on Windows.	JavaScript 1.2
screen	Property	colorDepth		The bit depth of the color palette, if one is in use; otherwise, the value is derived from screen.pixelDepth.	JavaScript 1.2
screen	Property	height		Display screen height.	JavaScript 1.2
screen	Property	pixelDepth		Display screen color resolution (bits per pixel).	JavaScript 1.2
screen	Property	width		Display screen width.	JavaScript 1.2
Select	Object			A selection list on an HTML form. The user can choose one or more items from a selection list, depending on how the list was created.	JavaScript 1.0 JavaScript 1.1: added type property; added the ability to add and delete options. JavaScript 1.2: added handleEvent method.

Objet	Type	Nom	Syntaxe	Description	Version JS
Select	Property	form		Specifies the form containing the selection list.	JavaScript 1.0
Select	Property	length		Reflects the number of options in the selection list.	JavaScript 1.0
Select	Property	name		Reflects the NAME attribute.	JavaScript 1.0
Select	Property	options	options[]	Reflects the OPTION tags.	JavaScript 1.0
Select	Property	selectedIndex		Reflects the index of the selected option (or the first selected option, if multiple options are selected).	JavaScript 1.0
Select	Property	type		Specifies that the object is represents a selection list and whether it can have one or more selected options.	JavaScript 1.1
Select	Method	blur	blur()	Removes focus from the selection list.	JavaScript 1.0
Select	Method	focus	focus()	Gives focus to the selection list.	JavaScript 1.0
Select	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
Submit	Object			A submit button on an HTML form. A submit button causes a form to be submitted.	JavaScript 1.0 JavaScript 1.1: added type property; added onBlur and onFocus event handlers; added blur and focus methods JavaScript 1.2: added handleEvent method
Submit	Property	form		Specifies the form containing the Submit object.	JavaScript 1.0
Submit	Property	name		Reflects the NAME attribute.	JavaScript 1.0
Submit	Property	type		Reflects the TYPE attribute.	JavaScript 1.1
Submit	Property	value		Reflects the VALUE attribute.	JavaScript 1.0
Submit	Method	blur	blur()	Removes focus from the submit button.	JavaScript 1.0
Submit	Method	click	click()	Simulates a mouse-click on the submit button.	JavaScript 1.0
Submit	Method	focus	focus()	Gives focus to the submit button.	JavaScript 1.0
Submit	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2

Objet	Type	Nom	Syntaxe	Description	Version JS
Text	Object			A text input field on an HTML form. The user can enter a word, phrase, or series of numbers in a text field.	JavaScript 1.0 JavaScript 1.1: added type property JavaScript 1.2: added handleEvent method
Text	Property	defaultValue		Reflects the VALUE attribute.	JavaScript 1.0
Text	Property	form		Specifies the form containing the Text object.	JavaScript 1.0
Text	Property	name		Reflects the NAME attribute.	JavaScript 1.0
Text	Property	type		Reflects the TYPE attribute.	JavaScript 1.1
Text	Property	value		Reflects the current value of the Text object's field.	JavaScript 1.0
Text	Method	blur	blur()	Removes focus from the object.	JavaScript 1.0
Text	Method	focus	focus()	Gives focus to the object.	JavaScript 1.0
Text	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
Text	Method	select	select()	Selects the input area of the object.	JavaScript 1.0
Textarea	Object			A multiline input field on an HTML form. The user can use a textarea field to enter words, phrases, or numbers.	JavaScript 1.0 JavaScript 1.1: added type property JavaScript 1.2: added handleEvent method
Textarea	Property	defaultValue		Reflects the VALUE attribute.	JavaScript 1.0
Textarea	Property	form		Specifies the form containing the Textarea object.	JavaScript 1.0
Textarea	Property	name		Reflects the NAME attribute.	JavaScript 1.0
Textarea	Property	type		Specifies that the object is a Textarea object.	JavaScript 1.1
Textarea	Property	value		Reflects the current value of the Textarea object.	JavaScript 1.0
Textarea	Method	blur	blur()	Removes focus from the object.	JavaScript 1.0
Textarea	Method	focus	focus()	Gives focus to the object.	JavaScript 1.0
Textarea	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
Textarea	Method	select	select()	Selects the input area of the object.	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
window	Object			Represents a browser window or frame. This is the top-level object for each document, Location, and History object group.	JavaScript 1.0 JavaScript 1.1: added closed, history, and opener properties; added blur, focus, and scroll methods; added onBlur, onError, and onFocus event handlers JavaScript 1.2: added innerHeight, innerWidth, locationbar, menubar, outerHeight, outerWidth, pageXOffset, pageYOffset, personalbar, scrollbars, statusbar, and toolbar properties; added back, captureEvents, clearInterval, disableExternalCapture, enableExternalCapture, find, forward, handleEvent, home, moveBy, moveTo, releaseEvents, resizeBy, resizeTo, routeEvent, scrollBy, scrollTo, setInterval, and stop methods; deprecated scroll method
window	Property	closed		Specifies whether a window has been closed.	JavaScript 1.1
window	Property	defaultStatus		Reflects the default message displayed in the window's status bar.	JavaScript 1.0
window	Property	document		Contains information on the current document, and provides methods for displaying HTML output to the user.	JavaScript 1.0
window	Property	frames	frames[]	An array reflecting all the frames in a window.	JavaScript 1.0
window	Property	history		Contains information on the URLs that the client has visited within a window.	JavaScript 1.1
window	Property	innerHeight		Specifies the vertical dimension, in pixels, of the window's content area.	JavaScript 1.2
window	Property	innerWidth		Specifies the horizontal dimension, in pixels, of the window's content area.	JavaScript 1.2

Objet	Type	Nom	Syntaxe	Description	Version JS
window	Property	length		The number of frames in the window.	JavaScript 1.0
window	Property	location		Contains information on the current URL.	JavaScript 1.0
window	Property	locationbar		Represents the browser window's location bar.	JavaScript 1.2
window	Property	menubar		Represents the browser window's menu bar.	JavaScript 1.2
window	Property	name		A unique name used to refer to this window.	JavaScript 1.0
window	Property	opener		Specifies the window name of the calling document when a window is opened using the open method	JavaScript 1.1
window	Property	outerHeight		Specifies the vertical dimension, in pixels, of the window's outside boundary.	JavaScript 1.2
window	Property	outerWidth		Specifies the horizontal dimension, in pixels, of the window's outside boundary.	JavaScript 1.2
window	Property	pageXOffset		Provides the current x-position, in pixels, of a window's viewed page.	JavaScript 1.2
window	Property	pageYOffset		Provides the current y-position, in pixels, of a window's viewed page.	JavaScript 1.2
window	Property	parent		A synonym for a window or frame whose frameset contains the current frame.	JavaScript 1.0
window	Property	personalbar		Represents the browser window's personal bar (also called the directories bar).	JavaScript 1.2
window	Property	scrollbars		Represents the browser window's scroll bars.	JavaScript 1.2
window	Property	self		A synonym for the current window.	JavaScript 1.0
window	Property	status		Specifies a priority or transient message in the window's status bar.	JavaScript 1.0
window	Property	statusbar		Represents the browser window's status bar.	JavaScript 1.2
window	Property	toolbar		Represents the browser window's tool bar.	JavaScript 1.2
window	Property	top		A synonym for the topmost browser window.	JavaScript 1.0
window	Property	window		A synonym for the current window.	JavaScript 1.0
window	Method	alert	alert(message)	Displays an Alert dialog box with a message and an OK button.	JavaScript 1.0

Objet	Type	Nom	Syntaxe	Description	Version JS
window	Method	back	back()	Undoes the last history step in any frame within the top-level window.	JavaScript 1.2
window	Method	blur	blur()	Removes focus from the specified object.	JavaScript 1.0
window	Method	captureEvents	captureEvents(eventType)	Sets the window or document to capture all events of the specified type.	JavaScript 1.2
window	Method	clearInterval	clearInterval(intervalID)	Cancels a timeout that was set with the setInterval method.	JavaScript 1.2
window	Method	clearTimeout	clearTimeout(timeoutID)	Cancels a timeout that was set with the setTimeout method.	JavaScript 1.0
window	Method	close	close()	Closes the specified window.	JavaScript 1.0: closes any window JavaScript 1.1: closes only windows opened by JavaScript JavaScript 1.2: must use signed scripts to unconditionally close a window
window	Method	confirm	confirm(message)	Displays a Confirm dialog box with the specified message and OK and Cancel buttons.	JavaScript 1.0
window	Method	disableExternalCapture	disableExternalCapture()	Disables external event capturing set by the enableExternalCapture method.	JavaScript 1.2
window	Method	enableExternalCapture	enableExternalCapture()	Allows a window with frames to capture events in pages loaded from different locations (servers).	JavaScript 1.2
window	Method	find	find([string[, caseSensitive[, backward]]])	Finds the specified text string in the contents of the specified window.	JavaScript 1.2
window	Method	focus	focus()	Gives focus to the specified object.	JavaScript 1.1
window	Method	forward	history.forward()	Loads the next URL in the history list.	JavaScript 1.2
window	Method	handleEvent	handleEvent(event)	Invokes the handler for the specified event.	JavaScript 1.2
window	Method	home	home()	Points the browser to the URL specified in preferences as the user's home page.	JavaScript 1.2
window	Method	moveBy	moveBy(horizontal, vertical)	Moves the window by the specified amounts.	JavaScript 1.2
window	Method	moveTo	moveTo(x-coordinate, y-coordinate)	Moves the top-left corner of the window to the specified screen coordinates.	JavaScript 1.2

Objet	Type	Nom	Syntaxe	Description	Version JS
window	Method	open	open(URL, windowName[, windowFeatures])	Opens a new web browser window.	JavaScript 1.0 JavaScript 1.2: added several new windowFeatures
window	Method	print	print()	Prints the contents of the window or frame.	JavaScript 1.2
window	Method	prompt	prompt(message[, inputDefault])	Displays a Prompt dialog box with a message and an input field.	JavaScript 1.0
window	Method	releaseEvents	releaseEvents(eventType)	Sets the window to release captured events of the specified type, sending the event to objects further along the event hierarchy.	JavaScript 1.2
window	Method	resizeBy	resizeBy(horizontal, vertical)	Resizes an entire window by moving the window's bottom-right corner by the specified amount.	JavaScript 1.2
window	Method	resizeTo	resizeTo(outerWidth, outerHeight)	Resizes an entire window to the specified outer height and width.	JavaScript 1.2
window	Method	routeEvent	routeEvent(event)	Passes a captured event along the normal event hierarchy.	JavaScript 1.2
window	Method	scroll		Scrolls a window to a specified coordinate.	JavaScript 1.1 JavaScript 1.2: deprecated
window	Method	scrollBy	scrollBy(horizontal, vertical)	Scrolls the viewing area of a window by the specified amount.	JavaScript 1.2
window	Method	scrollTo	scrollTo(x-coordinate, y-coordinate)	Scrolls the viewing area of the window to the specified coordinates, such that the specified point becomes the top-left corner.	JavaScript 1.2
window	Method	setInterval	setInterval(expression, msec) setInterval(function, msec[, arg1[, ..., argN]])	Evaluates an expression or calls a function every time a specified number of milliseconds elapses.	JavaScript 1.2
window	Method	setTimeout	setTimeout(expression, msec) setTimeout(function, msec[, arg1[, ..., argN]])	Evaluates an expression or calls a function once after a specified number of milliseconds has elapsed.	JavaScript 1.0: evaluating an expression JavaScript 1.2: calling a function
window	Method	stop	stop()	Stops the current download.	JavaScript 1.2

ANNEXE G : RÉFÉRENCES DOM W3C

Le tableau ci-dessous indique les directives du langage JavaScript propres aux recommandations du DOM W3C et les versions du DOM de leur introduction.

