# Annexes

## Annexe A

**Bien baliser votre contenu : XHTML sémantique**

## Annexe B

**Aspect irréprochable et flexible : CSS 2.1**

## Annexe C

**Le « plus » de l'expert : savoir lire une spécification**

## Annexe D

**Développer avec son navigateur web**

## Annexe E

**Tour d'horizon des autres frameworks JavaScript**

# A

# Bien baliser votre contenu : XHTML sémantique

Au commencement était le contenu. La toute première étape de la création de votre page web doit être ce contenu. Il n'y a pas plus important. C'est lui qui fait la différence entre votre page et les autres. C'est ce contenu que voient avant tout les moteurs de recherche et les périphériques alternatifs. C'est lui qui définit la valeur, le sens et l'identité de votre page. Vous avez le devoir, je dis bien le devoir, de lui consacrer tout le soin possible.

Ce n'est qu'une fois votre contenu défini et mis en place que vous pouvez vous poser la question de son aspect et de son comportement. L'aspect sera le rôle des CSS, et nous verrons à l'annexe B que leur usage ne nécessite qu'extrêmement peu d'intrusions dans votre contenu pour satisfaire vos envies les plus variées. Quant au comportement, ce sera le rôle de JavaScript, et si vous devez retenir une chose des chapitres 2 et 3, c'est que son utilisation n'entache pas non plus votre contenu.

Cette annexe commence par présenter rapidement les nombreux bénéfices qu'apporte la présence de contenus « propres » dans vos pages web, c'est-à-dire, pour l'essentiel, des contenus valides et surtout sémantiques. Ensuite, vous trouverez un court passage en revue des contraintes syntaxiques, des éléments disponibles et de leur utilisation pertinente, pour finir par quelques exemples concrets.

# Les avantages insoupçonnés

Prendre les bonnes habitudes pour créer un contenu de qualité ne rend pas le travail plus difficile (bien au contraire), mais apporte de nombreux bénéfices, parfois inattendus.

## Pour le site et ses propriétaires

En tant que commanditaire d'un site web, qu'avez-vous à gagner à ce qu'il soit réalisé en XHTML 1 Strict, et à ce qu'il utilise un balisage rigoureusement sémantique ?

Beaucoup de choses :

1. Il sera infiniment plus compréhensible par les moteurs d'indexation et de recherche. Par conséquent, votre visibilité sur Google et autres, mais surtout la pertinence de cette visibilité, seront grandement améliorées.
2. Le balisage sémantique entraîne automatiquement une accessibilité accrue, ce qui signifie que votre site est consultable par un plus large public.
   Il s'agit d'abord des internautes atteints d'un handicap (visuel, moteur, cognitif), ce qui vous met en conformité avec les lois présentes ou à venir quant à l'accès aux sites web pour ces personnes.
   Il s'agit aussi des visiteurs utilisant autre chose qu'un ordinateur avec un grand écran, un clavier et une souris : ceux navigant sur votre site à l'aide d'un Palm, d'un PocketPC, d'un TabletPC, d'un téléphone 3G, d'un Blackberry, etc., ont par nécessité une préférence marquée pour les sites accessibles. Ils constituent qui plus est une cible marketing intéressante (adeptes de haute technologie et disposant de moyens financiers suffisants pour se les acheter).
3. Un tel site « pèse » beaucoup moins lourd en termes d'octets. D'innombrables refontes complètes de sites appliquant ces préceptes ont abouti à un contenu tout aussi riche mais beaucoup moins lourd. Conséquence directe : la consommation de bande passante (et donc son coût) diminue sensiblement, alors même que la fréquentation augmente (et donc les revenus aussi).
4. La fréquentation du site augmente, car il n'est plus nécessaire de maintenir différentes versions du contenu : l'utilisation des CSS permet de s'adapter automatiquement d'un périphérique de consultation à l'autre. Oublié, le temps où les versions alternatives (pour impression ou pour non-voyants) étaient souvent en retard sur la version principale ! Non seulement votre contenu est à jour pour tous, mais il devient accessible à une base plus large d'utilisateurs. Par ailleurs, cette centralisation du contenu signifie que sa mise à jour est plus simple et plus rapide : vous gagnez en parts, mais aussi en réactivité !
   Ce cercle vertueux ne peut que générer des revenus supplémentaires, qui ont tôt fait d'amortir le coût d'une éventuelle refonte complète du site.

Vous retrouverez ces arguments sous un développement différent dans la section « À quoi servent les standards du Web ? » de l'avant-propos. Il contient notamment un lien vers l'exemple célèbre du site de la chaîne de sports américaine ESPN, qui constitue un cas d'école magistral, avec des chiffres impressionnants à l'appui.

### Pour le développeur web

Les développeurs chargés d'implémenter un site respectant ces principes de qualité ont aussi la vie plus facile.

1. Un document XHTML ne présente aucune ambiguïté, par comparaison à un document HTML classique. En y plaçant la bonne déclaration `DOCTYPE`, on peut tirer parti des validateurs en ligne pour vérifier la conformité du document à sa grammaire imposée, sans risque d'erreur. Des validateurs identiques existent pour les CSS, par exemple.
   Si les propriétaires du site imposent des règles éditoriales qui affectent la grammaire du document, il est possible de réaliser une DTD ou un schéma transcrivant ces contraintes pour bénéficier des mêmes tests automatisés de conformité.
2. Un des avantages de l'univers XML est qu'il permet, grâce au mécanisme des espaces de noms, de mélanger plusieurs vocabulaires dans un même document. Ainsi, en XHTML, on peut insérer des fragments de langages à balises supplémentaires, comme SVG pour les graphiques vectoriels, MathML pour les formules mathématiques ou encore SMIL pour le multimédia.
3. Un document sémantique a par définition beaucoup moins de balises qu'un document des années 1990, et beaucoup plus d'attributs `id` correctement définis : cela le rend plus simple à habiller avec les CSS, et plus simple à scripter en terme de DOM.

## Règles syntaxiques et sémantiques

XHTML 1 Strict est la variante stricte de XHTML. XHTML lui-même peut se résumer à HTML 4.01 auquel on applique les règles syntaxiques de XML. Ces distinctions sont signalées dans la section 4 de la recommandation pour XHTML 1.0, et elles sont très simples.

- **Les documents doivent être correctement formés.** Cela signifie essentiellement que les balises doivent se fermer dans l'ordre inverse de leur ouverture : `<b><i>…</i>…</b>` est correct, mais `<b><i>…</b>…</i>` ne l'est pas. Les balises ne sont pas comme des commutateurs, mais constituent véritablement des conteneurs : on doit pouvoir déterminer quel élément est dans quel autre.

- **Par extension, tout élément doit être fermé.** En HTML, de nombreux éléments pourtant non vides n'étaient pas fermés par négligence : principalement `li` et `option`, mais aussi trop souvent `p`. On ferme un élément avec sa balise classique précédée d'un `/` : `<p>` devient `</p>`.
- **Les noms d'éléments et d'attributs doivent être en minuscules.** En XML, on est sensible à la casse, et les grammaires formelles pour XHTML utilisent les minuscules (c'est par ailleurs plus agréable visuellement).
- **Les éléments vides doivent aussi être fermés.** Il s'agit des éléments `area`, `base`, `br`, `col`, `frame`, `hr`, `img`, `input`, `link`, `meta` et `param`. Pour fermer un élément vide en XML, on ajoute simplement un `/` avant son chevron fermant. Toutefois, certains vieux navigateurs sont perturbés par cette notation ; aussi prend-on généralement la précaution d'ajouter une espace devant le `/`, comme ceci :
  `<img src="logo.png" />`.
- **Les valeurs d'attributs sont entre guillemets.** Techniquement, XML autorise tant les apostrophes (`'`) que les guillemets (`"`). Je recommande vivement, à titre personnel, la deuxième possibilité.
- **Les attributs bascules ont tout de même une valeur.** En HTML, certains attributs sont obligatoires, mais sans qu'il soit nécessaire de préciser une valeur. C'est le cas notamment des attributs `checked`, `compact`, `disabled`, `readonly` et `selected`. En XML, un attribut a forcément une valeur. XHTML définit pour ces attributs une seule valeur autorisée, qui est leur propre nom. On écrira donc par exemple le code suivant : `<option value="carrier" selected="selected">Express</option>`.
- **L'attribut `name` n'est plus autorisé pour les éléments suivants :** `a`, `applet`, `form`, `frame`, `iframe`, `img` et `map`. En XML, le moyen standard d'identification d'un élément, quel qu'il soit, est l'attribut `id`. L'attribut `name` reste valide sur les autres éléments, notamment les champs de formulaires.
- Les valeurs par défaut des attributs sont fournies en minuscules uniquement (par exemple, `get` pour l'attribut `method` de `form`), toujours en raison de la sensibilité de XML à la casse.
- Un dernier détail : si vous utilisez des entités numériques avec un code hexadécimal, en XHTML le `x` initial est forcément en minuscules : `&#x2026;` et non `&#X2026;`.

## La DTD et le prologue

On prendra également soin de toujours référencer la DTD en début de document, avant l'élément ouvrant `html`. Un contenu de qualité utilise soit la variante `Strict`, soit la variante `Frameset` si vous avez des cadres (frames). Toutefois, les cadres sont généralement considérés comme problématiques en termes d'accessibilité et de scripting. Ne les utilisez que si vous ne pouvez absolument pas faire autrement. Voici la déclaration `DOCTYPE` qui devrait figurer en haut de tous vos documents :

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

Cette déclaration n'a rien de superflu : sur la plupart des navigateurs, référencer une DTD stricte déclenche ce qu'on appelle le *doctype switching*, en forçant le navigateur à se placer en mode « respect des standards », ce qui change beaucoup de choses, notamment pour MSIE. D'ailleurs, toutes les améliorations CSS de MSIE 7 ne fonctionneront que dans ce mode, exigeant donc une déclaration de DTD stricte à l'ouverture du document.

Petit point de détail : un document XML (donc XHTML) utilisant un encodage autre que UTF-8 est censé le signaler d'entrée de jeu dans son prologue. Il s'agit d'une syntaxe spéciale présente dès le premier octet du document, et qui ressemble à ceci :

```
<?xml version="1.0" encoding="iso-8859-15"?>
```

Toutefois, MSIE 6 ne fonctionne pas correctement lorsqu'il reçoit un tel document servi comme du (X)HTML (ce point est corrigé dans MSIE 7). Aussi, on s'abstient pour l'instant de fournir un prologue. Heureusement, tous les navigateurs gèrent correctement la balise `meta`, qui fournit la même information, à condition que vous pensiez à la faire systématiquement figurer dans vos `head` :