Les lignes surlignées en rouge interdisent leur implémentation vis à vis de leur compatibilité et de leur difficulté de mise en oeuvre.

Les lignes surlignées en orange demandent une implémentation de détection du fait de leur problème de compatibilité (voir Norme de développement JS du présent document pour explications).

Les lignes surlignées en bleu autorisent des directives JavaScript propres au MSIE DHTML DOM du fait de leurs absences des recommandations du DOM W3C mais d'une nécessité d'extansion reprise par l'ensemble des navigateurs ciblés.

Les directives MSIE DHTML DOM (autres que celles surlignées en bleu) ou Gecko DOM ne sont pas listées dans ce référencement. Elles demeurent pour autant interdites d'implémentation du fait de leurs implémentations propriétaires.

De plus, pour les éléments HTML de type INPUT, TEXTAREA et SELECT, cette liste ne reprend pas les directives du DOM Level 0 qui ont été pourtant reprises par les recommandations DOM W3C.

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
All	Collection	attributes	attributes[]	Retrieves a collection of attributes of the object.	Level 1
All	Collection	childNodes	childNodes[]	Retrieves a collection of HTML Elements and TextNode objects that are direct descendants of the specified object.	Level 1
All	Property	className		Sets or retrieves the class of the object.	Level 1
All	Property	dir		Sets or retrieves the reading order of the object.	Level 1
All	Property	firstChild		Retrieves a reference to the first child in the childNodes collection of the object.	Level 1
All	Property	id		Retrieves the string identifying the object.	Level 1
All	Property	innerHTML		Returns all of the markup and content within a given element	MSIE DHTML DOM
All	Property	lang		Sets or retrieves the language to use.	Level 1
All	Property	lastChild		Retrieves a reference to the last child in the childNodes collection of an object.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
All	Property	length		Retrieves the number of characters in a TextNode object \ Sets or retrieves the number of objects in a collection.	Level 1
All	Property	localName		localName returns the local part of the qualified name of this node.	Level 1
All	Property	namespaceURI		The namespace URI of this node, or NULL if it is unspecified.	Level 1
All	Property	nextSibling		Retrieves a reference to the next child of the parent for the object.	Level 1
All	Property	nodeName		Retrieves the name of a particular type of node.	Level 1
All	Property	nodeType		Retrieves the type of the requested node.	Level 1
All	Property	nodeValue		Sets or retrieves the value of a node.	Level 1
All	Property	offsetHeight		Retrieves the height of the object relative to the layout or coordinate parent, as specified by the offsetParent property.	MSIE DHTML DOM
All	Property	offsetLeft		Retrieves the calculated left position of the object relative to the layout or coordinate parent, as specified by the offsetParent property.	MSIE DHTML DOM
All	Property	offsetParent		Retrieves a reference to the container object that defines the offsetTop and offsetLeft properties of the object.	MSIE DHTML DOM
All	Property	offsetTop		Retrieves the calculated top position of the object relative to the layout or coordinate parent, as specified by the offsetParent property.	MSIE DHTML DOM
All	Property	offsetWidth		Retrieves the width of the object relative to the layout or coordinate parent, as specified by the offsetParent property.	MSIE DHTML DOM
All	Property	ownerDocument		Sets or retrieves the document object associated with the node.	Level 1
All	Property	parentNode		Retrieves the parent object in the document hierarchy.	Level 1
All	Property	previousSibling		Retrieves a reference to the previous child of the parent for the object.	Level 1
All	Property	style		Sets an inline style for the element.	Level 1
All	Property	tabIndex		Sets or retrieves the index that defines the tab order for the object.	Level 1
All	Property	tagName		Retrieves the tag name of the object.	Level 1
All	Property	title		Sets or retrieves advisory information (a ToolTip) for the object.	Level 1
All	Method	addEventListener	addEventListener("evt",func, capt)	addEventListener allows the registration of event listeners on the event target.	Level 1
All	Method	appendChild	appendChild(elem)	Appends an element as a child to the object.	Level 1
All	Method	blur	blur()	Causes the element to lose focus and fires the onblur event.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
All	Method	click	click()	Simulates a click by causing the onclick event to fire.	Level 1
All	Method	cloneNode	cloneNode(deep)	Copies a reference to the object from the document hierarchy.	Level 1
All	Method	dispatchEvent	dispatchEvent(evt)	The dispatchEvent method allows the dispatch of events into the implementation's event model.	Level 1
All	Method	focus	focus()	Causes the element to receive the focus and executes the code specified by the onfocus event.	Level 1
All	Method	getAttribute	getAttribute("attr")	Returns the value of the named attribute from the userProfile object.	Level 1
All	Method	getAttributeNode	getAttributeNode("attr")	Retrieves an attribute object referenced by the attribute.name property.	Level 1
All	Method	getElementsByTagName	getElementsByTagName("tag")	Retrieves a collection of objects based on the specified element name.	Level 1
All	Method	hasChildNodes	hasChildNodes()	Returns a value that indicates whether the object has children.	Level 1
All	Method	insertBefore	insertBefore(new[,ref])	Inserts an element into the document hierarchy as a child node of a parent object.	Level 1
All	Method	item	item(index)	Retrieves an attribute for an element from the attributes collection \ Retrieves an object from the all collection or various other collections.	Level 1
All	Method	normalize	normalize()	Merges adjacent TextNode objects to produce a normalized document object model.	Level 1
All	Method	removeAttribute	removeAttribute("attr")	Removes the specified attribute from the object.	Level 1
All	Method	removeAttributeNode	removeAttributeNode(node)	Removes an attribute object from the object.	Level 1
All	Method	removeChild	removeChild(node)	Removes a child node from the object.	Level 1
All	Method	removeEventListener	removeEventListener("evt",func,capt)		
All	Method	replaceChild	replaceChild(new,old)	Replaces an existing child element with a new child element.	Level 1
All	Method	setAttribute	setAttribute("attr",val)	Sets the value of the specified attribute.	Level 1
All	Method	setAttributeNode	setAttributeNode(node)	Sets an attribute object node as part of the object.	Level 1
All	Method	supports	supports("feature")	The supports method tests if this DOM implementation supports a particular feature.	Level 1
A (anchor/link)	Property	charset		Sets or retrieves the character set used to encode the object.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
A (anchor/link)	Property	coords		Sets or retrieves the coordinates of the object.	Level 1
A (anchor/link)	Property	hash		Sets or retrieves the subsection of the href property that follows the number sign (#).	Level 1
A (anchor/link)	Property	host		Sets or retrieves the hostname and port number of the location or URL.	Level 1
A (anchor/link)	Property	hostname		Sets or retrieves the host name part of the location or URL.	Level 1
A (anchor/link)	Property	href		Sets or retrieves the destination URL or anchor point.	Level 1
A (anchor/link)	Property	hreflang		Sets or retrieves the language code of the object.	Level 1
A (anchor/link)	Property	name		Sets or retrieves the name of the object.	Level 1
A (anchor/link)	Property	pathname		Sets or retrieves the file name or path specified by the object.	Level 1
A (anchor/link)	Property	port		Sets or retrieves the port number associated with a URL.	Level 1
A (anchor/link)	Property	protocol		Sets or retrieves the protocol portion of a URL.	Level 1
A (anchor/link)	Property	rel		Sets or retrieves the relationship between the object and the destination of the link.	Level 1
A (anchor/link)	Property	rev		Sets or retrieves the relationship between the object and the destination of the link.	Level 1
A (anchor/link)	Property	search		Sets or retrieves the substring of the href property that follows the question mark.	Level 1
A (anchor/link)	Property	shape		Sets or retrieves the shape of the object.	Level 1
A (anchor/link)	Property	target		Sets or retrieves the window or frame at which to target content.	Level 1
APPLET	Property	align		Sets or retrieves how the object is aligned with adjacent text.	Level 1
APPLET	Property	alt		Sets or retrieves a text alternative to the graphic.	Level 1
APPLET	Property	archive		Sets or retrieves a character string that can be used to implement your own archive functionality for the object.	Level 1
APPLET	Property	code		Sets or retrieves the URL of the file containing the compiled Java class.	Level 1
APPLET	Property	codeBase		Sets or retrieves the URL of the component.	Level 1
APPLET	Property	height		Sets or retrieves the height of the object.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
APPLET	Property	hspace		Sets or retrieves the horizontal margin for the object.	Level 1
APPLET	Property	name		Sets or retrieves the name of the object.	Level 1
APPLET	Property	object		Retrieves the contained object.	Level 1
APPLET	Property	vspace		Sets or retrieves the vertical margin for the object.	Level 1
APPLET	Property	width		Sets or retrieves the width of the object.	Level 1
AREA	Property	alt		Sets or retrieves a text alternative to the graphic.	Level 1
AREA	Property	coords		Sets or retrieves the coordinates of the object.	Level 1
AREA	Property	hash		Sets or retrieves the subsection of the href property that follows the number sign (#).	Level 1
AREA	Property	host		Sets or retrieves the hostname and port number of the location or URL.	Level 1
AREA	Property	hostname		Sets or retrieves the host name part of the location or URL.	Level 1
AREA	Property	href		Sets or retrieves the destination URL or anchor point.	Level 1
AREA	Property	noHref		Sets or retrieves whether clicks in this region cause action.	Level 1
AREA	Property	pathname		Sets or retrieves the file name or path specified by the object.	Level 1
AREA	Property	port		Sets or retrieves the port number associated with a URL.	Level 1
AREA	Property	protocol		Sets or retrieves the protocol portion of a URL.	Level 1
AREA	Property	search		Sets or retrieves the substring of the href property that follows the question mark.	Level 1
AREA	Property	shape		Sets or retrieves the shape of the object.	Level 1
AREA	Property	target		Sets or retrieves the window or frame at which to target content.	Level 1
BLOCKQUOTE	Property	cite		Sets or retrieves reference information about the object.	Level 1
BODY	Property	alink		Sets or retrieves the color of all active links in the element.	Level 1
BODY	Property	background		Sets or retrieves the background picture tiled behind the text and graphics in the object.	Level 1
BODY	Property	bgcolor		Sets or retrieves the background color behind the object.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
BODY	Property	link		Sets or retrieves the color of the document links for the object.	Level 1
BODY	Property	text		Sets or retrieves the text (foreground) color for the document body.	Level 1
BODY	Property	vLink		Sets or retrieves the color of links in the object that have already been visited.	Level 1
BR	Property	clear		Sets or retrieves the side on which floating objects are not to be positioned when a line break is inserted into the document.	Level 1
CAPTION	Property	align		Sets or retrieves the alignment of the caption or legend.	Level 1
CAPTION	Property	vAlign		Sets or retrieves whether the caption appears at the top or bottom of the table.	Level 1
COL, COLGROUP	Property	align		Sets or retrieves the alignment of the object relative to the display or table.	Level 1
COL, COLGROUP	Property	ch		Sets or retrieves a value that you can use to implement your own ch functionality for the object.	Level 1
COL, COLGROUP	Property	chOff		Sets or retrieves a value that you can use to implement your own chOff functionality for the object.	Level 1
COL, COLGROUP	Property	span		Sets or retrieves the number of columns in the group.	Level 1
COL, COLGROUP	Property	vAlign		Sets or retrieves how text and other content are vertically aligned within the object that contains them.	Level 1
COL, COLGROUP	Property	width		Sets or retrieves the width of the object.	Level 1
cssRule	Property	cssText		Sets or retrieves the persisted representation of the style rule.	Level 2
cssRule	Property	parentStyleSheet		Retrieves the style sheet that imported the current style sheets.	Level 2
cssRule	Property	selectorText		Retrieves a string that identifies which elements the corresponding style sheet rule applies to.	Level 2