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
15" />
```

La valeur initiale dépend d'ailleurs de la façon dont vous voulez « servir » votre contenu. Ici, c'est du HTML. On peut aussi opter pour le mode « XML », qui opère quelques petits changements à l'usage, ce qui affecte notamment le comportement du DOM. Le type MIME à utiliser alors est `application/xhtml+xml`. Choisir entre les deux et gérer les conséquences fait l'objet d'un débat incessant depuis de longues années...

## XHTML, oui mais lequel ?

La version actuelle est XHTML 1.1, qui est toutefois encore loin d'être utilisée partout. Cette version n'a plus les variantes `Transitional` et `Loose` de la version 1.0 : elle considère qu'on écrit désormais du XHTML forcément strict (même si on conçoit des cadres) !

La version 1.1 date déjà : elle fut publiée le 31 mai 2001, soit à peine 16 mois après la publication initiale de la version 1.0. Fait plus marquant encore : la dernière révision de XHTML 1.0 date du 1er août 2002, elle est donc *plus récente* que XHTML 1.1...

Les différences avec XHTML 1.0, qui est de loin la variante la plus utilisée ces dernières années, sont très minimes, même pour une sous-version : l'attribut universel `lang` a été remplacé par `xml:lang` à des fins de compatibilité avec la modularisation en cours de XHTML, et une série d'éléments dits « ruby » a été ajoutée, qui permet de fournir des petits blocs de texte dans la marge, annotant le corps de texte principal.

Le gros du travail sur XHTML se concentre aujourd'hui sur XHTML 2.0, mais il s'agit là d'un des nombreux symptômes d'un malaise dans le W3C. En effet, l'immense majorité des ténors du Web ont vivement critiqué cette spécification, et estiment, pour résumer, que leurs auteurs se fourvoient.

Indiquons simplement qu'elle est énorme, divisée en pas moins de 26 modules (la tendance au W3C étant à la modularisation ces dernières années), certains étant minuscules (attributs noyau), d'autres énormes (XForms).

Les développeurs web et leurs spécialistes ont le sentiment que le comité n'a jamais eu à rédiger véritablement un site web, et travaille dans sa tour d'ivoire pour produire une recommandation que personne n'utilisera jamais...

## Le balisage sémantique

C'est un concept très simple : il s'agit d'utiliser une balise pour son sens et non pas pour son aspect par défaut.

Encore aujourd'hui, vous trouverez nombre de développeurs web qui vous diront que « `h1`, c'est pour écrire en gros », ou que « `blockquote` sert à décaler vers la droite ». Ce genre de perception remonte aux premiers navigateurs, vers 1994. Autant dire, à l'échelle du Web, que ça remonte au Jurassique.

L'aspect par défaut d'une balise est sans aucune importance. Je dis bien : aucune. D'ailleurs, il n'existe aucun standard pour ces aspects par défaut (juste une suggestion en annexe D de la recommandation CSS 2.1). Vous ne trouverez pas un seul document formel disant à quoi doit ressembler une balise de titre, ni même `strong` ou `em`.

Bien sûr, on a longtemps eu des balises dédiées à l'aspect, par exemple `font`, `b`, `i`, `u` et `s`. Depuis HTML 4.01 (depuis 7 ans), seules les balises `b` et `i` sont encore autorisées, en raison de certains usages fréquents qui restent plus simples, et plus propres finalement, que de coller un `<span class="bold">…</span>` à la place.

Il est temps d'apprendre le véritable sens de ces balises. Ce faisant, vous découvrirez certaines balises et attributs dont vous n'aviez jamais entendu parler. Par exemple, connaissez-vous les balises `abbr`, `acronym`, `caption`, `cite`, `del`, `fieldset`, `ins`, `kbd`, `label`, `legend`, `q`, `samp`, `tbody`, `tfoot`, `th`, `thead` et `var` ? Je suis sûr qu'au moins certaines ne vous disent absolument rien. Et que dire des attributs `accesskey`, `axis`, `datetime`, `dir`, `for`, `lang`, `longdesc`, `scope` et `summary` ? Tout cela est pourtant lié à la sémantique et à l'accessibilité.

Ce n'est vraiment pas difficile de changer votre façon d'écrire du HTML. Commencez par aller découvrir les balises et attachez-vous à bien comprendre leur sens et leurs contextes autorisés d'utilisation. Lisez bien toute la description de la balise et de ses attributs. Fuyez toute balise et tout attribut dépréciés. La recommandation W3C est là pour ça, elle est claire, et HTML 4.01 dispose même d'une version française de bonne qualité (voir en fin de cette annexe).

# Les balises XHTML 1 Strict par catégorie

Cette section liste l'ensemble des balises autorisées en HTML 4.01 et donc en XHTML 1 (1.0/1.1, à l'exception du module Ruby de XHTML 1.1, trop peu implémenté, et des variantes `Transitional` et `Loose` de 1.0). Les balises dépréciées sont rapidement listées, dans une catégorie à part.

Dans les descriptions, vous trouverez une explication courte du sens et du contexte d'utilisation éventuel pour la balise, ainsi que quelques remarques, mises en garde, et surtout invitations à aller explorer certains attributs trop souvent ignorés ou mal employés.

Il ne s'agit pas de vous apprendre le HTML, pas plus que l'annexe B n'a pour ambition de vous apprendre CSS. Il y a d'excellents livres et sites pour cela ; vous en trouverez quelques-uns dans les sections « Pour aller plus loin... » de ces annexes. Il s'agit plutôt ici de vous rappeler les fondamentaux, et de vous ouvrir les yeux sur le reste.

## Balises structurelles

Les balises structurelles sont celles qui décrivent la structure du document : en-têtes, corps, regroupement d'éléments en ligne ou de type bloc, tableaux...

**Tableau A–1** Balises structurelles de XHTML 1

| Balise | Description |
|---|---|
| `body` | Corps du document. Ne peut contenir que des éléments de type bloc, `script`, `ins` ou `del`. Les attributs `background`, `text`, `link`, `vlink` et `alink` ont été dépréciés en faveur des CSS. |
| `col` | Décrit le format d'une colonne de tableau. Présente dans `table` ou `colgroup`. |
| `colgroup` | Alternative de description des formats d'un groupe de colonnes. Présente dans `table`. |
| `div` | Conteneur de type bloc utilisé pour grouper d'autres éléments à des fins de CSS ou de script. Ne sombrez pas inutilement toutefois dans la *divitis* (http://fr.wikipedia.org/Divitis) ! |
| `head` | En-têtes du document : titre, métadonnées, liens vers des feuilles de styles, scripts et documents connexes. |
| `html` | L'élément racine de tout document (X)HTML. Pour être impeccable, on renseigne ses attributs `xmlns` et `xml:lang`. comme indiqué dans la section 3.1.1 de la recommandation pour XHTML 1.0. |
| `script` | Permet de charger un script séparé (usage correct, dans le cadre de `head`) ou de définir un script interne. Ce dernier usage lui vaut sa place ici, mais il va à l'encontre des principes de l'*unobtrusive JavaScript*, décrits au chapitre 2. Voir également le tableau A-3. |
| `span` | Conteneur servant à regrouper des éléments de type *inline* à des fins de CSS ou de script. |
| `table` | Tableau de données. Ne jamais utiliser pour obtenir une mise en page ! Connaissez-vous son attribut `summary` ? |
| `tbody` | Conteneur de tout ou partie des lignes du corps d'un tableau, présent dans `table` après d'éventuels `thead` et `tfoot`. Oubliez les `tr` directement dans `table`, ce n'est pas soigné ! Lorsqu'un tableau contient plusieurs sous-sections logiques, il est bon de consacrer un `tbody` à chacune. |
| `td` | Cellule de tableau contenant une donnée (*Table Data*). Les attributs `abbr`, `headers` et `scope` font des merveilles pour l'accessibilité à l'intention des logiciels de lecture d'écran ou des personnes souffrant d'un handicap cognitif. |
| `tfoot` | Pied de tableau. Présente dans `table` après un éventuel `thead`, mais avant `tbody`. Censée être répétée en bas de chaque portion visible du tableau quand celui-ci est imprimé sur plusieurs pages, par exemple. |
| `th` | Cellule de tableau contenant un titre de ligne ou de colonne (*Table Heading*). Même remarque que pour `td`. |
| `thead` | En-tête de tableau. Présente dans `table` comme premier élément fils (exception faite de `caption`). Censée être répétée en haut de chaque portion visible du tableau quand celui-ci est imprimé sur plusieurs pages, par exemple. |
| `tr` | Ligne de cellules dans une portion de tableau (*Table Row*). Peut contenir des éléments `th` ou `td`. |

## Balises sémantiques

Les balises sémantiques forment la plus grande catégorie, pour la simple raison que HTML est conçu pour indiquer au mieux la signification des portions de texte qui forment son contenu. Si ce tableau ne vous fait pas découvrir un seul élément, chapeau bas !

**Tableau A–2** Balises sémantiques de XHTML 1

| Balise | Description |
|---|---|
| `abbr` | Indique une abréviation. Si on cherche l'exactitude, une abréviation est soit la version raccourcie d'un mot (par exemple « Mass. » pour « Massachusets »), soit une série d'initiales n'étant pas conçue pour être prononcée, mais plutôt épelée (WWW, URI, HTTP, SNCF…). |
| `acronym` | Indique un acronyme, c'est-à-dire une série d'initiales conçue pour être prononcée (Wysiwyg, Radar, Adullact…). |
| `blockquote` | Bloc de citation, approprié pour une citation longue. L'attribut `cite` permet de fournir l'URI de la source. Voir aussi `q`. |
| `caption` | Titre d'un tableau, préférable de loin à une première `tr` dans `thead` avec un `th` doté d'un éventuel attribut `colspan`. Élément fils immédiat de `table`. Généralement affiché par défaut au-dessus du tableau (réglable par CSS), centré sur la largeur de celui-ci. |
| `cite` | Référence à une source externe (citation au sens de « citer ses sources »). Voir aussi `blockquote` et `q`. |
| `code` | Fragment de code informatique (dans un langage quelconque). Voir aussi `kbd`, `samp` et `var`. |
| `dd` | Corps d'une définition associée au dernier terme (`dt`) la précédant dans une liste de définition (`dl`). Un même terme peut avoir plusieurs définitions successives, ou plusieurs fragments de définition, sous forme de plusieurs éléments `dd`. |
| `del` | Indique une excision (retrait) de contenu dans le cadre d'une révision du document. Un des éléments les plus sous-utilisés de HTML. L'attribut `datetime` permet de dater la modification, et l'attribut `cite` peut référencer l'URI d'un document détaillant ses motivations. L'affichage par défaut est explicite, en laissant son contenu visible mais barré. Voir aussi `ins`. |
| `dfn` | La balise la plus mal spécifiée de tout HTML. Censée encadrer la première occurrence d'un terme, à valeur de définition. Mais la recommandation n'explique pas comment isoler le mot dans sa définition… |
| `dl` | Liste de définitions. Contient une série de termes (`dt`) suivis d'une ou plusieurs définitions (`dd`). La sémantique autorisée est un peu plus large que les définitions à proprement parler, de sorte qu'on peut s'en servir pour un glossaire, un menu, voire même des dialogues dans un scénario. |
| `dt` | Terme à définir, dans le cadre d'une liste `dl`. Suivie d'une ou plusieurs définitions `dd`. |
| `em` | Emphase simple, plus légère que `strong`. Dans un texte, correspond souvent à l'effet produit par la mise en italique. |
| `h1…h6` | Titres à niveaux hiérarchiques, `h1` étant le premier niveau (le plus haut), `h6` le dernier (on en a rarement besoin, ou alors votre page est un immense pavé). Les moteurs de recherche leur accordent en général un « poids » proportionnel. |
| `img` | Insère une image. Devrait toujours avoir un attribut `alt`, vide uniquement si l'image est purement décorative (pour ne pas gêner la représentation textuelle). Les attributs `longdesc` et `title` ont aussi un rôle à jouer dans l'accessibilité. |
| `ins` | Symétrique à `del` : indique un ajout de contenu dans le cadre d'une révision du document. Un des éléments les plus sous-utilisés de HTML. L'attribut `datetime` permet de dater la modification, et l'attribut `cite` peut référencer l'URI d'un document détaillant ses motivations. |
| `kbd` | Indique un texte saisi par l'utilisateur dans le cadre d'une manipulation. Voir aussi `samp`. |

**Tableau A–2** Balises sémantiques de XHTML 1 (suite)

| Balise | Description |
|---|---|
| `label` | Indique un libellé de champ dans un formulaire. Figure aussi au tableau A-6 pour l'aspect formulaire. Ses attributs `accesskey` et `for` sont primordiaux, car certains types de champs ne peuvent définir leur propre `accesskey`. Attention : `for` référence un `id`, pas un `name` ! Peut se présenter sous deux formes, mais on préfère qu'il n'encadre que le libellé lui-même, et pas le champ : cela ouvre davantage de possibilités tant pour les CSS que pour les scripts. |
| `li` | Élément de liste (`ul` ou `ol`). Pensez à bien les fermer en XHTML ! |
| `ol` | Liste ordonnée (séquence croissante de numéros ou lettres). Tous les attributs de séquencement (`start`, `value`) et de format (`type`, `compact`) ont été dépréciés au profit des CSS, plus puissantes d'ailleurs. |
| `p` | Définit un paragraphe, ce qui est en soi un type sémantique de contenu. Pensez à bien les fermer en XHTML ! |
| `q` | Citation courte. L'attribut `cite` permet de référencer l'URI de la source. Voir aussi `blockquote`. |
| `samp` | Exemple d'un affichage ou d'une sortie imprimée par un programme (affichage en console, etc.). Voir aussi `kbd`. |
| `strong` | Emphase forte, plus appuyée que `em`. Dans un texte, correspond à l'effet produit par la mise en gras. |
| `sub` | Texte en indice, généralement dans une formule (*subscript*). |
| `sup` | Texte en exposant, généralement dans une formule ou pour une note de bas de page (*superscript*). |
| `title` | Titre de la page, présente dans `head`. Contient souvent au moins l'équivalent de l'éventuel élément `h1` présent dans le corps de la page, mais avec plus de contexte. Son contenu constitue souvent le titre de la fenêtre ou de l'onglet du navigateur. |
| `ul` | Liste non ordonnée (à puces). Les attributs `type` et `compact` ont été dépréciés au profit des CSS, plus puissantes d'ailleurs. Idéal pour représenter sémantiquement un menu, les CSS pouvant le transformer à volonté… |
| `var` | Indique une variable ou un argument de programme. Voir aussi `code` et `kbd`. |

## Balises de liaison

Il s'agit de balises établissant des liens vers d'autres ressources ou documents.

**Tableau A–3** Balises de liaison de XHTML 1

| Balise | Description |
|---|---|
| `a` | La balise de lien par excellence. Attention : son attribut `target` est déprécié depuis déjà 7 ans, et l'attribut `name` n'est plus autorisé en XHTML : utilisez plutôt un `id` sur l'élément vers lequel vous souhaitez définir une URL de fragment. Côté sémantique, l'utilisation de l'attribut `hreflang` est en pleine explosion (notamment grâce aux sélecteurs CSS 2), l'attribut `rel` commence à faire surface grâce à des fonctionnalités proposées par Google, et l'attribut `type` peut être utile. |
| `base` | Modifie l'URL de base pour les URL relatives au sein du document. Présente dans `head`. Son attribut `target` permet, le cas échéant, de faire que tous les liens s'ouvrent dans un autre cadre que le cadre courant. |

**Tableau A–3** Balises de liaison de XHTML 1 (suite)

| Balise | Description |
|---|---|
| `link` | Lien vers une ressource externe. Principal moyen de chargement de CSS. Ses attributs `type` et `href` sont incontournables, mais l'attribut `rel` est aussi très utile, tant pour proposer plusieurs feuilles de styles alternatives que pour définir des hiérarchies ou réseaux de pages. Connaissez-vous ses valeurs `start`, `next`, `prev`, `contents`, `chapter` ou encore `section` ? Il y en a bien d'autres : http://www.w3.org/TR/html401/types.html#type-links. Enfin, l'attribut `hreflang` existe ici aussi, ce qui est sémantiquement intéressant. |
| `script` | Chargement de scripts externes depuis le `head`. Seule utilisation correcte, contrairement aux fragments de scripts intégrés au contenu de la page (à l'exception des retours Ajax, comme cela est montré dans plusieurs chapitres). Attention : l'attribut `language` est déprécié depuis 7 ans, au profit de l'attribut `type`, qui vaut donc le plus souvent `text/javascript`. |

## Balises de métadonnées

**Tableau A–4** Balises de métadonnées de XHTML 1

| Balise | Description |
|---|---|
| `address` | Encadre les informations de contact des responsables de la page (liens `mailto:`, paragraphes avec une adresse postale,etc.). Extrêmement peu utilisé. À tort ? |