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
cssRule	Property	style		Sets an inline style for the element.	Level 2
cssRule	Property	type		Retrieves the CSS language in which the style sheet is written.	Level 2
DL, DT, DD, DIR, MENU	Property	compact		Sets or retrieves a Boolean value indicating whether the list should be compacted by removing extra space between list objects.	Level 1
document	Collection	anchors	anchors[]	Retrieves a collection of all a objects that have a name and/or id property. Objects in this collection are in HTML source order.	Level 1
document	Collection	applets	applets[]	Retrieves a collection of all applet objects in the document.	Level 1
document	Collection	attributes	attributes[]	Retrieves a collection of attributes of the object.	Level 1
document	Collection	childNodes	childNodes[]	Retrieves a collection of HTML Elements and TextNode objects that are direct descendants of the specified object.	Level 1
document	Collection	embeds	embeds[]	embeds returns a list of the embedded OBJECTS within the current document.	Level 1
document	Collection	forms	forms[]	Retrieves a collection, in source order, of all form objects in the document.	Level 1
document	Collection	images	images[]	Retrieves a collection, in source order, of img objects in the document.	Level 1
document	Collection	links	links[]	Retrieves a collection of all a objects that specify the HREF property and all area objects in the document.	Level 1
document	Collection	styleSheets	styleSheets[]	The stylesheets property returns a list of the stylesheet objects on the current document.	Level 1
document	Property	alinkColor		Sets or retrieves the color of all active links in the document.	Level 1
document	Property	bgColor		Sets or retrieves the background color behind the object.	Level 1
document	Property	body		body returns the BODY node of the current document.	Level 1
document	Property	characterSet		Returns the character set being used by the document.	Level 1
document	Property	charSet		Returns the character set being used by the document.	MSIE DHTML DOM
document	Property	cookie		Sets or retrieves the string value of a cookie.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
document	Property	doctype		Retrieves the document type declaration associated with the current document.	Level 1
document	Property	documentElement		Retrieves a reference to the root node of the document	Level 1
document	Property	domain		Sets or retrieves the security domain of the document.	Level 1
document	Property	fgColor		Sets or retrieves the foreground (text) color of the document.	Level 1
document	Property	firstChild		Retrieves a reference to the first child in the childNodes collection of the object.	Level 1
document	Property	height		Sets or retrieves the height of the object.	Level 1
document	Property	implementation		Retrieves the implementation object of the current document.	Level 1
document	Property	lastChild		Retrieves a reference to the last child in the childNodes collection of an object.	Level 1
document	Property	lastModified		Retrieves the date the page was last modified, if the page supplies one.	Level 1
document	Property	linkColor		Sets or retrieves the color of the document links.	Level 1
document	Property	location		Returns the URI of the current document.	Level 1
document	Property	namespaceURI		Returns the XML namespace of the current document.	Level 1
document	Property	nextSibling		Retrieves a reference to the next child of the parent for the object.	Level 1
document	Property	nodeName		Retrieves the name of a particular type of node.	Level 1
document	Property	nodeType		Retrieves the type of the requested node.	Level 1
document	Property	ownerDocument		Sets or retrieves the document object associated with the node.	Level 1
document	Property	parentNode		Retrieves the parent object in the document hierarchy.	Level 1
document	Property	plugins		Returns an array of the available plugins.	Level 1
document	Property	previousSibling		Retrieves a reference to the previous child of the parent for the object.	Level 1
document	Property	referrer		Retrieves the URL of the location that referred the user to the current page.	Level 1
document	Property	title		Sets or retrieves advisory information (a ToolTip) for the object.	Level 1
document	Property	URL		Sets or retrieves the URL for the current document.	Level 1
document	Property	vlinkColor		Sets or retrieves the color of the links that the user has visited.	Level 1
document	Property	width		Sets or retrieves the width of the object.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
document	Method	clear	clear()	Clears a document.	Level 1
document	Method	close	close()	Closes an output stream and forces the sent data to display.	Level 1
document	Method	createAttribute	createAttribute("name")	Creates an attribute object with a specified name.	Level 1
document	Method	createDocumentFragment	createDocumentFragment()	Creates a new document.	Level 1
document	Method	createElement	createElement("tag")	Creates an instance of the element for the specified tag.	Level 1
document	Method	createTextNode	createTextNode("txt")	Creates a text string from the specified value.	Level 1
document	Method	focus	focus()	Causes the element to receive the focus and executes the code specified by the onfocus event.	Level 1
document	Method	getElementById	getElementById("ID")	Returns a reference to the first object with the specified value of the ID attribute.	Level 1
document	Method	getElementByName	getElementByName("name")	Retrieves a collection of objects based on the value of the NAME attribute.	Level 1
document	Method	getElementbyTagName	getElementbyTagName("tag")	Retrieves a collection of objects based on the specified element name.	Level 1
document	Method	open	open("mimeType"[,replace])	This method works in two ways. It opens a document to collect the output of the write and writeln methods. In this case, only the first two parameters, url and name are used. When values for the additional parameters are specified, this method opens a window in the same way as the window.open method for the window object.	Level 1
document	Method	write	write("str")	Writes one or more HTML expressions to a document in the specified window.	Level 1
document	Method	writeln	writeln("str")	Writes one or more HTML expressions, followed by a carriage return, to a document in the specified window.	Level 1
EMBED	Property	align		Sets or retrieves how the object is aligned with adjacent text.	Level 1
EMBED	Property	height		Sets or retrieves the height of the object.	Level 1
EMBED	Property	name		Sets or retrieves the name of the object.	Level 1
EMBED	Property	src		Retrieves the URL to an external file that contains the source code or data.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
EMBED	Property	width		Sets or retrieves the width of the object.	Level 1
event	Property	altkey		Sets or retrieves a value that indicates the state of the ALT key.	Level 2
event	Property	bubbles		Returns a boolean indicating whether the event bubbles up through the DOM or not.	Level 2
event	Property	button		Retrieves the value to which mouse button is pressed during the event.	Level 2
event	Property	cancelable		Returns a boolean indicating whether the event is cancelable.	Level 2
event	Property	cancelBubble		Sets or retrieves whether the current event should bubble up the hierarchy of event handlers.	Level 2
event	Property	charCode		Returns a number representing the character that was pressed as part of the key event.	Level 2
event	Property	clientX		Sets or retrieves the x-coordinate of the mouse pointer's position relative to the client area of the window, excluding window decorations and scroll bars.	Level 2
event	Property	clientY		Sets or retrieves the y-coordinate of the mouse pointer's position relative to the client area of the window, excluding window decorations and scroll bars.	Level 2
event	Property	ctrlKey		Sets or retrieves the state of the CTRL key.	Level 2
event	Property	currentTarget		Returns a reference to the currently registered target for the event.	Level 2
event	Property	detail		Returns detail about the event, depending on the type of event.	Level 2
event	Property	eventPhase		Used to indicate which phase of the event flow is currently being evaluated.	Level 2
event	Property	isChar		Returns a boolean indicating whether the event produced a key character or not.	Level 2
event	Property	keyCode		Sets or retrieves the Unicode key code associated with the key that caused the event.	Level 2
event	Property	layerX		Returns the horizontal coordinate of the event relative to the current layer.	Level 2

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
event	Property	layerY		Returns the vertical coordinate of the event relative to the current layer.	Level 2
event	Property	metaKey		Returns a boolean indicating whether the meta key was pressed during the event.	Level 2
event	Property	pageX		Returns the horizontal coordinate of the event relative to the page.	Level 2
event	Property	pageY		Returns the vertical coorindate of the page relative to the page.	Level 2
event	Property	relatedTarget		Identifies a secondary target for the event.	Level 2
event	Property	screenX		Retrieves the x-coordinate of the mouse pointer's position relative to the user's screen.	Level 2
event	Property	screenY		Sets or retrieves the y-coordinate of the mouse pointer's position relative to the user's screen.	Level 2
event	Property	shiftKey		Sets or retrieves the state of the SHIFT key.	Level 2
event	Property	srcElement		Sets or retrieves the element at which to fire event.	MSIE DHTML DOM
event	Property	target		Sets or retrieves the element at which to fire event.	Level 2
event	Property	timeStamp		Returns the time that the event was created.	Level 2
event	Property	type		Retrieves the type of event	Level 2
event	Property	view		The view attribute identifies the AbstractView from which the event was generated.	Level 2
event	Method	initEvent	initEvent("type",bubble,cancelable)	The initEvent method is used to initialize the value of an Event created through the DocumentEvent interface.	Level 2
event	Method	initKeyEvent	initKeyEvent("type",evtArgs)		Level 2
event	Method	initMouseEvent	initMouseEvent("type",evtArgs)	This method intializes the value of a mouse event once it's been created	Level 2
event	Method	initUIEvent	initUIEvent("type",evtArgs)	Initializes a UI event once it's been created.	Level 2
event	Method	preventDefault	preventDefault()	Cancels the event (if it is cancelable).	Level 2
event	Method	stopPropagation	stopPropagation()	Stops the propagation of events further along in the DOM.	Level 2