| `bdo` | Court-circuite l'algorithme déterminant le sens du texte (gauche à droite ou droite à gauche) sur la base des attributs `dir` des éléments contenant celui-ci (*Bi-Directional Override*). Son attribut `dir` redéfinit la direction par défaut de son contenu textuel, tandis que son attribut `lang` (ou `xml:lang` en XHTML 1.1) permet d'en changer aussi la langue indiquée. J'avoue ne pas percevoir l'intérêt par rapport à `span`, qui n'a pas plus de valeur sémantique et peut faire la même chose… |
| `meta` | L'élément de métadonnées par excellence. Figure dans `head`, autant de fois que nécessaire. Peut aussi bien remplacer ou compléter les en-têtes de réponse HTTP (attribut `http-equiv`, très utilisé notamment pour préciser le jeu de caractères grâce à `Content-Type`) que rajouter des métadonnées quelconques (attribut `name`). La valeur est dans l'attribut `content`. On l'utilise notamment pour remplir les pages de métadonnées appartenant à des ontologies (vocabulaires) reconnues, comme le Dublin Core (http://dublincore.org/documents/dcmi-terms/). Connaissez-vous son attribut `scheme`, plutôt avancé ? |

## Balises de présentation

**Tableau A–5** Balises de présentation de XHTML 1

| Balise | Description |
|---|---|
| `area` | Définit une zone cliquable sur une image à l'aide de ses attributs `shape`, `coords` et `href`. Ne négligez surtout pas ses attributs `alt`, `tabindex` et `accesskey` pour l'accessibilité ! Voir aussi `map`. |
| `b` | Mise en gras. Jugé plus soigné qu'un `<span class="bold">…</span>` en l'absence de connotation sémantique. |

**Tableau A–5** Balises de présentation de XHTML 1 (suite)

| Balise | Description |
|---|---|
| `big` | Utilisation de la taille de texte supérieure. J'avoue ne pas en voir l'utilité par rapport aux CSS. Voir aussi `small`. |
| `br` | Fin de ligne, ou retour chariot si vous préférez (*line break*). Force le passage à la ligne dans un texte. |
| `frame` | Définition d'un cadre dans un ensemble de cadres. Je rappelle que les cadres sont à éviter le plus possible, car ils nuisent à l'accessibilité et à la navigation. Si vous en utilisez néanmoins, assurez-vous de bien définir l'attribut `longdesc`. |
| `frameset` | Définition d'un ensemble de cadres, qui précise notamment la disposition (horizontale ou verticale). |
| `hr` | Ligne horizontale de séparation (*horizontal rule*). Tous ses attributs spécifiques sont dépréciés au profit des CSS. |
| `i` | Mise en italique. Jugé plus soigné qu'un `<span class="italic">…</span>` en l'absence de connotation sémantique. |
| `map` | Conteneur descriptif pour les zones cliquables (`area`) d'une image (`img` ; je vous déconseille d'utiliser des zones cliquables sur un `input type="image"` ou un `object`, même si c'est autorisé). Son attribut `name` est référencé par l'attribut `usemap` de l'image. Attention à bien renseigner les attributs d'accessibilité des balises `area`. |
| `noframes` | Contenu alternatif pour un navigateur ne prenant pas en charge les cadres. Principalement utile pour les moteurs de recherche, qui n'iront pas dans les cadres. En revanche, tous les navigateurs répandus, même textuels (w3m a supplanté lynx), gèrent aujourd'hui les cadres. |
| `object` | Balise fourre-tout pour objets ne disposant pas d'une balise dédiée : applets Java (la balise `applet` est dépréciée depuis 7 ans), animations Flash ou Shockwave, et sur MSIE n'importe quel contrôle ActiveX (par exemple Windows Update). Essentiellement utilisé pour fournir des animations et des lecteurs audio ou vidéo au sein de la page. |
| `param` | À l'intérieur d'un `object`, les éléments `param` permettent de définir des paramètres nommés pour l'objet. |
| `pre` | Texte préformaté : il s'agit généralement de l'équivalent d'un `<div style="white-space: pre;">`. On y laisse donc normalement les espacements tels quels, ainsi que les retours chariot : les lignes ne sont pas découpées pour satisfaire une contrainte de largeur. Très utilisée pour présenter du code source ou tout autre contenu textuel de nature informatique. La police de caractères par défaut est à chasse fixe (par exemple `Courier New`). |
| `small` | Utilisation de la taille de texte inférieure. J'avoue ne pas en voir l'utilité par rapport aux CSS. Voir aussi `big`. |
| `style` | Permet de fournir un fragment de script dans le corps du document. Cette utilisation est à proscrire, car elle est contraire à la séparation du contenu et de l'aspect. Ne pas confondre avec l'attribut `style` de la plupart des éléments, qui est parfois justifié dans certains contextes très délimités. Voir aussi `link` dans le tableau A-3. |
| `tt` | Utilisation d'une police à chasse fixe (*teletype*). À n'utiliser que si vous estimez que votre contenu n'est pas couvert sémantiquement par `code`, `kbd`, `samp` ou `var`. |

## Balises de formulaires et d'interaction

La plupart des champs de formulaires et boutons disposent d'attributs relatifs à l'ergonomie et l'accessibilité, qu'il vous faut bien connaître. Citons notamment `accesskey` (qu'il est préférable de laisser toutefois sur un `label` associé), `disabled`, `readonly` et `tabindex`.

**Tableau A–6** Balises de formulaire et d'interaction de XHTML 1

| Balise | Description |
|---|---|
| `button` | Variante peu utilisée (car souvent visuellement… bizarre) de `<input type="button"… />`. La balise `button` est en fait un conteneur, dans lequel on peut placer tout type de contenu en ligne, à l'exception des balises de formulaire et de `a`. Elle est donc censée permettre un contenu de bouton plus riche que `input`. Personnellement, je ne m'en sers jamais. |
| `fieldset` | Groupe logique de champs dans un formulaire, doté le plus souvent d'un titre grâce à `legend`. Encore trop peu utilisé et pourtant très utile pour des formulaires un peu voire très riches en contenu. Vous verrez un exemple plus loin dans cette annexe. |
| `form` | Formulaire. Grammaticalement, il n'est pas possible de placer un champ de formulaire hors d'une balise `form`, quitte à ce qu'elle soit inutile par ailleurs. Les principaux attributs sont `method` et `action`. Si vous avez des champs de type fichier, l'attribut `enctype` est indispensable. Quant à `name`, il est déprécié depuis XHTML 1.0, au profit de `id`.<br>Par ailleurs, la DTD exige que `form` ne contienne que des éléments de type bloc ou `script`, hormis `form` bien sûr. Conséquence : vos champs de formulaire doivent apparaître dans des conteneurs de type bloc, souvent `p` ou `fieldset`. |
| `input` | La principale balise de description de champ. Son attribut `type`, quoique doté d'une valeur par défaut (`text`), devrait toujours être précisé, ainsi que `name` (sauf pour le type `submit`, à moins qu'il y en ait plusieurs) et `tabindex`. Pensez aussi, quand c'est pertinent, aux attributs `accept`, `checked`, `disabled`, `maxlength`, `readonly` et `value`. Quant à `size`, préférez utiliser une règle CSS. |
| `label` | Libellé d'un champ de formulaire. Signalé au tableau A-2 pour son rôle sémantique, ce pour quoi je vous encourage bien entendu à fournir un libellé pour tout champ n'en ayant pas déjà un (donc, tous sauf les boutons). Pensez aussi à associer explicitement le `label` à son champ à l'aide d'un `id` sur le champ et du `for` correspondant sur le `label`. `select` ne disposant pas d'attribut `accesskey`, `label` permet de lui en associer un ; d'une manière générale, mettez les `accesskey` sur les libellés quand ils existent. |
| `legend` | Titre d'un groupe logique de champs et libellés dans un formulaire. Généralement le premier élément fils d'un `fieldset`. |
| `optgroup` | Dans un `select`, définit un groupe d'options possibles, ce qui est pratique quand il y en a beaucoup. On utilise l'attribut `label` pour nommer le groupe, et on peut même désactiver juste ce groupe avec `disabled`. On ne peut pas imbriquer les éléments `optgroup` : la hiérarchie n'a qu'un niveau. Suivant le navigateur, l'aspect par défaut varie mais un style peut bien entendu être appliqué. |
| `option` | Option sélectionnable dans un `select`. Peut figurer directement sous le `select` ou dans un `optgroup`. L'attribut `selected` parle de lui-même… Désactivable individuellement avec l'attribut `disabled`. Il est préférable de toujours préciser l'attribut `value`, le comportement normatif (utilisation du contenu textuel comme valeur par défaut) n'étant pas respecté au niveau DOM par MSIE. |

**Tableau A–6** Balises de formulaire et d'interaction de XHTML 1 (suite)

| Balise | Description |
|---|---|
| `script` | Permet l'inclusion de fragments de script directement dans le corps du document, principalement pour manipuler un fragment DOM juste après sa création, sans attendre le chargement complet de la page. C'est une pratique qui va plutôt à l'encontre de l'*unobtrusive JavaScript*, aussi je la déconseille fortement. Là encore, il existe quelques cas, notamment la récupération de fragments XHTML + JS en Ajax, où cette utilisation est toutefois acceptable. |
| `select` | Liste d'options, soit déroulante (attribut `size` de valeur `1`), soit dépliée mais avec un nombre d'options visibles fixe (valeur de l'attribut `size`). Peut contenir des balises `option` et `optgroup`. L'attribut `multiple`, s'il est fourni, indique que l'utilisateur peut sélectionner plusieurs éléments. Attention : il est ergonomiquement incompatible avec le mode déroulant. |
| `textarea` | Zone de saisie de texte multiligne. Utilise une balise fermante. Attention : tout le contenu entre la fin de la balise ouvrante et le début de la fermante fait partie de la valeur, espacements et retours chariot compris ! Les attributs importants pour l'aspect par défaut sont `cols` et `rows`. MSIE y ajoute une pléthore d'extensions, comme pour tous les champs de formulaire, mais la plus fréquemment rencontrée est l'attribut `wrap`. Évitez tous ces ajouts non standards. |

## Balises dépréciées

Voilà déjà 7 ans (HTML 4.01, décembre 1999) que les balises dépréciées n'ont plus droit de cité. Et pourtant, on les rencontre encore souvent... hélas ! Évitez donc de vous attirer les foudres des navigateurs modernes et de vos pairs.

**Tableau A–7** Balises dépréciées de XHTML 1

| Balise | Description |
|---|---|
| `applet` | Permettait l'inclusion d'une applet Java. Remplacée par `object`. |
| `basefont` | Permettait de préciser les valeurs par défaut pour les attributs des balises `font`. |
| `center` | Centrait tout son contenu. Remplacée par les propriétés CSS `text-align` et `margin` (valeur spéciale `auto`). Attention : l'attribut `align` est également déprécié pour tous les éléments ! |
| `dir` | Devait permettre de créer des listes à colonnes multiples, comme un annuaire (*directory*). N'a jamais été vraiment prise en charge. Voir aussi `menu`. |
| `font` | Remplacée par les CSS, bien plus puissantes d'ailleurs. |
| `iframe` | Sorte de cadre en ligne, qui était abondamment utilisé (et, hélas, l'est encore) pour réaliser des échanges en arrière-plan avec le serveur (façon Ajax), ce qui était au moins autant utilisé à des fins légitimes qu'à d'autres bien plus dangereuses. Pas véritablement dépréciée, mais qui ne figurait déjà plus que dans la variante `Loose` de HTML 4.01, la moins soignée. |
| `isindex` | Très vieille variante de champ de saisie textuelle à ligne unique. |
| `menu` | Devait permettre de créer des listes à colonne unique. N'a jamais été vraiment prise en charge. Voir aussi `dir`. |
| `s` | Synonyme de `strike`. |

**Tableau A–7** Balises dépréciées de XHTML 1 (suite)

| Balise | Description |
|---|---|
| `strike` | Barrait un contenu. Remplacée par `del` quand la sémantique convient, par la propriété CSS `text-decoration` sinon. |
| `u` | Soulignait le texte. Remplacée par la propriété CSS `text-decoration`. |

# Attributs incontournables

Pour finir, voici quelques attributs incontournables, dits « noyau », identifiables dans la DTD sous l'entité `%coreattrs`. Tous les éléments en disposent.

**Tableau A–8** Attributs incontournables de XHTML 1

| Attribut | Description |
|---|---|
| `class` | Contient une ou plusieurs classe(s) CSS. Le saviez-vous ? Il est tout à fait possible de fournir plusieurs classes à un même élément, qui cumule alors les applications de règles. Cela est très pratique, et évite d'avoir à dupliquer nombre de règles dans la feuille de styles. Les classes sont séparées dans la valeur de l'attribut par des espaces. |
| `dir` | Indique la direction d'écriture du contenu (`ltr` ou `rtl`, pour gauche à droite ou droite à gauche). Voir aussi `lang`. |
| `id` | De loin l'attribut le plus utilisé, avant même `class`, dans un document soigné et sémantique ! Contient un identifiant, unique à travers tout le document, grâce auquel tant les CSS que les scripts pourront référencer l'élément en un clin d'œil.<br>Le saviez-vous ? Un ID valide doit commencer par une lettre non accentuée et peut ensuite contenir des chiffres arabes et des traits de soulignement (_), mais aussi des tirets (-), des points (`.`) et des deux-points (`:`) ! Ces deux dernières possibilités peuvent toutefois poser un problème dans la syntaxe des règles CSS, qui ont alors besoin d'encadrer les ID par des apostrophes (`'`) pour s'y retrouver. |
| `lang` | Langue du contenu. Contient un code de langue de la RFC 1766, par exemple `fr-FR`. Voir aussi `dir`. |
| `style` | Style en ligne appliqué à l'élément. Généralement mal vu, car enfreint la séparation entre contenu et aspect. Parfois nécessaire cependant, en particulier pour masquer initialement un contenu avec `display: none` dans le cadre des bibliothèques Prototype et donc script.aculo.us (voir chapitres 4, 7 et 8). |
| `title` | Fournit une information descriptive ou complémentaire sur l'élément. L'information est généralement affichée dans une infobulle si le curseur de la souris reste un instant sur l'élément. Fondamental pour des balises comme `abbr` et `acronym`, il est aussi utile pour préciser la destination d'un lien ou nommer les feuilles de styles alternatives d'un document. Il a enfin de nombreux usages dans le cadre de l'accessibilité. |

# Besoins fréquents, solutions concrètes

À présent que nous avons fait le tour des balises disponibles en XHTML 1 Strict, il est temps d'en illustrer l'utilisation la plus correcte possible dans le cadre de certains besoins récurrents.

## Un formulaire complexe mais impeccable

Prenons un formulaire sur un site communautaire, servant à éditer des informations de profil. Les concepteurs du site ont décidé que la totalité des informations étaient présentées dans un même formulaire. Ces informations sont réparties en 5 groupes logiques :

1. connexion (identifiant, mot de passe) ;
2. identité (civilité, nom, prénom, surnom) ;
3. détails personnels (adresse courriel, identifiants de messagerie instantanée, site personnel, petit descriptif en texte libre, photo) ;
4. centres d'intérêt (série de cases à cocher) ;
5. conditions de fonctionnement (CGU, divulgation d'informations à des partenaires, type d'abonnement à la lettre de diffusion du site).

Le formulaire doit être le plus lisible possible, pour tout type de public. On veillera donc à éviter une mise en page tableau, pour lui préférer l'emploi de CSS adaptables au type de navigateur (petit ou grand écran, graphique ou texte, lecteur d'écran ou plage braille, etc.) et aux besoins de l'utilisateur (navigation intelligente à l'aide de la touche Tab et de touches de raccourci, libellés associés aux champs, etc.).

La photo peut être téléchargée à l'aide d'un champ de type fichier. Ce champ doit filtrer les possibilités pour ne retenir que les formats d'image pris en charge par le site, pour lequel on a décidé de se limiter à JPEG, PNG et GIF. Une limite de taille de fichier sera gérée côté serveur, et l'image y sera de toutes façons redimensionnée.

Évidemment, toute validation ou complément fonctionnel JavaScript sera réalisé de façon non intrusive, depuis un script séparé. Aucun gestionnaire d'événement ne sera présent dans le code, et un envoi sans activation JavaScript sera possible (avec un champ de type `submit` ou `image`).

Voici le code HTML répondant à ce besoin (vous le retrouverez dans l'archive de codes source pour ce livre, disponible sur le site des éditions Eyrolles).

**Listing A-1 Notre première section, dédiée aux informations de connexion**