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
FIELDSET,LEGEND	Property	align		Sets or retrieves how the object is aligned with adjacent text.	Level 1
FIELDSET,LEGEND	Property	form		Retrieves a reference to the form that the object is embedded in.	Level 1
FONT	Property	color		Sets or retrieves the color of the text of the object.	Level 1
FONT	Property	face		Sets or retrieves the current typeface family.	Level 1
FONT	Property	size		Sets or retrieves the font size of the object.	Level 1
FORM	Collection	elements	elements[]	Retrieves a collection, in source order, of all controls in a given form. input type=image objects are excluded from the collection.	Level 1
FORM	Property	acceptCharset		Sets or retrieves a list of character encodings for input data that must be accepted by the server processing the form.	Level 1
FORM	Property	action		Sets or retrieves the URL to which the form content is sent for processing.	Level 1
FORM	Property	encoding		Sets or retrieves the MIME encoding for the form.	Level 1
FORM	Property	enctype		Sets or retrieves the Multipurpose Internet Mail Extensions (MIME) encoding for the form.	Level 1
FORM	Property	length		Sets or retrieves the number of objects in a collection.	Level 1
FORM	Property	method		Sets or retrieves how to send the form data to the server.	Level 1
FORM	Property	name		Sets or retrieves the name of an input parameter for an element.	Level 1
FORM	Property	target		Sets or retrieves the window or frame at which to target content.	Level 1
FORM	Method	reset	reset()	Simulates a mouse click on a reset button for the calling form.	Level 1
FORM	Method	submit	submit()	Submits the form.	Level 1
FRAME	Property	frameBorder		Sets or retrieves whether to display a border for the frame.	Level 1
FRAME	Property	longDesc		Sets or retrieves a Uniform Resource Identifier (URI) to a long description of the object.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
FRAME	Property	marginHeight		Sets or retrieves the top and bottom margin heights before displaying the text in a frame.	Level 1
FRAME	Property	marginWidth		Sets or retrieves the left and right margin widths before displaying the text in a frame.	Level 1
FRAME	Property	noResize		Sets or retrieves whether the user can resize the frame.	Level 1
FRAME	Property	scrolling		Sets or retrieves whether the frame can be scrolled.	Level 1
FRAME	Property	src		Sets or retrieves a URL to be loaded by the object.	Level 1
FRAMESET	Property	cols		Sets or retrieves the frame widths of the object.	Level 1
FRAMESET	Property	rows		Sets or retrieves the frame heights of the object.	Level 1
H1…H6	Property	align		Sets or retrieves how the object is aligned with adjacent text.	Level 1
HR	Property	align		Sets or retrieves how the object is aligned with adjacent text.	Level 1
HR	Property	color		Sets or retrieves the color of the text of the object.	Level 1
HR	Property	noShade		Sets or retrieves whether the horizontal rule is drawn with 3-D shading.	Level 1
HR	Property	size		Sets or retrieves the font size of the object.	Level 1
HR	Property	width		Sets or retrieves the calculated width of the object.	Level 1
HTML	Property	version		Sets or retrieves the Document Type Definition (DTD) version that governs the current document.	Level 1
IFRAME	Property	align		Sets or retrieves how the object is aligned with adjacent text.	Level 1
IFRAME	Property	frameBorder		Sets or retrieves whether to display a border for the iFrame.	Level 1
IFRAME	Property	longDesc		Sets or retrieves a Uniform Resource Identifier (URI) to a long description of the object.	Level 1
IFRAME	Property	marginHeight		Sets or retrieves the top and bottom margin heights before displaying the text in a frame.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
IFRAME	Property	marginWidth		Sets or retrieves the left and right margin widths before displaying the text in a frame.	Level 1
IFRAME	Property	scrolling		Sets or retrieves whether the frame can be scrolled.	Level 1
IFRAME	Property	src		Sets or retrieves a URL to be loaded by the object.	Level 1
IMG	Property	align		Sets or retrieves how the object is aligned with adjacent text.	Level 1
IMG	Property	alt		Sets or retrieves a text alternative to the graphic.	Level 1
IMG	Property	border		Sets or retrieves the width of the border to draw around the object.	Level 1
IMG	Property	complete		Retrieves whether the object is fully loaded.	Level 1
IMG	Property	height		Sets or retrieves the height of the object.	Level 1
IMG	Property	href		Sets or retrieves the entire URL as a string.	Level 1
IMG	Property	hspace		Sets or retrieves the horizontal margin for the object.	Level 1
IMG	Property	isMap		Sets or retrieves whether the image is a server-side image map.	Level 1
IMG	Property	longDesc		Sets or retrieves a Uniform Resource Identifier (URI) to a long description of the object.	Level 1
IMG	Property	lowSrc		Sets or retrieves a lower resolution image to display.	Level 1
IMG	Property	name		Sets or retrieves the name of the object.	Level 1
IMG	Property	src		Sets or retrieves a URL to be loaded by the object.	Level 1
IMG	Property	useMap		Sets or retrieves the URL, often with a bookmark extension (#name), to use as a client-side image map.	Level 1
IMG	Property	vspace		Sets or retrieves the vertical margin for the object.	Level 1
IMG	Property	width		Sets or retrieves the width of the object.	Level 1
INPUT (button, reset, submit, radio, checkbox)	Property	disabled		Sets or retrieves the value that indicates whether the user can interact with the object.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
INPUT (image)	Property	disabled		Sets or retrieves the value that indicates whether the user can interact with the object.	Level 1
INPUT (image)	Property	form		Retrieves a reference to the form that the object is embedded in.	Level 1
INPUT (image)	Property	name		Sets or retrieves the name of an input parameter for an element.	Level 1
INPUT (image)	Property	src		Sets or retrieves a URL to be loaded by the object.	Level 1
INPUT (image)	Property	type		Retrieves the classification and default behavior of the button.	Level 1
INPUT (text, password, hidden)	Property	disabled		Sets or retrieves the value that indicates whether the user can interact with the object.	Level 1
INPUT (text, password, hidden)	Property	maxLength		Sets or retrieves the maximum number of characters that the user can enter in a text control.	Level 1
INPUT (text, password, hidden)	Property	readOnly		Sets or retrieves the value indicated whether the content of the object is read-only.	Level 1
INPUT (text, password, hidden)	Property	size		Sets or retrieves the size of the control.	Level 1
LABEL	Property	accessKey		Sets or retrieves the accelerator key for the object.	Level 1
LABEL	Property	form		Retrieves a reference to the form that the object is embedded in.	Level 1
LABEL	Property	htmlFor		Sets or retrieves the object to which the given label object is assigned.	Level 1
LI	Property	type		Retrieves the type of list.	Level 1
LI	Property	value		Sets or retrieves the value of the object.	Level 1
LINK	Property	charset		Sets or retrieves the character set used to encode the object.	Level 1
LINK	Property	disabled		Sets or retrieves the status of the object.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
LINK	Property	href		Sets or retrieves the baseline URL on which relative links will be based.	Level 1
LINK	Property	hreflang		Sets or retrieves the language code of the object.	Level 1
LINK	Property	media		Sets or retrieves the media type.	Level 1
LINK	Property	rel		Sets or retrieves the relationship between the object and the destination of the link.	Level 1
LINK	Property	rev		Sets or retrieves the relationship between the object and the destination of the link.	Level 1
LINK	Property	target		Sets or retrieves the name of a window or frame that is the target for navigation.	Level 1
MAP	Collection	areas	areas[]	Retrieves a collection of the area objects defined for the given map object.	Level 1
MAP	Property	name		Sets or retrieves the name of the object.	Level 1
META	Property	charset		Sets or retrieves the character set used to encode the object.	Level 1
META	Property	content		Sets or retrieves meta-information to associate with HTTP-EQUIV or NAME.	Level 1
META	Property	httpEquiv		Sets or retrieves information used to bind the META tag's content to an HTTP response header.	Level 1
META	Property	name		Sets or retrieves the value specified in the CONTENT attribute of the meta object.	Level 1
OBJECT	Property	align		Sets or retrieves how the object is aligned with adjacent text.	Level 1
OBJECT	Property	alt		Sets or retrieves a text alternative to the graphic.	Level 1
OBJECT	Property	code		Sets or retrieves the URL of the file containing the compiled Java class.	Level 1
OBJECT	Property	codeBase		Sets or retrieves the URL of the component.	Level 1
OBJECT	Property	codeType		Sets or retrieves the Internet media type for the code associated with the object.	Level 1
OBJECT	Property	height		Sets or retrieves the height of the object.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
OBJECT	Property	hspace		Sets or retrieves the horizontal margin for the object.	Level 1
OBJECT	Property	name		Sets or retrieves the name of the object.	Level 1
OBJECT	Property	object		Retrieves the contained object.	Level 1
OBJECT	Property	vspace		Sets or retrieves the vertical margin for the object.	Level 1
OBJECT	Property	width		Sets or retrieves the width of the object.	Level 1
OL	Property	compact		Sets or retrieves a Boolean value indicating whether the list should be compacted by removing extra space between list objects.	Level 1
OL	Property	start		Sets or retrieves the starting number for an ordered list.	Level 1
OL	Property	type		Retrieves the type of list.	Level 1
OPTGROUP	Property	form		Retrieves a reference to the form that the object is embedded in.	Level 1
OPTGROUP	Property	label		Sets or retrieves the label for the option group.	Level 1
OPTION	Property	defaultSelected		Sets or retrieves the status of the option.	Level 1
OPTION	Property	disabled		Sets or retrieves the value that indicates whether the user can interact with the object.	Level 1
OPTION	Property	form		Retrieves a reference to the form that the object is embedded in.	Level 1
OPTION	Property	label		Sets or retrieves the label for the option.	Level 1
OPTION	Property	selected		Sets or retrieves whether the option in the list box is the default item.	Level 1
OPTION	Property	text		Retrieves or sets the text of the object as a string.	Level 1
OPTION	Property	value		Sets or retrieves the value of the object.	Level 1
Range	Property	collapsed		Returns a boolean indicating whether a range is collapsed.	Level 2
Range	Property	commonAncestorContainer		Returns the deepest Node that contains the startContainer and endContainer Nodes.	Level 2
Range	Property	endContainer		Returns the Node within which the Range ends.	Level 2
Range	Property	endOffset		Returns a number representing where in the endContainer the Range ends.	Level 2