```
<form id="profile" enctype="multipart/form-data"
      ➥ method="post" action="/profile/update">
```

```
<fieldset id="credentials">
    <legend>Connexion</legend>
    <p>
        <label for="edtReqLogin" accesskey="E">Identifiant</label>
        <input type="text" id="edtReqLogin" name="login"
               ➥ value="tdd" tabindex="1" />
    </p>
    <p>
        <label for="edtPassword" accesskey="A">Nouveau mot de passe
        </label>
        <input type="password" id="edtPassword" name="password"
               ➥ tabindex="2" />
    </p>
    <p>
        <label for="edtConfirm" accesskey="C">Confirmation</label>
        <input type="password" id="edtConfirm" name="confirm"
               ➥ tabindex="3" />
    </p>
</fieldset>
```

- Notez l'attribut `enctype` du formulaire `form`, en prévision du champ de téléchargement de fichier du listing A-3.
- Chaque champ fait l'objet d'une valeur à progression logique pour `tabindex`.
- Le mot de passe étant saisi sans confirmation visuelle, on ajoute un champ de confirmation pour détecter une saisie erronée.
- Tous les libellés définissent une `accesskey` pour leur champ.

Voyons à présent la deuxième section du formulaire.

**Listing A-2 Informations sur l'identité de l'utilisateur**

```
<fieldset id="identity">
    <legend>Identité</legend>
    <p>
        <label for="cbxTitle" accesskey="V">Vous êtes</label>
        <span id="basicIdentity">
           <select id="cbxTitle" name="title" size="1" tabindex="4">
                <option value="1" selected="selected">M.</option>
                <option value="2">Mlle</option>
                <option value="3">Mme</option>
            </select>
           <input type="text" id="edtReqFirstName" name="firstName"
                ➥ accesskey="P" value="Christophe" tabindex="5" />
            <input type="text" id="edtReqFirstName" name="lastName"
                ➥ accesskey="N" value="Porteneuve" tabindex="6" />
        </span>
    </p>
```

```
        <p>
            <label for="edtNickname" accesskey="S">Surnom</label>
            <input type="text" id="edtNickname" name="nickname"
                   ➥ value="TDD" tabindex="7" />
        </p>
    </fieldset>
```

- Afin de gagner en place et en ergonomie, nous plaçons les champs composant l'identité globale (civilité, prénom, nom) sur une même ligne. Pressentant un besoin de groupement pour la CSS, on encadre les trois par un `span`.
- Le `select`, en `size` 1, est une liste déroulante. L'élément retenu par défaut est marqué par un attribut bascule `selected`. Notez sa valeur, seule autorisée par la DTD.

La section sur les détails personnels est de loin la plus riche.

**Listing A-3 Détails personnels**

```
<fieldset id="details">
  <legend>Détails personnels</legend>
  <p>
    <label for="edtReqEmail" accesskey="O">Courriel</label>
    <input type="text" id="edtReqEmail" name="email"
           ➥ value="tdd@example.com" tabindex="8" />
  </p>
  <p>
    <label><abbr title="Instant Messaging,
                ➥ Messagerie instantanée">IM</abbr></label>
    <span id="imIDs">
      <label for="edtMSN" accesskey="1">MSN</label>
      <input type="text" id="edtMSN" name="im[msn]"
             ➥ value="tdd@example.com" tabindex="9" title="Alt+1" />
      <label for="edtICQ" accesskey="2">ICQ</label>
      <input type="text" id="edtICQ" name="im[icq]"
             ➥ value="123456789" tabindex="10" title="Alt+2" />
      <label for="edtJabber" accesskey="3">Jabber</label>
      <input type="text" id="edtJabber" name="im[jabber]"
            ➥ value="tdd@example.com" tabindex="11" title="Alt+3" />
    </span>
  </p>
  <p>
    <label for="edtHomePage" accesskey="T">Site personnel</label>
    <input type="text" id="edtHomePage" name="homePage"
           ➥ value="http://www.example.com" tabindex="12" />
  </p>
  <p id="detailsContainer">
    <label for="memDetails" accesskey="D">Détails</label>
    <textarea id="memDetails" name="details" cols="40" rows="3"
              ➥ tabindex="13"></textarea>
  </p>
```

```
    <p>
      <label for="filPhoto" accesskey="H">Photo</label>
      <span id="photo">
        <input type="file" id="filPhoto" name="photo" tabindex="14"
               ➥ accept="image/jpeg,image/gif,image/png" />
        <span class="note">(JPEG, PNG ou GIF, 150ko maximum)</span>
      </span>
    </p>
  </fieldset>
```

- Notez l'explication de l'abréviation IM, à l'aide d'`abbr` et de son attribut `title`. La CSS incitera l'utilisateur à amener sa souris sur le terme et à l'y laisser un instant pour obtenir l'explication.
- Là aussi, quelques éléments sur la même ligne sont regroupés dans un `span` en prévision des CSS.
- Remarquez comme les champs de saisie d'identifiants IM précisent leur touche de raccourci avec l'attribut `title` : le caractère actif ne figurant pas dans leur libellé, il ne pourra pas être mis en évidence par un script. Il faut tenter de fournir l'information autrement, et `title` produit généralement une infobulle.
- Remarquez que `textarea` est un élément non vide : il faut utiliser une balise fermante. Les suggestions de taille proposées par les attributs `cols` et `rows` seront remplacées par celles de la CSS quand celle-ci est active.
- Exemple de champ de type `file`. Il n'est pas possible de spécifier une valeur dans le HTML. Notez l'attribut `accept`.

La section 4, celle des centres d'intérêt, est plus simple. On suppose que la liste de ceux-ci est dynamiquement extraite de la base de données, ce qui empêche l'utilisation de touches de raccourci, car on ne connaît pas forcément tous les libellés affichés.

**Listing A-4 Centres d'intérêt**

```
<fieldset id="interests">
    <legend>Centres d'intérêt</legend>
    <ul>
        <li>
            <input type="checkbox" id="chkInterests_1"
                   ➥ name="interests" value="1" />
            <label for="chkInterests_1">Jeux vidéo</label>
        </li>
        <li>
            <input type="checkbox" id="chkInterests_2"
                   ➥ name="interests" value="2" />
            <label for="chkInterests_2">Mode</label>
        </li>
```

```
            <li>
                <input type="checkbox" id="chkInterests_3"
                       ➥ name="interests" value="3" />
                <label for="chkInterests_3">Gadgets</label>
            </li>
        </ul>
    </fieldset>
```

Ici, une liste non ordonnée est sémantiquement parfaite pour représenter cette liste de choix individuels. La CSS fera disparaître les puces et ajustera les positions. Pour faciliter le traitement côté serveur, on donne à tous les champs le même nom, mais une valeur différente (ici probablement l'ID de l'information dans la base), utilisée pour composer l'`id` HTML et ainsi associer le libellé à la case. Cette association permet à l'utilisateur de basculer l'état de la case en cliquant sur le libellé lui-même, ce qui est plus accessible et similaire aux interfaces classiques. Côté serveur, on recevra le champ une fois par case cochée, avec à chaque fois la bonne valeur.

Enfin, voici la dernière section, le bouton d'envoi et la clôture du formulaire.

**Listing A-5 Conditions de fonctionnement**

```
    <fieldset id="account-behavior">
        <legend>Conditions de fonctionnement</legend>
        <ul>
            <li>
                <input type="checkbox" id="chkCGU" name="cgu"
                       ➥ value="yes"checked="checked" />
                <label for="chkCGU" accesskey="U">J'accepte les
conditions générales d'utilisation.</label>
            </li>
            <li>
                <input type="checkbox" id="chkSpreadData"
                       ➥ name="spreadData" value="yes" />
               <label for="chkSpreadData" accesskey="J">Je consens à ce
que mes informations personnelles soient divulguées à des partenaires.
               </label>
            </li>
        </ul>
        <h2>La <span lang="en">newsletter</span> du site</h2>
        <ul>
            <li>
                <input type="radio" id="rbtNoSub" name="newsletter"
                       ➥ value="no" checked="checked" />
                <label for="rbtNoSub" accesskey="R">Je ne désire pas la
recevoir.</label>
            </li>
```

```
            <li>
                <input type="radio" id="rbtWeekly"
                       ➥ name="newsletter" value="weekly" />
                <label for="rbtWeekly" accesskey="B">Je désire la
recevoir hebdomadairement.</label>
            </li>
            <li>
                <input type="radio" id="rbtMonthly"
                       ➥ name="newsletter" value="monthly" />
                <label for="rbtMonthly" accesskey="L">Je désire la
recevoir mensuellement.</label>
            </li>
        </ul>
    </fieldset>
    <p class="submit"><input type="submit" id="btnSubmit" value="Mettre
à jour mon profil" accesskey="M" title="Alt+M" /></p>
</form>
```

- Pour une case à cocher isolée, on précise la valeur afin d'éviter de dépendre du navigateur pour la valeur par défaut (suivant les cas on obtient `on`, `1`, `yes`...), ce qui compliquerait le test côté serveur. Tester simplement la présence du champ dans la requête ne serait pas très robuste.
- Remarquez l'attribut bascule `checked` et sa seule valeur autorisée.
- Notez la précision de la langue pour le terme « newsletter », qui permettra notamment aux lecteurs d'écran de le prononcer correctement à l'intention des non-voyants ou malvoyants. La CSS pourrait même basculer en italique de tels éléments, par exemple (sélecteurs d'attribut CSS 2, ou de langue CSS 3).
- Les boutons radio mutuellement exclusifs portent le même nom, mais des valeurs différentes. Le champ n'est envoyé qu'une fois, pour le bouton sélectionné.
- Le bouton d'envoi, champ de type `submit`, n'a pas de libellé externe : il précise donc son `accesskey` et la rend consultable avec `title`. Il est par ailleurs inutile de lui donner un attribut `name`, ce qui enverrait un champ au serveur alors qu'il n'y a pas plusieurs modes d'envoi parmi lesquels choisir...

Le fichier HTML complet pèse 5 Ko, et la CSS en ajoute 1,5, soit un total de 6,5 Ko. Croyez-moi sur parole, l'équivalent en HTML des années 1990 est beaucoup plus lourd, alors qu'il est aussi beaucoup plus rigide et incapable de s'adapter d'un mode de consultation à un autre.

La figure A–1 vous montre le résultat sans aucune feuille de styles.


**Figure A–1**
Notre formulaire sans aucune mise en page

Cela fait certes peur, mais ce n'est pas grave : le contenu de la page a du sens, *beaucoup de sens*. La figure A–2 montre le formulaire, avec une feuille de styles et l'application d'un petit script de décoration automatique de libellés, comme celui vu au chapitre 3 dans la section « Décoration automatique de labels » :

Quand je vous disais que les CSS font des miracles...

Il est par ailleurs possible d'y ajouter un script non intrusif de validation automatique de formulaire, comme vu au chapitre 3, dans la section « Validation automatique de formulaires ». Vous trouverez aussi en fin d'annexe, dans la section « Pour aller plus loin... », l'URL d'un article OpenWeb dédié à la conception de formulaires sémantiques.


**Figure A–2**
Le même formulaire, mis en forme par CSS et un script complémentaire

## Un tableau de données à en-têtes groupés

Prenons maintenant l'exemple d'un tableau de données. La balise `table` et ses collègues : `thead`, `tbody`, `tfoot`, `tr`, `th` et `td`, ne sont pas à éviter comme la peste ! Simplement, elles servent à décrire des tableaux de données, pas à réaliser une mise en page aussi rigide (et lourde) qu'un parpaing.

Supposons que nous souhaitions réaliser un tableau représentant en abscisse des produits (classés par catégories) et en ordonnée des modes de livraison, eux-mêmes déclinés par zone géographique de livraison. On aimerait que les en-têtes du tableau persistent au-delà d'un saut de page à l'impression. On aimerait aussi qu'il soient sémantiques et un minimum accessibles aux utilisateurs de lecteurs d'écran.

Voici comment procéder...

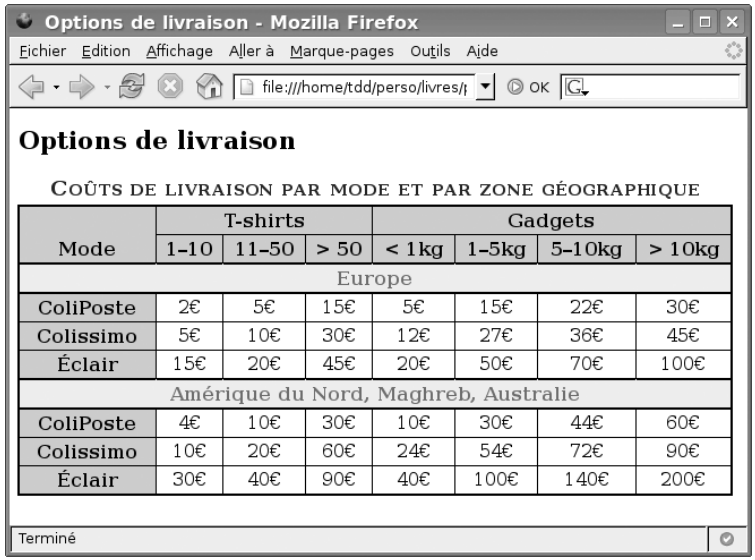
**Listing A-6 Un tableau bien structuré, sémantique et accessible**

```
<table id="shipment" summary="Options de livraisons et leurs coûts">
    <caption>Coûts de livraison par mode et par zone géographique
    </caption>
    <thead>
        <tr>
            <th rowspan="2" class="bodyHeading">Mode</th>
            <th colspan="3">T-shirts</th>
            <th colspan="4">Gadgets</th>
        </tr>
        <tr>
            <th>1&ndash;10</th>
            <th>11&ndash;50</th>
            <th>&gt; 50</th>
            <th>&lt; 1kg</th>
            <th>1&ndash;5kg</th>
            <th>5&ndash;10kg</th>
            <th>&gt; 10kg</th>
        </tr>
    </thead>
    <tbody>
        <tr><th colspan="8" class="zone">Europe</th></tr>
        <tr>
            <th>ColiPoste</th>
            <td>2€</td>
            <td>5€</td>
            <td>15€</td>
            <td>5€</td>
            <td>15€</td>
            <td>22€</td>
            <td>30€</td>
        </tr>
        <tr>
            <th>Colissimo</th>
…
        </tr>
    </tbody>
    <tbody>
        <tr><th colspan="8" class="zone">Amérique du Nord, Maghreb,
Australie</th></tr>
…
    </tbody>
</table>
```

- Notez l'attribut `summary`, qui résume le contenu de la table, et l'élément fils `caption`, qui lui fournit un titre explicite.

- Les en-têtes sont, logiquement, dans `thead`. Les cellules qui y figurent ne fournissent que des titres, ce sont donc des `th`.
- Les différents corps de données (ici un par zone géographique) sont des `tbody`. Chaque corps a ici un sous-titre dédié, réalisé avec une première ligne ne comportant qu'un `th` sur toute la largeur.
- Chaque ligne démarre par un titre, donc un `th`, puis n'a que des données, donc des `td`.

Le fichier HTML pèse 2 Ko, la CSS, quoique bien aérée, ne pèse que 610 octets. Voici l'aspect du tableau, doté de cette feuille de styles minimaliste :




| Mode | T-shirts | | | Gadgets | | | |
|---|---|---|---|---|---|---|---|
| | 1–10 | 11–50 | > 50 | < 1kg | 1–5kg | 5–10kg | > 10kg |
| Europe | | | | | | | |
| ColiPoste | 2€ | 5€ | 15€ | 5€ | 15€ | 22€ | 30€ |
| Colissimo | 5€ | 10€ | 30€ | 12€ | 27€ | 36€ | 45€ |
| Éclair | 15€ | 20€ | 45€ | 20€ | 50€ | 70€ | 100€ |
| Amérique du Nord, Maghreb, Australie | | | | | | | |
| ColiPoste | 4€ | 10€ | 30€ | 10€ | 30€ | 44€ | 60€ |
| Colissimo | 10€ | 20€ | 60€ | 24€ | 54€ | 72€ | 90€ |
| Éclair | 30€ | 40€ | 90€ | 40€ | 100€ | 140€ | 200€ |

**Figure A–3**
Notre tableau avec un style léger

Si on souhaite rendre maximale l'accessibilité, en particulier à destination des lecteurs d'écran (non-voyants, malvoyants) et des personnes souffrant d'un handicap cognitif, les éléments `th` et `td` disposent d'attributs `abbr`, `headers` et `scope` extrêmement utiles. Vous pourrez en apprendre davantage sur cet aspect et les tableaux en général ici : http://pompage.net/pompe/autableau/.

## Un didacticiel technique

Pour finir, penchons-nous sur les balises sémantiques en ligne, c'est-à-dire conçues pour indiquer le sens d'un fragment de texte. Nous allons prendre l'exemple d'une documentation technique qui, après avoir référencé quelques sources et cité un fragment pertinent de l'une d'elles, fournit aux lecteurs un morceau de code source et les guide à travers quelques manipulations au clavier dans leur console.

**Listing A-7 Une documentation au balisage hautement sémantique**

```
<h1>Lister les libellés d'un document</h1>

<p>Nous allons apprendre à lister les libellés d'un document HTML à l'aide
du DOM niveau 2 (noyau et HTML). On se reposera essentiellement sur
<code>document.getElementsByTagName</code>. Cette méthode, je cite,
<q cite="http://www.w3.org/TR/DOM-Level-2-Core/core.html#ID-A6C9094"
lang="en">Returns a <code>NodeList</code> of all the <code>Element</code>s
with a given tag name in the order in which they are encountered in a
preorder traversal of the <code>Document</code> tree.</q>.</p>

<p>Qu'est-ce qu'une <code>NodeList</code> ? C'est une interface
d'énumération de n&oelig;uds. La spécification la décrit ainsi :
</p>

<blockquote cite="http://www.w3.org/TR/DOM-Level-2-Core/core.html#ID-
536297177" lang="en">
    <p>The <code>NodeList</code> interface provides the abstraction of
an ordered collection of nodes, without defining or constraining how
this collection is implemented. <code>NodeList</code> objects in the DOM
are live.</p>
    <p>The items in the <code>NodeList</code> are accessible via an
integral index, starting from 0.</p>
</blockquote>

<p>Voici le code nécessaire. Par simplicité, on suppose qu'une fonction
<code>getInnerText</code> existe :</p>