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
Range	Property	startContainer		Returns the Node within which the Range starts.	Level 2
Range	Property	startOffset		Returns a number representing where in the startContainer the Range starts.	Level 2
Range	Method	cloneContents	cloneContents()	Returns a document fragment copying the nodes of a Range.	Level 2
Range	Method	cloneRange	cloneRange()	Returns a Range object with boundary points identical to the cloned Range.	Level 2
Range	Method	collapse	collapse([start])	Collapses the Range to one of its boundary points.	Level 2
Range	Method	compareBondaryPoints	compareBondaryPoints(type, src)	Compares the boundary points of two Ranges.	Level 2
Range	Method	createContextualFragment	createContextualFragment("text")	Returns a document fragment.	Gecko DOM
Range	Method	createRange	createRange()	Returns a new Range object.	Level 2
Range	Method	deleteContents	deleteContents()	Removes the contents of a Range from the document.	Level 2
Range	Method	detach	detach()	Releases Range from use to improve performance.	Level 2
Range	Method	extractContents	extractContents()	Moves contents of a Range from the document tree into a document fragment	Level 2
Range	Method	insertNode	insertNode(node)	Insert a node at the start of a Range.	Level 2
Range	Method	selectNode	selectNode(node)	Sets the Range to contain the node and its contents.	Level 2
Range	Method	selectNodeContents	selectNodeContents(node)	Sets the Range to contain the contents of a Node.	Level 2
Range	Method	setEnd	setEnd(node,offset)	Sets the end position of a Range.	Level 2
Range	Method	setEndAfter	setEndAfter(node)	Sets the end position of a Range relative to another Node.	Level 2
Range	Method	setEndBefore	setEndBefore(node)	Sets the end position of a Range relative to another Node.	Level 2
Range	Method	setStart	setStart(node,offset)	Sets the start position of a Range.	Level 2
Range	Method	setStartAfter	setStartAfter(node)	Sets the start position of a Range relative to another Node.	Level 2
Range	Method	setStartBefore	setStartBefore(node)	Sets the start position of a Range relative to another Node.	Level 2
Range	Method	surroundContents	surroundContents(node)	Moves content of a Range into a new node.	Level 2
Range	Method	toString	toString()	Returns the text of the Range.	Level 2
SCRIPT	Property	defer		Sets or retrieves the status of the script.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
SCRIPT	Property	event		Sets or retrieves the event for which the script is written.	Level 1
SCRIPT	Property	htmlFor		Sets or retrieves the object to which the given label object is assigned.	Level 1
SCRIPT	Property	language		Sets or retrieves the language in which the current script is written.	Level 1
SCRIPT	Property	src		Retrieves the URL to an external file that contains the source code or data.	Level 1
SCRIPT	Property	text		Retrieves or sets the text of the object as a string.	Level 1
SELECT	Collection	options	options[]	Retrieves a collection of the option objects in a select object.	Level 2
SELECT	Property	disabled		Sets or retrieves the value that indicates whether the user can interact with the object.	Level 1
SELECT	Property	length		Sets or retrieves the number of objects in a collection.	Level 1
SELECT	Property	multiple		Sets or retrieves the Boolean value indicating whether multiple items can be selected from a list.	Level 1
SELECT	Property	size		Sets or retrieves the number of rows in the list box.	Level 1
SELECT	Property	value		Sets or retrieves the default or selected value of the control.	Level 1
SELECT	Method	item	item(i)	Retrieves an object from the all collection or various other collections.	Level 1
SELECT	Method	namedItem	namedItem("optionID")	Retrieves an object or a collection from the specified collection.	Level 1
SELECT	Method	options.remove	options[i].remove()	Removes an element from the collection.	Level 1
selection	Property	anchorNode		Retrieves the node representing one end of the selection on the web page.	Level 2
selection	Property	anchorOffset		Retrieves the offset within the (text) node where the selection begins.	Level 2
selection	Property	focusNode		Retrieves the node with the keyboard focus.	Level 2
selection	Property	focusOffset		Retrieves the offset within the (text) node where focus starts.	Level 2
selection	Property	isCollapsed		Retrieves if the selection is collapsed or not.	Level 2
selection	Property	rangeCount		Retrieves the number of ranges in the selection made on a web page.	Level 2
selection	Method	addRange	addRange(range)	Adds a range to the current selection.	Level 2
selection	Method	collapse	collapse()	Collapses the selection to a single point, at the specified offset in the given DOM node	Level 2

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
selection	Method	collapseToEnd	collapseToEnd()	Collapses the whole selection to a single point at the end of the current selection (irrespective of direction).	Level 2
selection	Method	collapseToStart	collapseToStart()	Collapses the whole selection to a single point at the start of the current selection (irrespective of direction).	Level 2
selection	Method	containsNode	containsNode(node,recurse)	Determines whether the selection on the web page contains a node.	Level 2
selection	Method	deleteFromDocument	deleteFromDocument()	Deletes the selection from document the nodes belong to.	Level 2
selection	Method	extend	extend(node,offset)	Extends the selection by moving the focus to the specified node and offset, preserving the anchor position.	Level 2
selection	Method	getRangeAt	getRangeAt()	Returns the range at the specified index from any selection made on a web page.	Level 2
selection	Method	removeAllRanges	removeAllRanges()	Removes all ranges from the current selection.	Level 2
selection	Method	removeRange	removeRange(range)	Removes a range to the current selection.	Level 2
STYLE	Property	media		Specifies the intended destination medium for style information.	Level 1
STYLE	Property	type		Retrieves the type of style being applied by this statement.	Level 1
styleSheet	Collection	cssRules	cssRules[]	Returns all of the CSS rules in the stylesheet as an array.	Level 2
styleSheet	Collection	rules	rules[]	Returns all of the CSS rules in the stylesheet as an array.	MSIE DHTML DOM
styleSheet	Property	disabled		Sets or retrieves whether a style sheet is applied to the object.	Level 2
styleSheet	Property	href		Sets or retrieves the URL of the linked style sheet.	Level 2
styleSheet	Property	media		Sets or retrieves the media type.	Level 2
styleSheet	Property	ownerNode		Returns the node that associates this style sheet with the document.	Level 2
styleSheet	Property	ownerRule		If this style sheet comes from an @import rule, the ownerRule property will contain the CSSImportRule.	Level 2
styleSheet	Property	parentStyleSheet		Retrieves the style sheet that imported the current style sheets.	Level 2
styleSheet	Property	title		Sets or retrieves the title of the style sheet.	Level 2
styleSheet	Property	type		Retrieves the Cascading Style Sheets (CSS) language in which the style sheet is written.	Level 2

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
styleSheet	Method	deleteRule	deleteRule(index)	Deletes a rule from the stylesheet.	Level 2
styleSheet	Method	insertRule	insertRule("rule",index)	Inserts a new style rule into the current style sheet.	Level 2
TABLE	Collection	rows	rows[]	Retrieves a collection of tr (table row) objects from a table object.	Level 1
TABLE	Collection	tBodies	tBodies[]	Retrieves a collection of all tBody objects in the table. Objects in this collection are in source order.	Level 1
TABLE	Property	align		Sets or retrieves a value that indicates the table alignment.	Level 1
TABLE	Property	bgColor		Sets or retrieves the background color behind the object.	Level 1
TABLE	Property	border		Sets or retrieves the width of the border to draw around the object.	Level 1
TABLE	Property	caption		Retrieves the caption object of the table.	Level 1
TABLE	Property	cellPadding		Sets or retrieves the amount of space between the border of the cell and the content of the cell.	Level 1
TABLE	Property	cellSpacing		Sets or retrieves the amount of space between cells in a table.	Level 1
TABLE	Property	frame		Sets or retrieves the way the border frame around the table is displayed.	Level 1
TABLE	Property	height		Sets or retrieves the height of the object.	Level 1
TABLE	Property	rules		Sets or retrieves which dividing lines (inner borders) are displayed.	Level 1
TABLE	Property	summary		Sets or retrieves a description and/or structure of the object.	Level 1
TABLE	Property	tFoot		Retrieves the tFoot object of the table.	Level 1
TABLE	Property	tHead		Retrieves the tHead object of the table.	Level 1
TABLE	Property	width		Sets or retrieves the width of the object.	Level 1
TABLE	Method	createCaption	createCaption()	Creates an empty caption element in the table.	Level 1
TABLE	Method	createTFoot	createTFoot()	Creates an empty tFoot element in the table.	Level 1
TABLE	Method	createTHead	createTHead()	Creates an empty tHead element in the table.	Level 1
TABLE	Method	deleteCaption	deleteCaption()	Deletes the caption element and its contents from the table.	Level 1
TABLE	Method	deleteRow	deleteRow(i)	Removes the specified row (tr) from the element and from the rows collection.	Level 1
TABLE	Method	deleteTFoot	deleteTFoot()	Deletes the tFoot element and its contents from the table.	Level 1
TABLE	Method	deleteTHead	deleteTHead()	Deletes the tHead element and its contents from the table.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
TABLE	Method	insertRow	insertRow(i)	Creates a new row (tr) in the table, and adds the row to the rows collection.	Level 1
TBODY, TFOOT, THEAD	Property	align		Sets or retrieves a value that indicates the table alignment.	Level 1
TBODY, TFOOT, THEAD	Property	bgColor		Sets or retrieves the background color behind the object.	Level 1
TBODY, TFOOT, THEAD	Property	ch		Sets or retrieves a value that you can use to implement your own ch functionality for the object.	Level 1
TBODY, TFOOT, THEAD	Property	chOff		Sets or retrieves a value that you can use to implement your own chOff functionality for the object.	Level 1
TBODY, TFOOT, THEAD	Property	rows		Sets or retrieves the number of horizontal rows contained in the object.	Level 1
TBODY, TFOOT, THEAD	Property	vAlign		Sets or retrieves how text and other content are vertically aligned within the object that contains them.	Level 1
TBODY, TFOOT, THEAD	Method	deleteRow	deleteRow(i)	Removes the specified row (tr) from the element and from the rows collection.	Level 1
TBODY, TFOOT, THEAD	Method	insertRow	insertRow(i)	Creates a new row (tr) in the table, and adds the row to the rows collection.	Level 1
TD,TH	Property	abbr		Sets or retrieves abbreviated text for the object.	Level 1
TD,TH	Property	align		Sets or retrieves the alignment of the object relative to the display or table.	Level 1
TD,TH	Property	axis		Sets or retrieves a comma-delimited list of conceptual categories associated with the object.	Level 1
TD,TH	Property	background		Sets or retrieves the background picture tiled behind the text and graphics in the object.	Level 1
TD,TH	Property	bgColor		Sets or retrieves the background color behind the object.	Level 1
TD,TH	Property	cellIndex		Retrieves the position of the object in the cells collection of a row.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
TD,TH	Property	ch		Sets or retrieves a value that you can use to implement your own ch functionality for the object.	Level 1
TD,TH	Property	chOff		Sets or retrieves a value that you can use to implement your own chOff functionality for the object.	Level 1
TD,TH	Property	colSpan		Sets or retrieves the number columns in the table that the object should span.	Level 1
TD,TH	Property	headers		Sets or retrieves a list of header cells that provide information for the object.	Level 1
TD,TH	Property	height		Sets or retrieves the height of the object.	Level 1
TD,TH	Property	noWrap		Sets or retrieves whether the browser automatically performs wordwrap.	Level 1
TD,TH	Property	rowSpan		Sets or retrieves how many rows in a table the cell should span.	Level 1
TD,TH	Property	vAlign		Sets or retrieves how text and other content are vertically aligned within the object that contains them.	Level 1
TD,TH	Property	width		Sets or retrieves the width of the object.	Level 1
Text	Property	data		Sets or retrieves the value of a TextNode object.	Level 1
Text	Property	length		Retrieves the number of characters in a TextNode object.	Level 1
Text	Method	appendChild	appendChild(node)	Appends an element as a child to the object.	Level 1
Text	Method	appendData	appendData("text")	Adds a new character string to the end of the object.	Level 1
Text	Method	cloneNode	cloneNode(deep)	Copies a reference to the object from the document hierarchy.	Level 1
Text	Method	deleteData	deleteData(offset,count)	Removes a specified range of characters from the object.	Level 1
Text	Method	hasChildNodes	hasChildNodes()	Returns a value that indicates whether the object has children.	Level 1
Text	Method	insertBefore	insertBefore(new,"text")	Inserts an element into the document hierarchy as a child node of a parent object.	Level 1
Text	Method	insertData	insertData(offset,"text")	Inserts a new character string in the object at a specified offset.	Level 1
Text	Method	normalize	normalize()	Merges adjacent TextNode objects to produce a normalized document object model.	Level 1
Text	Method	removeChild	removeChild()	Removes a child node from the object.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
Text	Method	replaceChild	replaceChild(offset,count,"text")	Replaces an existing child element with a new child element.	Level 1
Text	Method	splitText	splitText(offset)	Divides a text node at the specified index.	Level 1
Text	Method	substringData	substringData(offset,count)	Extracts a range of characters from the object.	Level 1
TEXTAREA	Property	cols		Sets or retrieves the width of the object.	Level 1
TEXTAREA	Property	disabled		Sets or retrieves the value that indicates whether the user can interact with the object.	Level 1
TEXTAREA	Property	readOnly		Sets or retrieves the value indicated whether the content of the object is read-only.	Level 1
TEXTAREA	Property	rows		Sets or retrieves the number of horizontal rows contained in the object.	Level 1
TITLE	Property	text		Retrieves or sets the text of the object as a string.	Level 1
TR	Collection	cells	cells[]	Retrieves a collection of all cells in the table row or in the entire table.	Level 1
TR	Property	align		Sets or retrieves the alignment of the object relative to the display or table.	Level 1
TR	Property	bgColor		Sets or retrieves the background color behind the object.	Level 1
TR	Property	ch		Sets or retrieves a value that you can use to implement your own ch functionality for the object.	Level 1
TR	Property	chOff		Sets or retrieves a value that you can use to implement your own chOff functionality for the object.	Level 1
TR	Property	rowIndex		Retrieves the position of the object in the rows collection for the table.	Level 1
TR	Property	vAlign		Sets or retrieves how text and other content are vertically aligned within the object that contains them.	Level 1
TR	Method	deleteCell	deleteCell(i)	Removes the specified cell (td) from the table row, as well as from the cells collection.	Level 1

Elément HTML associé	Type	Nom	Syntaxe	Description	Version DOM
TR	Method	insertCell	insertCell(i)	Creates a new cell in the table row (tr), and adds the cell to the cells collection.	Level 1
UL	Property	compact		Sets or retrieves a Boolean value indicating whether the list should be compacted by removing extra space between list objects.	Level 1
UL	Property	type		Retrieves the type of list.	Level 1

ANNEXE H : REFERENCES DES EVENEMENTS JS

Le tableau ci-dessous indique les événements et les versions de leur introduction (Version de JavaScript correspond au DOM Level 0 et DOM W3C son élargissement).

Les lignes surlignées en rouge interdisent leur implémentation pour éviter tout problème de compatibilité et de fiabilité.

Les lignes surlignées en orange demandent une autorisation précise avant toute utilisation du fait de leur problème de fiabilité suivant leur contexte d'utilisation

Ce tableau ne référence pas les événements non compatibles propriétaires à un éditeur qui sont interdits d'utilisation :

Microsoft DHTML DOM : onActivate, onAfterPrint, onAfterUpdate, onBeforeActivate, onBeforeCopy, onBeforeCut, onBeforeDeactivate, onBeforeEditFocus, onBeforePaste, onBeforePrint, onBeforePrint, onBeforeUnload, onBeforeUpdate, onBounce, onCellChange, onContextMenu, onControlSelect, onCopy, onCut, onDataAvailable, onDataSetChanged, onDataSetComplete, onDeactivate, onDrag, onDragEnd, onDragEnter, onDragLeave, onDragOver, onDragStart, onDrop, onErrorUpdate, onFilterChange, onFinish, onFocusIn, onFocusOut, onHelp, onLayoutComplete, onLoseCapture, onMouseEnter, onMouseLeave, onMouseWheel, onMoveEnd, onMoveStart, onPaste, onPropertyChange, onReadyStateChange, onResizeEnd, onResizeStart, onRowEnter, onRowExit, onRowsDelete, onRowsInserted, onSelectionChange, onSelectStart,onStart, onStop.

Netscape 6 Original DOM : onAfterPrint, onBeforePrint, onBeforeUnload, onClose.

Gecko DOM : onAttrModified, onBroadcast, onCharacterDataModified, onClose, onCommand, onCommandUpdate, onDragEnter, onDragOver, onDragExit, onDragEsture, onInput, onOverFlowChanged, onNodeInserted, onNodeRemoved, onNodeRemovedFromDocument, onNodeInsertedIntoDocument, onPaint, onPopupShowing, onPopupShown, onPopupHidding, onPopupHidden, onSubtreeModified, onText, onUnderFlow.

Elément HTML associé	Nom	Description	Version
Image	onAbort	Executes JavaScript code when an abort event occurs; that is, when the user aborts the loading of an image (for example by clicking a link or clicking the Stop button).	JavaScript 1.1 : Image DOM W3C : Idem
All	onBlur	Executes JavaScript code when a blur event occurs; that is, when a form element loses focus or when a window or frame loses focus.	JavaScript 1.0 JavaScript 1.1: event handler of Button, Checkbox, Frame, Password, Radio, Reset, Submit, and window DOM W3C : All HTML Elements

Elément HTML associé	Nom	Description	Version
FileUpload, Select, Text, Textarea	onChange	Executes JavaScript code when a change event occurs; that is, when a Select, Text, or Textarea field loses focus and its value has been modified.	JavaScript 1.0 : event handler for Select, Text, and Textarea JavaScript 1.1 : added as event handler of FileUpload
All	onClick	Executes JavaScript code when a click event occurs; that is, when an object on a form is clicked. (A click event is a combination of the MouseDown and MouseUp events).	JavaScript 1.0 : Button, document, Checkbox, Link, Radio, Reset, Submit JavaScript 1.1 : added the ability to return false to cancel the action associated with a click event DOM W3C : All HTML Elements
All	onDblClick	Executes JavaScript code when a DblClick event occurs; that is, when the user double-clicks a form element or a link.	JavaScript 1.2 : document, Link DOM W3C : All HTML Elements
window	onDragDrop	Executes JavaScript code when a DragDrop event occurs; that is, when the user drops an object onto the browser window, such as dropping a file.	JavaScript 1.2
Image, window	onError	Executes JavaScript code when an error event occurs; that is, when the loading of a document or image causes an error.	JavaScript 1.1
All	onFocus	Executes JavaScript code when a focus event occurs; that is, when a window, frame, or frameset receives focus or when a form element receives input focus.	JavaScript 1.0 : Select, Text and Textarea JavaScript 1.1 : event handler of Button, Checkbox, FileUpload, Frame, Password, Radio, Reset, Submit, and window JavaScript 1.2 : event handler of Layer DOM W3C : All HTML Elements sauf Layer
All	onKeyDown	Executes JavaScript code when a KeyDown event occurs; that is, when the user depresses a key.	JavaScript 1.2 : document, Image, Link, Textarea DOM W3C : All HTML Elements
All	onKeyPress	Executes JavaScript code when a KeyPress event occurs; that is, when the user presses or holds down a key.	JavaScript 1.2 : document, Image, Link, Textarea DOM W3C : All HTML Elements

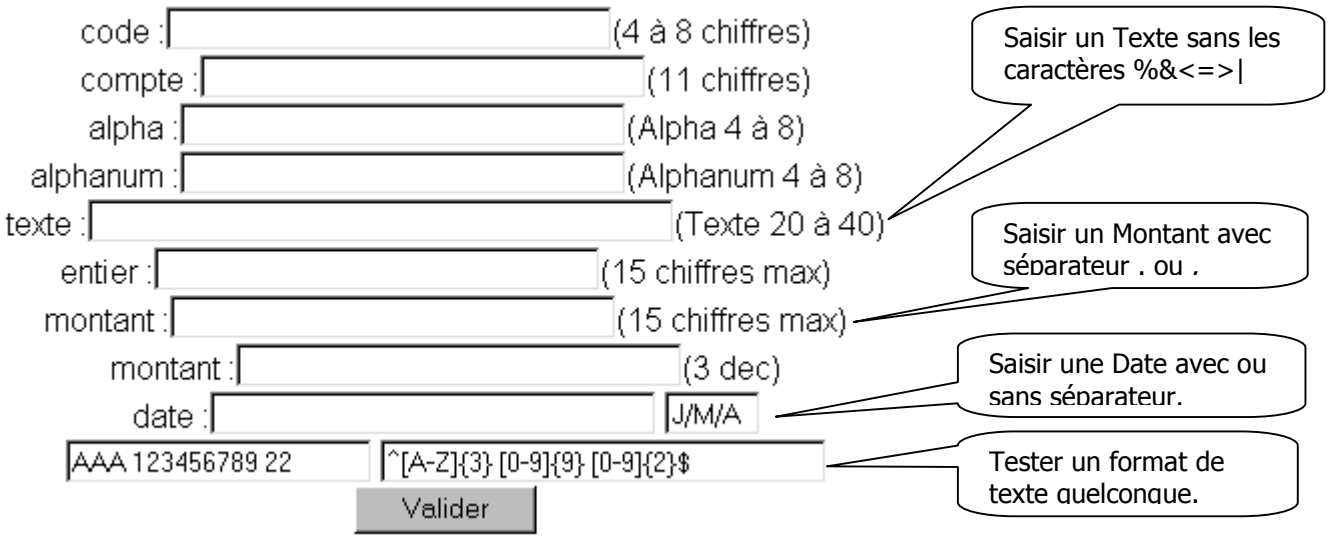
Elément HTML associé	Nom	Description	Version
All	onKeyUp	Executes JavaScript code when a KeyUp event occurs; that is, when the user releases a key.	JavaScript 1.2 : document, Image, Link, Textarea DOM W3C : All HTML Elements
Image, Layer, window, frame, frameset, iframe	onLoad	Executes JavaScript code when a load event occurs; that is, when the browser finishes loading a window or all frames within a FRAMESET tag.	JavaScript 1.0 : window JavaScript 1.1: event handler of Image JavaScript 1.2: event handler of Layer DOM W3C : Idem sauf Layer
All	onMouseDown	Executes JavaScript code when a MouseDown event occurs; that is, when the user depresses a mouse button.	JavaScript 1.2 : Button, document, Link DOM W3C : All HTML Elements
All	onMouseMove	Executes JavaScript code when a MouseMove event occurs; that is, when the user moves the cursor.	JavaScript 1.2 : Link DOM W3C : All HTML Elements
All	onMouseOut	Executes JavaScript code when a MouseOut event occurs; that is, each time the mouse pointer leaves an area (client-side image map) or link from inside that area or link.	JavaScript 1.1 : Link DOM W3C : All HTML Elements
All	onMouseOver	Executes JavaScript code when a MouseOver event occurs; that is, once each time the mouse pointer moves over an object or area from outside that object or area.	JavaScript 1.0 : Link JavaScript 1.1 : event handler of Area DOM W3C : All HTML Elements
All	onMouseUp	Executes JavaScript code when a MouseUp event occurs; that is, when the user releases a mouse button.	JavaScript 1.2 : Button, document, Link DOM W3C : All HTML Elements
window	onMove	Executes JavaScript code when a move event occurs; that is, when the user or script moves a window or frame.	JavaScript 1.2 : window
Form	onReset	Executes JavaScript code when a reset event occurs; that is, when a user resets a form (clicks a Reset button).	JavaScript 1.1 : Form DOM W3C : Idem
All	onResize	Executes JavaScript code when a resize event occurs; that is, when a user or script resizes a window or frame.	JavaScript 1.2 : window DOM W3C : All HTML Elements
DIV, Table, Textarea, window	onScroll	Fires when the user repositions the scroll box in the scroll bar on the object.	DOM W3C : DIV, Table, Textarea, window
Text, Textarea	onSelect	Executes JavaScript code when a select event occurs; that is, when a user selects some of the text within a text or textarea field.	JavaScript 1.0 : Text, Textarea DOM W3C : Idem