<pre class="code">
<b>function</b> printLabelsInfo() {
    <b>var</b> msg = '';
    <b>var</b> labels = document.getElementsByTagName('label');
    <b>for</b> (<b>var</b> index = 0; index &lt; labels.length; ++index) {
        <b>var</b> label = labels[index];
        msg += getInnerText(label);
        <b>if</b> (label.hasAttribute('for') &amp;&amp;
                document.getElementById(label.htmlFor))
            msg += ' -&gt;' + label.htmlFor;
        <b>if</b> (label.hasAttribute('accesskey'))
            msg += ' (Alt+' + label.accessKey)';
        msg += '\n';
    }
    alert(msg);
} <i>// printLabelsInfo</i>
</pre>

<p>Sauvegardez ce code dans un fichier <tt>labels.js</tt>. Vous
remarquez qu'il est de faible taille :</p>
<pre class="console">
```

```
<samp>$</samp> <kbd>ls -l</kbd>
<samp>total 4
-rw-r--r-- 1 demo demo 530 2006-08-23 16:58 labels.js
$</samp>
</pre>
```

- On voit ici des exemples de nombreuses balises sémantiques. On le voit, `code` représente du code de façon générale (lorsqu'il s'agit d'un nom de variable dans un code source, on utilise de préférence `var`), et `tt` sert à indiquer les textes perçus comme du contenu informatique mais qui ne sont pas forcément affichés par le système (`samp`), ni saisis (`kbd`). Classiquement, les noms de fichiers.
- Dans un code source, on peut vouloir améliorer la mise en page, par exemple en mettant les mots réservés en gras et les commentaires en italique, et peut-être en gris. Ce n'est que de la coloration syntaxique, sans sémantique : `i` et `b` sont parfaits pour cela !
- Remarquez les citations brèves, en ligne, avec `q` (et son attribut `cite` pour avoir la source), et celles plus longues, en bloc, avec `blockquote` (et son attribut `cite`, là encore).

Là aussi, le HTML et la CSS ne totalisent que 3 Ko. Voici le résultat, doté d'une légère feuille de styles.

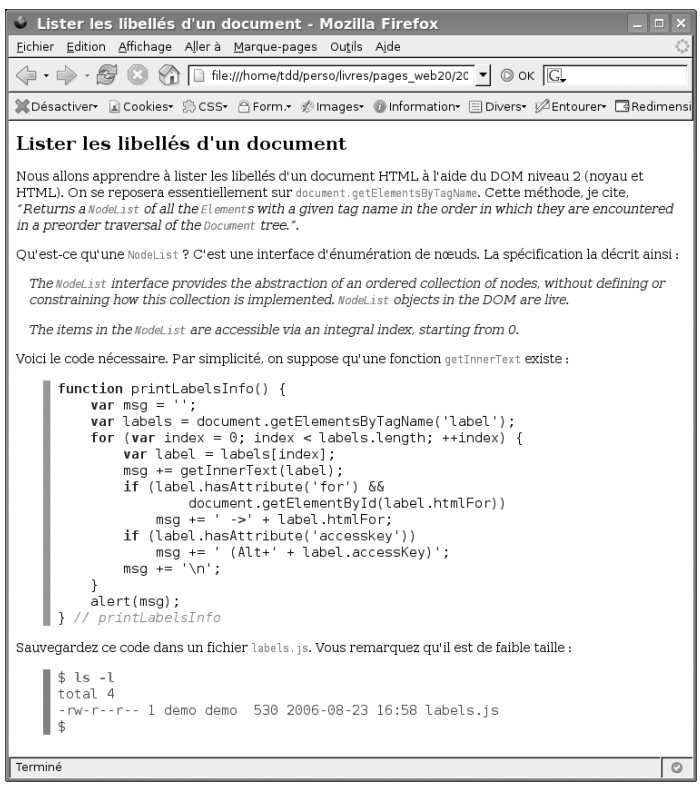



**Figure A–4**
Notre fragment de documentation avec un style léger

# Pour aller plus loin...

## Livres

*Réussir son site web avec XHTML et CSS, 2e édition*
Mathieu Nebra
Editions Eyrolles, mars 2008, 316 pages
ISBN 978-2-212-12307-4

*HTML avec CSS et XHTML Tête la première*
Elisabeth Freeman, Eric Freeman
O'Reilly France, août 2006, 720 pages (de bonheur)
ISBN 2-841-77413-9

*Introduction à HTML et CSS*
Eric Sarrion
O'Reilly, février 2006, 231 pages
ISBN 2-841-77400-7

*HTML et CSS 2*
Molly Holzschlag
Pearson Education, septembre 2005, 330 pages
ISBN 2-744-01994-1

*Design web : utiliser les standards*
Jeffrey Zeldman
Eyrolles, avril 2005, 414 pages (d'évangile)
ISBN 2-212-11548-2

*HTML : précis et concis*
Jennifer Niederst
O'Reilly, avril 2002, 122 pages (d'aide-mémoire)
ISBN 2-841-77157-1

## Sites

- Le W3C évidemment : http://w3.org. Et les recommandations fondamentales (seule la version anglaise a valeur de référence) :
  - HTML 4.01 : http://w3.org/TR/html401/ (version française à cette adresse : http://www.la-grange.net/w3c/html4.01/cover.html)
  - XHTML 1.0 : http://w3.org/TR/xhtml1/ (version française à cette adresse : http://www.la-grange.net/w3c/xhtml1/)

- OpenWeb, un excellent site didactique en français, joli et pratique : http://openweb.eu.org/. On notera en particulier :
  - http://openweb.eu.org/articles/xhtml_une_heure/
  - http://openweb.eu.org/articles/respecter_semantique/
  - http://openweb.eu.org/articles/html_au_xhtml/
  - http://openweb.eu.org/articles/formulaire_accessible/
- Pompage, excellent site voué à la traduction francophone d'articles de grande qualité sur la conception web : http://pompage.net. On consultera tout particulièrement, histoire de se mettre en appétit, les articles suivants :
  - http://pompage.net/pompe/bonpiedstandards/
  - http://pompage.net/pompe/cederholm/
  - http://pompage.net/pompe/separation/
  - http://pompage.net/pompe/autableau/
  - http://pompage.net/pompe/listes/
  - http://pompage.net/pompe/listesdefinitions/
  - http://pompage.net/pompe/doctypecontenttype/
- AccessiWeb, la cellule accessibilité de BrailleNet, regorge de documentations et manuels pratiques en français pour rendre vos sites et créations numériques plus accessibles : http://www.accessiweb.org/.
- Opquast est un référentiel français de bonnes pratiques de conception web classées suivant de multiples axes, doté en plus d'un outil d'aide à l'estimation et au suivi de la qualité de vos sites : http://www.opquast.com/.
- Alsa Créations est un site didactique plein d'excellents articles sur les standards, XHTML et CSS : http://www.alsacreations.com/. Un forum permet par ailleurs aux visiteurs de s'entraider.
- A List Apart est un incontournable : http://alistapart.com/.

# B

# Aspect irréprochable et flexible : CSS 2.1

Cette annexe n'a pas l'intention de vous apprendre CSS. Le sujet est si vaste, et les subtilités si nombreuses, qu'il faudrait pour cela des livres entiers (et il en existe d'excellents, voir en fin d'annexe). Non, il s'agit juste de faire le point sur la prise en charge des CSS dans les navigateurs, de rappeler quelques grandes notions fondamentales (structure des règles, cascade, modèles fondateurs), et de faire un tour d'horizon des sélecteurs et des propriétés.

Notez que ce survol sera moins détaillé que pour l'annexe A, dont l'objectif était d'attirer votre attention sur les bons et les mauvais usages fréquents et de mettre en avant des attributs sous-employés, ce qui exigeait de longues descriptions et quelques exemples solides. Ici, on a plutôt une sorte de référence laconique.

# Statut, état et vocabulaire

CSS est un standard W3C, publié sous forme de recommandations. À l'exception de (X)HTML, il n'y a sans doute pas de standard du Web plus utilisé que CSS.

## Les versions de CSS

**Tableau B–1** Les versions du standard CSS

| Version | Description |
|---|---|
| 1.0 | Première édition le 17 décembre 1996 (eh oui, les CSS ont dix ans !), avec une révision le 11 janvier 1999. Bien qu'assez simple, cette première mouture fournissait déjà l'essentiel des modèles de boîte et de mise en forme visuelle. |
| 2.0 | Sortie le 12 mai 1998, cette version ajoutait énormément de choses : les types de média, la valeur spéciale `inherit`, la pagination, des fonctionnalités d'internationalisation, une gestion détaillée des tableaux, les positionnements absolu, relatif et fixe, la gestion du débordement et de la troncature de contenu, le contenu généré, et j'en passe. |
| 2.1 | Toujours à l'état d'ébauche (cinquième révision le 11 avril 2006), cette version constitue surtout un affinage et une série de corrections et de précisions à la version 2.0. Elle constitue l'état « actuel » de CSS dans la plupart des navigateurs. |
| 3.0 | Dans la tendance actuelle de modularisation au W3C, CSS 3 est constituée de 37 (oui, vous avez bien lu) modules, chacun faisant l'objet d'un travail indépendant. À l'heure où j'écris ces lignes, aucun n'est totalement finalisé !<br>Huit le sont presque (CR, *Candidate Recommandation*), trois sont au stade précédent (*last call*), dont celui des sélecteurs (extrêmement attendu !) et celui des polices de caractères, et le reste est en ébauche (parfois figée depuis des années), voire même pas démarré (cinq modules).<br>En somme, le travail n'avance pas. |

## Prise en charge actuelle

Comme on peut le voir dans l'avant-propos, la prise en charge de CSS est généralement bonne, avec toutefois quelques écarts. Pour simplifier, on peut dire qu'avec la sortie de MSIE 7, tous les navigateurs véritablement répandus prenaient décemment en charge CSS 2.1.

Voici tout de même un petit bilan (mais n'hésitez pas à consulter les informations techniques de vos navigateurs cibles : ces données évoluent souvent). Attention : les informations pour MSIE supposent une page en mode strict (utilisation d'un `DOCTYPE` sur une DTD (X)HTML `Strict`). Voir l'annexe A à ce sujet.

Les adjectifs expriment le degré de prise en charge.

| Navigateur | CSS 1 | CSS 2.1 | CSS 3 |
|---|---|---|---|
| MSIE 6 | Bon | Moyen | - |
| MSIE 7 | Très bon | Bon | - |
| MSIE 8 b1 | Très bon | Très bon | Léger |
| Mozilla, Firefox, Camino | Très bon | Très bon | Partiel |
| Safari | Très bon | Très bon | Léger |
| Opera | Très bon | Très bon | Léger |

Les parties les moins bien prises en charge de CSS 2.1 concernent surtout les feuilles de styles auditives (*aural style sheets*), qui visent à contrôler la lecture vocale des pages : sexe de la voix, richesse, prosodie, intonation, débit... Ce ne sont pas vraiment les navigateurs qui doivent implémenter cela, mais les lecteurs d'écran !

## Le jargon CSS

Lorsqu'on parle de CSS, certains termes précis reviennent régulièrement. Reprenons-les ici avant de poursuivre la lecture de cette annexe.

- Une **propriété** est un nom spécifique représentant un aspect CSS. Par exemple, `font-weight` gère l'épaisseur des caractères (normale, grasse, etc.) tandis que `white-space` régit le traitement des espacements et retours chariot. Chaque propriété dispose d'une série de valeurs possibles, avec généralement plusieurs syntaxes autorisées. Elle peut avoir une valeur par défaut, bénéficier ou non de l'héritage par cascade, et ne s'appliquer qu'à certains éléments.
- Un **sélecteur** est un texte respectant une syntaxe spéciale qui permet de désigner des éléments de la page en fonction de critères variés : leur nom de balise, leur ID, leurs classes CSS, la présence d'un attribut, leur position dans le document, la langue de leur contenu, etc. En combinant des sélecteurs, on peut décrire une extraction très fine et très précise des éléments dans le document.
- Une **règle** est l'élément de base d'une feuille de styles CSS : elle définit une série de propriétés pour les éléments désignés par un ou plusieurs sélecteurs.

Prêt pour un exemple complexe ? C'est parti ! Dans le code CSS suivant :

```
#examples tr.critical>th:first-child, th.critical {
    background: red;
    color: white;
    font-weight: bold;
}
```

- `#examples` est un sélecteur d'ID, l'espace qui suit est un sélecteur de descendant, `tr` est un sélecteur de type, `.critical` est un sélecteur de classe, le chevron fermant (>) est un sélecteur d'élément fils, `th` est un sélecteur de type, `:first-child` est une pseudo-classe et la virgule est un opérateur de groupement entre deux combinaisons de sélecteurs. Si vous êtes perdu, ne vous en faites pas, on reverra ces sélecteurs plus loin.
- `background`, `color` et `font-weight` sont des propriétés, dont `red`, `white` et `bold` sont les valeurs respectives.
- L'ensemble du fragment constitue une règle.

Histoire de ne pas vous faire languir si vous n'êtes pas très au point côté CSS, la règle s'applique ici aux éléments suivants :

- Les éléments `th` apparaissant comme premier élément fils dans les `tr` ayant une classe CSS `critical`, lesquels apparaissent quelque part dans un élément d'ID `examples`.
- Les éléments `th` ayant une classe CSS `critical`.

# Bien comprendre la « cascade » et l'héritage

Les feuilles de styles applicables à un document peuvent avoir trois origines :

- **Le navigateur**, qui fournit généralement une feuille de styles par défaut (grâce à laquelle les titres sont plus gros, en italique, etc.).
- **L'auteur** du document ; la feuille est alors généralement un fichier `.css` associé par une balise `link`.
- **L'internaute**, qui peut appliquer une feuille de styles utilisateur (pour peu que son navigateur l'y autorise).

La cascade détermine comment sélectionner, propriété par propriété, la valeur spécifique à retenir pour un élément donné. Contrairement à une idée répandue, elle ne décrit pas le mécanisme d'héritage, bien plus simple et qui n'a pas besoin d'elle. Je reviendrai sur cet héritage un peu plus tard.

## Le sens de la cascade

La cascade se déroule approximativement de la façon suivante :

1. On récupère toutes les règles applicables (en vertu de leurs sélecteurs) à l'élément concerné, pour le média courant (saviez-vous qu'on peut restreindre une feuille de styles à certains médias seulement, comme l'écran, l'imprimante ou la vidéoprojection ?).

2. On les trie par importance (présence de la clause `!important`) et par origine, dans l'ordre suivant (du moins prioritaire au plus prioritaire) :
   a. règles « par défaut » venant du navigateur ;
   b. règles normales de l'internaute ;
   c. règles normales de l'auteur ;
   d. règles importantes de l'auteur ;
   e. règles importantes de l'internaute ;
3. À priorité égale, on les trie par **spécificité** (voir section suivante).
4. Enfin, on trie par ordre d'apparition : quand plusieurs définitions ont la même priorité et la même spécificité, la dernière est retenue.

La syntaxe `!important` en fin de déclaration de propriété est là pour améliorer l'accessibilité en garantissant à l'internaute la possibilité de remplacer une propriété qui nuit à son confort : il suffit de déclarer une autre valeur dans une feuille de styles utilisateur, dotée de l'attribut final `!important`.

## Calcul de la spécificité d'une règle

Voyons à présent comment calculer la spécificité, ou le « poids », d'une règle CSS. La spécificité est une valeur à quatre composantes : `a-b-c-d`. Ces composantes étant souvent de valeur inférieure à 10, on a en général un nombre décimal entre 0 et 9999. C'est le *poids*. Mais comment déterminer la valeur de chaque composante ?

**a.** La composante `a` vaut `1` pour un attribut HTML `style`, `0` pour une feuille externe.

**b.** La composante `b` est le nombre de sélecteurs d'ID impliqués (`#truc`).

**c.** La composante `c` est le nombre de sélecteurs sur attributs (`[truc]`, `.bidule`), ainsi que les pseudo-classes (par exemple `:hover` ou `:focus`).

**d.** La composante `d` est le nombre de sélecteurs de type (par exemple `h1`) et de pseudo-éléments (par exemple `:first-line`).

Vous avez besoin d'exemples ? D'accord !

**Tableau B–2** Exemples de calculs de spécificité

| Sélecteurs | a | b | c | d | Total |
|---|---|---|---|---|---|
| * | 0 | 0 | 0 | 0 | 0 |
| li | 0 | 0 | 0 | li → 1 | 1 |
| li:first-line | 0 | 0 | 0 | li, :first-line → 2 | 2 |
| ul li | 0 | 0 | 0 | ul, li → 2 | 2 |
| ul ol + li | 0 | 0 | 0 | ul, ol, li → 3 | 3 |

**Tableau B–2** Exemples de calculs de spécificité (suite)

| Sélecteurs | a | b | c | d | Total |
|---|---|---|---|---|---|
| `h1 + *[rel=up]` | 0 | 0 | `[rel=up]` → 1 | `h1` → 1 | 11 |
| `ul ol li.red` | 0 | 0 | `.red` → 1 | `ul`, `ol`, `li` → 3 | 13 |
| `li.red.level` | 0 | 0 | `.red`, `.level` → 2 | `li` → 1 | 21 |
| `#x34y` | 0 | `#x34y` → 1 | 0 | 0 | 100 |
| `style="..."` | 1 | 0 | 0 | 0 | 1000 |

## Que se passe-t-il avec la présentation dans HTML ?

Le détail obscur : si, horreur ultime, le HTML contenait des attributs de présentation (vous savez, ces momies que sont `bgcolor`, `border`, `align`, etc.), le navigateur peut les prendre en compte à condition de les considérer comme des règles placées en début de feuille de styles auteur, avec une spécificité zéro. En d'autres termes, elles seront prioritaires uniquement sur le style par défaut fourni par le navigateur et céderont le pas devant toute spécification de type CSS.

## L'héritage

L'héritage est un mécanisme par lequel des éléments voient certaines de leurs propriétés « hériter » leur valeur de celle utilisée par un élément conteneur.

Par exemple, si `body` précise une `font-size` de valeur `large`, les éléments `p` à l'intérieur de `body` (directement ou non) utiliseront la même valeur pour leur propre `font-size`, sauf instruction contraire.

Pour bien comprendre le rôle de l'héritage, il faut examiner comment le navigateur détermine la valeur concrète d'une propriété. Il ne suffit pas de prendre la valeur spécifiée dans la CSS, loin de là !

Pour commencer, sachez qu'un navigateur doit avoir défini une valeur concrète pour toutes les propriétés de tous les éléments lorsqu'il a achevé le *rendering* de la page ! Cela fait donc beaucoup de valeurs, même si on exclut les propriétés qui ne s'appliquent pas au média courant (par exemple, les propriétés de pagination lorsqu'on affiche à l'écran).

Cette valeur effective (*actual value* dans la spécification) est le résultat d'un calcul en quatre étapes. Dit comme cela, ça semble très compliqué, mais vous allez voir, c'est en fait plutôt naturel.

On commence par déterminer la **valeur spécifiée**, celle qui fait l'objet d'une déclaration ou d'une valeur par défaut. Pour cela, on cherche une valeur indiquée explicitement dans les CSS, en résolvant les conflits éventuels à l'aide de l'algorithme de cascade décrit ci-dessus.

C'est là que l'héritage peut entrer en jeu. Chaque propriété peut ou non en bénéficier automatiquement : cela est précisé dans la spécification CSS 2.1, à l'aide de la caractéristique *Inheritable*. Si on n'a pas trouvé de valeur explicite pour une propriété « héritable », on utilise alors la valeur calculée de l'élément parent (pour lequel toutes les propriétés ont déjà été résolues). On peut obtenir le même effet pour une propriété sans héritage automatique, en définissant la propriété avec la valeur spéciale `inherit`.

### De la valeur spécifiée à la valeur concrète

En revanche, s'il n'y a pas d'héritage (la propriété n'est pas définie et elle ne bénéficie pas d'un héritage automatique), on utilise alors la valeur initiale, indiquée dans la spécification CSS 2.1 pour cette propriété.

Deuxième étape : obtenir la **valeur calculée**. Il s'agit d'ajustements à la valeur spécifiée, par exemple la transformation d'URI relatifs en URI absolus, la conversion de tailles exprimées en unités relatives (`em`, `ex`...) en unités absolues (`px`), et tout autre changement qui n'a pas besoin d'un véritable *rendering* pour être déterminé. Cette valeur existe pout tout élément, même si la propriété ne s'applique pas à l'élément lui-même, afin de pouvoir utiliser l'héritage sur ses éléments descendants.

Troisième étape : passer à la **valeur utilisée**. On effectue alors les conversions résultant d'un *rendering*, comme transformer une largeur exprimée en pourcentage en une largeur absolue (puisqu'on connaît alors la largeur réelle de son conteneur).

Ultime étape, qui n'intéresse d'ailleurs pas toujours le développeur web : la **valeur effective (ou *concrète*)**, celle qui sera réellement employée. C'est le résultat des contraintes externes (notamment celles du périphérique) sur la valeur utilisée. Par exemple, une largeur absolue peut encore être de 2,8 pixels, alors qu'à l'écran, on ne peut afficher que des pixels entiers ; on arrondira donc à 3. Une police pourrait demander du 13 points, mais si le système de gestion des polices ne peut obtenir cette taille exacte, on ajustera sur la taille la plus proche, par exemple 12 points. Le document est en couleurs, mais l'imprimante n'autorise que les nuances de gris ; on interpolera donc les valeurs vers de telles nuances. Vous voyez l'idée.

Voilà certainement plus d'informations que vous ne souhaitiez en avoir. Il est vrai que ce sont surtout les deux premières étapes qui nous intéressent !

## Les modèles de boîte et de mise en forme visuelle

Je vous l'annonce sans ambages : nous n'irons pas dans tous les détails, loin s'en faut. Le modèle de boîte (*box model*) a fait couler tant d'encre et d'octets qu'on pourrait remplir un volume encyclopédique avec la somme des articles, chapitres et billets sur le sujet.

Je vais simplement couvrir la partie émergée, l'essentiel, le courant, sans me soucier des cas particuliers.

## Les côtés d'une boîte : ordre et syntaxes courtes

De très nombreuses propriétés CSS peuvent s'appliquer sur les quatre côtés d'une boîte, ou sur ses côtés horizontaux (haut et bas) et verticaux (droite et gauche), ou sur ses quatre côtés individuellement. Vous trouverez alors toujours des propriétés dites « abrégées » (*shorthand properties*), permettant de spécifier tous ces côtés d'un seul coup.

Les variantes et l'ordre sont toujours les mêmes, aussi les précisé-je ici une bonne fois pour toutes. Prenons l'exemple de la propriété `margin`, qui régit la marge externe d'une boîte, comme nous le verrons à la prochaine section.

Il existe des propriétés dédiées pour chaque côté : `margin-top`, `margin-right`, `margin-bottom` et `margin-left`. Il existe aussi la propriété courte `margin`, justement, qui peut prendre les trois formes suivantes.

**Tableau B–3** Variantes de définition d'une propriété courte

| Variante de format | Résultat |
|---|---|
| `margin: 1em;` | Même marge pour les quatre côtés : `1em`. |
| `margin: 1em auto;` | Marge de `1em` en haut et en bas, marge automatique (centrage de bloc) à droite et à gauche. |
| `margin: 1em 0 0.5em` | Marge haute de `1em`, droite et gauche de `0`, basse de `0.5em` |
| `margin: 0.5em 0 1em 2ex;` | Marge haute de `0.5em`, droite de `0`, basse de `1em`, gauche de `2ex`. |

Si vous avez du mal à retenir l'ordre, pensez à ceci : on part du haut et on suit les aiguilles d'une montre.

## Unités absolues et relatives

CSS fournit de nombreuses unités pour exprimer les tailles (de police, de bloc...). Certaines sont absolues et d'autres relatives. Certaines n'ont de sens que pour certains médias. En voici un récapitulatif.

**Tableau B–4** Unités de CSS 2.1

| Unité | Nature | Signification |
|---|---|---|
| % | relative | Pourcentage de la valeur calculée héritée (ou initiale, pour `body`). Par exemple dans un `body` avec `font-size: 1.5em`, un `p` avec `font-size: 150 %` aura en fait un `font-size` de `2.25em`. |

**Tableau B–4** Unités de CSS 2.1 (suite)

| Unité | Nature | Signification |
|---|---|---|
| `cm` | absolue | Centimètres. S'ajuste normalement aux résolutions de l'imprimante comme de l'écran (72 ou 96 ppp). |
| `em` | relative | La hauteur de la police courante, définie comme le *em square* typographique. En gros, en dépit de ses origines liées à la largeur, c'est à peu près la hauteur d'un caractère majuscule. Unité de choix pour les marges et espacements en général (même si j'ai plus tendance à limiter aux côtés haut et bas) et aux hauteurs de blocs. |
| `ex` | relative | La hauteur d'un caractère minuscule sans jambe (on utilise généralement « x »). Correspond souvent à la largeur moyenne d'un caractère : un conteneur de largeur `10ex` peut contenir environ 10 caractères.<br>Unité pertinente aussi pour les marges et espacements en général (je l'utilise surtout pour les bords droit et gauche, et les largeurs de blocs). |
| `in` | absolue | Pouces (*inches*). Je rappelle qu'un pouce vaut 2,54 cm. Même remarque que pour `cm`. |
| `mm` | absolue | Millimètres. Même remarque que pour `cm`. |
| `pc` | absolue imprimante | Picas (unité typographique). Cette merveilleuse unité semi-préhistorique vaut 12 points (voir `pt`). |
| `pt` | absolue imprimante | Points. Unité typographique qui n'a vraiment de sens que pour une CSS d'impression (média `print`). CSS 2.1 cale le point à 1/72 de pouce, soit environ 0,35 mm. Pas étonnant que beaucoup préfèrent le système métrique aux unités britanniques.<br>Notez que lors de l'impression, caler les tailles de fontes en points est bien plus pertinent qu'en `em`/`ex`, car cela favorise l'homogénéité sur un grand nombre d'imprimantes et de systèmes. |
| `px` | classée relative, mais plutôt « absolue écran » | Pixels. Unité un peu fourbe, car selon la résolution de l'écran (pas 1024 × 768, mais 72 ppp), un pixel sera plus ou moins gros. Sur Mac OS (avant OS X) notamment, les pixels étaient notoirement plus petits, ce qui rendait toute police `10px` illisible.<br>N'utilisez cette unité que pour des éléments dont la taille doit coller à des ressources non redimensionnables, comme des images. Pour du texte, je suis partisan de la proscrire, surtout sur des petites et moyennes tailles (moins de `30px`), car le texte ne peut alors pas être redimensionné (notamment agrandi) suivant les besoins de l'internaute. |

Je vous recommande très chaudement d'utiliser des **unités relatives** pour tout ce qui a vocation à suivre la taille du texte. Cela inclut souvent les bordures, les marges, les positionnements, et suivant l'esthétique retenue, parfois les espacements (*paddings*).

Ainsi, l'ensemble de votre page s'ajustera mieux lorsque l'internaute modifiera la taille de base des polices de caractères pour y voir plus clair, ce que tous les navigateurs répandus permettent de faire.

On donnera, notamment, priorité à `em`, `ex` et `%` à l'écran, et `pt` à l'impression !

Dernière remarque : pour une taille de zéro, on ne précise pas l'unité en général. C'est en effet mal vu par les esthètes (on parle de *bad form*), car zéro est zéro, quelle que soit l'unité.

## Marge, bordure et espacement

Une boîte est dotée, de l'extérieur vers l'intérieur, d'une marge (espacement externe à sa bordure), d'une bordure (avec une couleur, un type de trait et une épaisseur), d'un espacement (*padding* en anglais, espacement interne à la bordure), et tout au centre, de son contenu. Le schéma suivant illustre cette imbrication :

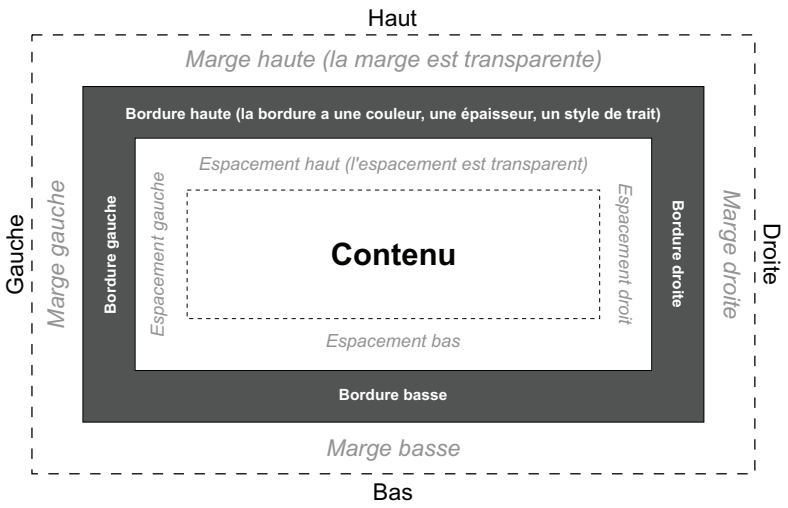



**Figure B–1**
Imbrication de marge, bordure, espacement et contenu

À présent, un point très important, car contre-intuitif : la largeur et la hauteur d'un élément sont en réalité les dimensions de son contenu. Prenons la règle suivante :