Elément HTML associé	Nom	Description	Version
Form	onSubmit	Executes JavaScript code when a submit event occurs; that is, when a user submits a form.	JavaScript 1.0 : Form DOM W3C : Idem
Window	onUnload	Executes JavaScript code when an unload event occurs; that is, when the user exits a document.	JavaScript 1.0 : window DOM W3C : Idem

ANNEXE I : EXEMPLES D'APPLICATION DU FRAMEWORK POUR GERER UN FORMULAIRE

Cette annexe décrit un exemple d'application des notions et du framework de contrôles de surface détaillés dans le présent document.

Vous pouvez implémenter cette page illustrant l'utilisation du framework des contrôles de surface :



L'implémentation de cette page :

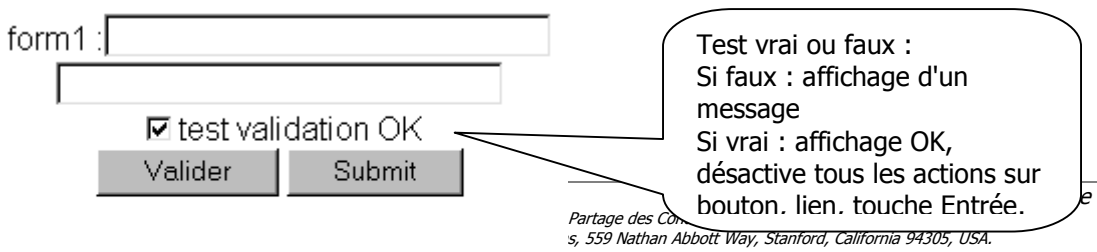
```
<html>
<head>
<script type="text/javascript" src="fw_cs.js"></script>
<script type="text/javascript">
function valider()
{
  var ret = true;
  var f = document.form0;
  if(ret) ret = ctrlSaisie(f.code,"","4 à 8 chiffres","",false, estNumString, 4, 8);
  if(ret) ret = ctrlSaisie(f.compte,"","11 chiffres","",true, estCompteString);
  if(ret) ret = ctrlSaisie(f.alpha,"","Alpha 4 à 8 car","",false, estAlphaString, 4,
8);
  if(ret) ret = ctrlSaisie(f.alphanum,"","Alphanum 4 à 8 car","",false,
estAlphanumString, 4, 8);
  if(ret) ret = ctrlSaisie(f.txt,"","Saisir un texte entre 20 à 40 car, les car
'%&<=>|' sont interdits",false,false, estTexteString, 20, 40);
  if(ret) ret = ctrlSaisie(f.entier,"vide","tout entier avec 15 chiffres
max",false,true, estEntierString);
  if(ret) ret = ctrlSaisie(f.montant,"vide","tout montant avec 15 chiffres
significatifs max",false,true, estMontantString);
  if(ret) ret = ctrlSaisie(f.montant2,"vide","montant entre -20.001 et 1000000.005
avec 3 decimal",false,true, estMontantString, -20.001, 1000000.005, 3,",");
  if(ret) ret = ctrlSaisie(f.date,"vide","date 01/01/2002 à 31/12/2002",false,true,
estDateString,f.dateFrm.value,(new Date(2002,1 -1,1)),(new Date(2002,12 -1,31)));
  if(ret) ret = ctrlSaisie(f.dateFrm,"vide","format date
?","J/M/A",false,estFormatString,"^J/M/A|A/M/J|M/A/J$");
```

```
  if(ret) ret = ctrlSaisie(f.autre,"vide","format ?","AAA 123456789
22",false,estFormatString,f.autreFrm.value);
  if(ret) alert("validation OK");
}
</script>
</head>
<body>
<center>
<h1> TEST </h1>

<form name=form0>
<br>code :<input type=text name=code value="" size=30>(4 à 8 chiffres)
<br>compte :<input type=text name=compte value="" size=30>(11 chiffres)
<br>alpha :<input type=text name=alpha value="" size=30>(Alpha 4 à 8)
<br>alphanum :<input type=text name=alphanum value="" size=30>(Alphanum 4 à 8)
<br>texte :<input type=text name=txt value="" size=40>(Texte 20 à 40)
<br>entier :<input type=text name=entier value="" size=30>(15 chiffres max)
<br>montant :<input type=text name=montant value="" size=30>(15 chiffres max)
<br>montant :<input type=text name=montant2 value="" size=30>(3 dec)
<br>date :<input type=text name=date value="" size=30>
<input type=text name=dateFrm value="J/M/A" size=5>
<br><input type=text name=autre value="AAA 123456789 22" size=20>
<input type=text name=autreFrm value="^[A-Z]{3} [0-9]{9} [0-9]{2}$" size=30>
<br><input type="button" value="   Valider   " onClick="javascript:valider()">
</form>

</body>
</html>
```

Vous pouvez aussi implémenter cette page illustrant les notions décrites dans le présent document lors de la validation d'un formulaire :



L'implémentation de cette page :

```
<html>
<head>
<script type="text/javascript" src="fw_form.js"></script>
<script type="text/javascript">
document.onverif = false;
function valider()
{
  if(! document.onverif)
  { document.onverif = true;
    if(document.forms[0].chk.checked)
    {
      alert("validation OK !");
      disableAllAction();
      //document.forms[0].submit();
    }
    else
    alert("validation KO !");
    document.onverif = false;
  }
}
</script>
</head>
<body>
<center>
<h1> TEST </h1>

<form name="form1" action="jsc_ctrl_saisie_teste.html">
```

```
<br>form1 :<input type="text" name="nom" value="" size="30">
<br><input type="password" name="code" value="" size="30">
<br><input type="checkbox" name="chk" checked>test validation OK
<br><input type="button" value="   Valider   " name="coucou"
onClick="javascript:valider()">
<input type="submit" value="   Submit   ">
</form>

<form name="form2" action="jsc_ctrl_saisie_teste.html">
<br>form2 :<input type="text" name="nom" value="" size="30">
<br><input type="password" name="code" value="" size="30">
<br><input type="checkbox" name="chk" checked>simuler 2 seconde
<input type="image" src="http://www.ddd.com/ddd1.gif" alt="Image">
<br><input type="button" value="   Valider   " name="coucou"
onClick="javascript:valider()">
</form>

<form name="form3" action="jsc_ctrl_saisie_teste.html">
<br>form3 :<input type="text" name="nom" value="" size="30">
<br><input type="password" name="code" value="" size="30">
<br><input type="checkbox" name="chk" checked>coucou
</form>

<form name="form4" action="jsc_ctrl_saisie_teste.html">
<br>form4 :<input type="password" name="nom" value="" size="30">
</form>

<a name="l1" href="jsc_ctrl_saisie_teste.html?x=1">teste 1</a>
<a name="l2" href="jsc_ctrl_saisie_teste.html?x=2">teste 2</a>
<br>
<a name="l3" href="jsc_ctrl_saisie_teste.html?x=3">teste 3</a>
<a name="coucou" href="javascript:valider()">Valider</a>
<br>
<script type="text/javascript">
  initForm(valider);
</script>
</body>
</html>
```

ANNEXE J : COMPARATIF INNERHTML VS DOM W3C

Sur le site http://www.xs4all.nl/~ppk/js/index.html?version5.html, un scénario de test de génération d'un tableau de taille 50x50 est proposé suivant différentes méthodes de génération W3C et innerHTML.

Nous avons appliqué sur un prototype la méthode 4 du innerHTML, les résultats de ce scénario sur nos différentes plateformes de test (IE6.0 Win., IE5.5 Win., IE5.0 Win., IE5.1 Mac., Moz. 1.1, Opéra 7.0 bêta, NS 7.0 Prv. 1) sont décrits ci-dessous et semblent confirmer ce choix en terme de performance et de compatibilité (sauf pour Opéra 7.0).

Each of the four links in the table generates a new, huge table of 50 x 50 TD's. The questions before the house today is: which method is faster?

I use five techniques, three using the pure W3C DOM and two using innerHTML.

1. In the first W3C DOM example I create a table outside the document. Then I append a TBODY and the various TR's and TD's. At the end of the script I append the TABLE to the document.
2. The second W3C DOM example is the same, except that I append the TABLE to the document immediately after I create it.
3. In the third W3C DOM example I don't use intermediary variables.

```
Not var a = document.createElement('TR');
x.appendChild(a);
but
x.appendChild(document.createElement('TR'));
```

4. In the first innerHTML example I concatenate a huge string which at the end is written into the innerHTML of the document.

In the second innerHTML example I `push` small strings (`'</td>'` and such) into an array. At the end I join this array to one string and write it into the innerHTML of the document.

Test sur IE 6.0 Windows :

Test it	Method	Time	Average time	Index	Browsers
1)	W3C DOM, append table after building its tree.	3130 ms	3130 ms	569	This is the slowest method in Explorer 6 Windows (6.5 times slower)
2)	W3C DOM, append table immediately.	2910 ms	2910 ms	529	Overall this is the slowest method: Opera 7 (7 times slower) Mozilla 1.1 (2.5 times slower) Explorer 5 Mac (nearly 100 times) Konqueror (10 times slower)
3)	W3C DOM, don't use intermediary variables.	1640 ms	1640 ms	298	This is the fastest method in **Opera 7** and **Konqueror**.
4)	innerHTML with concatenation	940 ms	940 ms	171	This is the fastest method in **Mozilla 1.1** and **Explorer 5 Mac**

5)	innerHTML with push	550 ms	550 ms	100 **Fastest method**	This is the fastest method in **Explorer 6 Windows**. `push()` is not supported by Explorer 5.0 Windows and 5.x Mac.

Test sur IE 5.5 Windows :

Test it	Method	Time	Average time	Index	Browsers
1)	W3C DOM, append table after building its tree.	1700 ms	1700 ms	340	This is the slowest method in Explorer 6 Windows (6.5 times slower)
2)	W3C DOM, append table immediately.	1590 ms	1590 ms	318	Overall this is the slowest method: Opera 7 (7 times slower) Mozilla 1.1 (2.5 times slower) Explorer 5 Mac (nearly 100 times) Konqueror (10 times slower)
3)	W3C DOM, don't use intermediary variables.	990 ms	990 ms	198	This is the fastest method in **Opera 7** and **Konqueror**.
4)	innerHTML with concatenation	710 ms	710 ms	142	This is the fastest method in **Mozilla 1.1** and **Explorer 5 Mac**
5)	innerHTML with push	500 ms	500 ms	100 **Fastest method**	This is the fastest method in **Explorer 6 Windows**. `push()` is not supported by Explorer 5.0 Windows and 5.x Mac.

Test sur IE 5.0 Windows :

Test it	Method	Time	Average time	Index	Browsers
1)	W3C DOM, append table after building its tree.	1100 ms	1100 ms	153	This is the slowest method in Explorer 6 Windows (6.5 times slower)
2)	W3C DOM, append table immediately.	1150 ms	1150 ms	160	Overall this is the slowest method: Opera 7 (7 times slower) Mozilla 1.1 (2.5 times slower) Explorer 5 Mac (nearly 100 times) Konqueror (10 times slower)
3)	W3C DOM, don't use intermediary variables.	830 ms	830 ms	115	This is the fastest method in **Opera 7** and **Konqueror**.

4)	innerHTML with concatenation	720 ms	720 ms	100 **Fastest method**	This is the fastest method in **Mozilla 1.1** and **Explorer 5 Mac**
5)	innerHTML with push	Not Supported	Not Supported	Not Supported	This is the fastest method in **Explorer 6 Windows**. `push()` is not supported by Explorer 5.0 Windows and 5.x Mac.

Test sur IE 5.1 Mac :

Test it	Method	Time	Average time	Index	Browsers
1)	W3C DOM, append table after building its tree.	95082 ms	95082 ms	161	This is the slowest method in Explorer 6 Windows (6.5 times slower)
2)	W3C DOM, append table immediately.	277111 ms	277111 ms	468	Overall this is the slowest method: Opera 7 (7 times slower) Mozilla 1.1 (2.5 times slower) Explorer 5 Mac (nearly 100 times) Konqueror (10 times slower)
3)	W3C DOM, don't use intermediary variables.	262077 ms	262077 ms	443	This is the fastest method in **Opera 7** and **Konqueror**.
4)	innerHTML with concatenation	59196 ms	59196 ms	100 **Fastest method**	This is the fastest method in **Mozilla 1.1** and **Explorer 5 Mac**
5)	innerHTML with push	Not Supported	Not Supported	Not Supported	This is the fastest method in **Explorer 6 Windows**. `push()` is not supported by Explorer 5.0 Windows and 5.x Mac.

Test sur Mozilla 1.1 :

Test it	Method	Time	Average time	Index	Browsers
1)	W3C DOM, append table after building its tree.	1920 ms	1920 ms	231	This is the slowest method in Explorer 6 Windows (6.5 times slower)
2)	W3C DOM, append table	1920	1920 ms	231	Overall this is the slowest method:

	immediately.	ms			Opera 7 (7 times slower) Mozilla 1.1 (2.5 times slower) Explorer 5 Mac (nearly 100 times) Konqueror (10 times slower)
3)	W3C DOM, don't use intermediary variables.	1200 ms	1200 ms	145	This is the fastest method in **Opera 7** and **Konqueror**.
4)	innerHTML with concatenation	830 ms	830 ms	100 **Fastest method**	This is the fastest method in **Mozilla 1.1** and **Explorer 5 Mac**
5)	innerHTML with push	990 ms	990 ms	119	This is the fastest method in **Explorer 6 Windows**. `push()` is not supported by Explorer 5.0 Windows and 5.x Mac.

Test sur NS 7.0 prv 1 :

Test it	Method	Time	Average time	Index	Browsers
1)	W3C DOM, append table after building its tree.	1590 ms	1590 ms	241	This is the slowest method in Explorer 6 Windows (6.5 times slower)
2)	W3C DOM, append table immediately.	1590 ms	1590 ms	241	Overall this is the slowest method: Opera 7 (7 times slower) Mozilla 1.1 (2.5 times slower) Explorer 5 Mac (nearly 100 times) Konqueror (10 times slower)
3)	W3C DOM, don't use intermediary variables.	930 ms	930 ms	141	This is the fastest method in **Opera 7** and **Konqueror**.
4)	innerHTML with concatenation	660 ms	660 ms	100 **Fastest method**	This is the fastest method in **Mozilla 1.1** and **Explorer 5 Mac**
5)	innerHTML with push	760 ms	760 ms	115	This is the fastest method in **Explorer 6 Windows**. `push()` is not supported by Explorer 5.0 Windows and 5.x Mac.

Test sur Opéra 7.0 bêta :

Test it	Method	Time	Average time	Index	Browsers

1)	W3C DOM, append table after building its tree.	1210 ms	1210 ms	186	This is the slowest method in Explorer 6 Windows (6.5 times slower)
2)	W3C DOM, append table immediately.	7080 ms	7080 ms	1089	Overall this is the slowest method: Opera 7 (7 times slower) Mozilla 1.1 (2.5 times slower) Explorer 5 Mac (nearly 100 times) Konqueror (10 times slower)
3)	W3C DOM, don't use intermediary variables.	650 ms	650 ms	100 **Fastest method**	This is the fastest method in **Opera 7** and **Konqueror**.
4)	innerHTML with concatenation	3460 ms	3460 ms	532	This is the fastest method in **Mozilla 1.1** and **Explorer 5 Mac**
5)	innerHTML with push	2690 ms	2690 ms	414	This is the fastest method in **Explorer 6 Windows**. `push()` is not supported by Explorer 5.0 Windows and 5.x Mac.

ANNEXE K : GESTION DU CACHE

La gestion du cache est basée sur la norme HTTP RFC-2616 pour la gestion des en-têtes HTTP.

Elle répond à deux besoins nécessaires sur l'applicatif :

- spécifier sur une page qu'elle soit déclarée comme non cachable.
- spécifier une date de validation de la page dans le cache pour forcer le rechargement de celle-ci quand elle est expirée.

L'implémentation de ces besoins différent suivant la portabilité désirée de la gestion du cache :

- pour le cache du navigateur ou pour les caches des différents proxies traversés par une requête cliente.
- pour une requête cliente en HTTP 1.0 ou HTTP 1.1.

Ces spécificités sont décrites dans les paragraphes ci-dessous suivant l'emplacement des caches concernés.

1 VIS A VIS DES NAVIGATEURS

L'utilisation des balises <META HTTP-EQUIV> dans l'en-tête du document HTML suffisent pour déclarer une page non cachable.

Pour gérer le cache d'un navigateur en HTTP 1.0, il est nécessaire de déclarer en en-tête un Pragma:no-cache et un « Expires:Date en cours en GMT » La valeur « 0 » pour l'Expires permet au navigateur de spécifier cette date actuelle par lui même (Attention la valeur « –1 » est aussi autorisée mais elle n'est pas normalisée par le RFC) et ainsi de déclarer une date de validité expirée à chaque nouvel appel de cette page.

Pour gérer le cache d'un navigateur en HTTP 1.1, il est nécessaire de déclarer en en-tête un Cache-Control :no-cache pour que cette page ne soit pas prise en compte par le cache du navigateur.

Implémentation HTML pour déclarer une page non cachable :

```
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="0">
```

Implémentation HTML pour déclarer une page expirant à une certaine date :

```
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="Date GMT de validité">
```

2 VIS A VIS DES PROXY

Les caches des différents proxies traversés par une requête cliente n'interprètent pas le code HTML des pages renvoyées. Il est donc impossible de se suffire des balises <META HTTP-EQUIV> pour gérer ce type de cache.

Il est donc nécessaire de modifier directement sur le serveur les en-têtes HTTP de la requête de réponse du serveur.

Les valorisations nécessaires de ces en-têtes pour déclarer une page non cachable :

En HTTP 1.0 :

```
Pragma: no-cache
Expires: Date GMT en cours sur le serveur – 10s
```

En HTTP 1.1 :

```
Cache-Control: private,no-cache,no-store,max-age=0
```

Les valorisations nécessaires de ces en-têtes pour déclarer une page devant expirer dans un delay exprimé en seconde :

En HTTP 1.0 :

```
Expires: Date GMT en cours sur le serveur + delay
```

En HTTP 1.1 :

```
Cache-Control: public,max-age=delay,s-maxage=delay
```

Les calculs de date pour l'Expires s'effectuent sur le serveur à partir de la date en cours système et seront comparés avec cette même date par les proxies car lors du renvoi de la réponse le serveur spécifie cette date dans l'en-tête : « Date: Date en cours du serveur en GMT ».

ANNEXE L : BIBLIOGRAPHIE

Pour réaliser ces normes, les rédacteurs de ce document se sont inspirés des sites suivants :

http://www.allhtml.com

http://www.xhtml.net

http://www.openweb.eu.org

http://www.w3.org

http://www.quirksmode.org

http://www.w3schools.com

http://www.laltruiste.com/

Les ouvrages suivants restent incontournables pour entrer plus dans le détail des langages décrits :

HTML & XHTML - La référence

Chuck Musciano & Bill Kennedy

O'Reilly, 4e édition, février 2001

ISBN : 2-84177-132-6

Core CSS Cascading Style Sheet

K.Schengili-Roberts

Prentice Hall, 2e édition, octobre 2003

ISBN : 0-13-009278-9

Javascript Bible

Danny Goodman& Michael Morrison

Wiley Publishing, 4e édition, avril 2000

ISBN : 0-7645-3342-8