```
div#demo {
    width: 40ex;
    padding: 1ex;
    border: 1ex solid gray;
    margin: 1em 1ex;
}
```

Le `div` d'ID `demo` aura une « largeur visuelle » de 44ex ! En effet :

$$40\text{ex} + 2 \times 1\text{ex} + 2 \times 1\text{ex} = 44\text{ex}$$

Il s'agit de sa largeur de contenu, à laquelle on ajoute les largeurs des espacements gauche et droit ainsi que celles des bordures gauche et droite.

## Éléments en ligne et de type bloc

La plupart des éléments censés faire partie du flux d'un texte sont dits « en ligne » (*inline*). Ceux censés apparaître comme des blocs hors du flux de texte sont dits « de type bloc » (*block-level*).

Il existe en réalité toute une gamme de cas spécialisés, définis par la propriété `display`. CSS 2 a notamment ajouté une série de valeurs spécifiques aux composants de tableaux de données.

Des éléments naturellement en ligne (par exemple `span`, `em`, `strong`, `code`) peuvent devenir de type bloc à l'aide de la propriété `display`, et réciproquement.

Je mentionne cela parce qu'un élément en ligne a certaines limitations en terme de modèle de boîte. Par exemple, il n'a pas de marge verticale (haute ou basse).

## Éléments remplacés

Voici l'astuce du jour... Si vous lisez la spécification CSS 2.1, vous trouverez un peu partout le terme « élément en ligne non remplacé » (*non-replaced inline element*). On peut légitimement se demander à quoi rime ce « non remplacé », qu'on voit un peu partout. La réponse se cache à la section 3.1 de la recommandation, dont le titre, « Conformité », ne nous l'aurait pas laissé supposer.

Un élément remplacé est un élément hors du champ d'action du formateur CSS, en tout cas en ce qui concerne ses dimensions propres. Dans la pratique, c'est l'image obtenue par chargement d'une balise `img` ou alors le contenu chargé par une balise `object`. Tout autre élément en ligne est donc non remplacé, ce qui rend cette précision superflue dans la plupart des cas.

## Fusion des marges

Les marges expriment généralement un besoin d'espace autour d'un élément. Cependant, dans le cas des marges verticales, il est souvent inutile de cumuler les marges. Imaginons qu'on utilise la règle suivante :

```
p { margin: 1em 0; }
```

Cette règle demande des marges haute et basse, dites « marges verticales », de `1em` (hauteur moyenne d'un texte dans la police de caractères active), et pas de marges horizontales (droites et gauches).

Pourtant, lorsqu'on va enchaîner deux paragraphes, il serait curieux, voire agaçant, que ces marges se cumulent, produisant un espacement de 2 lignes de haut entre les paragraphes. Ce n'est généralement pas l'intention de l'auteur de la règle. Pour cette raison, dans de nombreuses situations, CSS 2.1 fusionne les marges verticales. Pour faire simple, lorsqu'on a deux marges verticales adjacentes, on n'utilise que la plus grande des deux.

Attention toutefois, ce mécanisme n'a pas cours dans tous les cas. La section 8.3.1 de la recommandation W3C détaille les cas concernés, dont la liste exhaustive va au-delà du cadre de cette annexe.

## Le modèle W3C et le modèle Microsoft

Voici la source de bien des soucis, en tout cas jusqu'à récemment (de nos jours, à peu près tous les professionnels ont pris le pli). Les développeurs de MSIE avaient à l'origine mal interprété le sens des propriétés `width` et `height`. Ils pensaient que ces dimensions incluaient l'espacement et la bordure.

Ce n'est pas déraisonnable : dans la pratique, quand vous faites référence aux dimensions d'une boîte, vous parlez des bords extérieurs de la boîte, pas de la taille du contenu calé dans la boîte par des protections en mousse ou en polystyrène. Hélas, ce n'est pas le sens retenu par la recommandation W3C.

Le souci est apparu dès que les autres navigateurs ont rattrapé MSIE 5 sur le terrain des CSS. Ces navigateurs implémentaient correctement la recommandation, et on s'est retrouvé avec des incohérences flagrantes entre l'affichage MSIE et celui des autres navigateurs. Je m'en souviens encore, et croyez-moi, ça nous a tous fait fulminer.

MSIE 6 a donc sorti une astuce, le *doctype switching*, expliquée au début de l'annexe A, astuce qui a rapidement été reprise par les autres navigateurs pour laisser l'auteur de la page décider de l'interprétation de `width` et `height`.

Lorsqu'une page déclare une DTD stricte (HTML 4.01 Strict ou XHTML 1.0 Strict, par exemple), le navigateur fonctionne en *Strict Mode*. On respecte alors la définition du W3C. Si une page ne déclare pas de DTD (rhôôô…) ou déclare une DTD non stricte, le navigateur utilise le *Quirks Mode*, qui simule le comportement de l'ancien MSIE en considérant que l'espacement et la bordure sont compris dans la taille.

En réalité, de nombreux navigateurs utilisent aussi le *Quirks Mode* pour émuler d'autres erreurs CSS de MSIE, et pas seulement de la version 5.5... Mais c'est un autre débat.

En conclusion : **déclarez toujours une DTD, et qu'elle soit stricte, bon sang !**

# Tour d'horizon des sélecteurs

CSS 2.1 définit de nombreux sélecteurs (et si vous voyiez CSS 3 !). En voici un tour d'horizon. Attention, certains ne sont pas pris en charge par MSIE 6, voire également ignorés par quelques autres navigateurs.

**Tableau B–5** Les sélecteurs de CSS 2.1

| Sélecteur | Description |
|---|---|
| `*` | Sélecteur universel. Correspond à tous les éléments. |
| `balise` | Sélecteur de type. Correspond aux éléments ayant ce nom de balise. |
| `balise1 balise2` | Sélecteur de descendants. Correspond aux éléments `balise2` situés quelque part à l'intérieur d'un élément `balise1`, à quelque niveau de profondeur que ce soit. |
| `.classe` | Sélecteur de classe. En (X)HTML, équivalent à `*[class~="classe"]` (voir plus bas). |
| `#unID` | Sélecteur d'ID. Correspond à l'élément dont l'attribut `id` possède la valeur `unID`. |
| `balise:link`<br>`balise:visited` | Pseudoclasses de lien. Correspond aux éléments `balise` (qui doivent être des sources de lien, donc généralement des éléments a) dont la cible n'a pas encore (ou a déjà, dans le second cas) été visitée. |
| `balise:hover`<br>`balise:active`<br>`balise:focus` | Pseudoclasses dynamiques. Correspond aux éléments `balise` à certains moments de l'interaction utilisateur. Respectivement, il s'agit d'un survol souris, d'une activation (clic ou touche Entrée, par exemple) ou de l'obtention du focus (arrivée sur l'élément par tabulation, par exemple). |
| `balise1 > balise2` | Sélecteur de fils. Correspond aux éléments `balise2` dont l'élément père est un élément `balise1`. Plus restrictif donc que l'espace, qui n'a pas de limite de profondeur. |
| `balise1 + balise2` | Sélecteur d'élément adjacent. Correspond à un élément `balise2` dont le précédent élément frère est de type `balise1`. Par exemple, `h2 + p` sélectionne les paragraphes qui suivent immédiatement des titres de deuxième niveau. |
| `balise:first-child` | Pseudoclasse de premier fils. Correspond à tout élément `balise` étant le premier élément fils de son élément père. Par exemple, permet de sélectionner le premier paragraphe d'une série. |
| `balise[attr]` | Sélecteur d'attribut. Correspond à tout élément `balise` ayant un attribut `attr` (quelle qu'en soit la valeur, même vide). |
| `balise[attr="value"]` | Sélecteur d'attribut. Correspond à tout élément `balise` dont l'attribut `attr` a exactement la valeur `value` (sensible à la casse). Les guillemets sont obligatoires. |
| `balise[attr~="part"]` | Sélecteur d'attribut. Correspond à tout élément `balise` dont l'attribut `attr` contient une série de valeurs séparées par des espaces, et dont l'une est `part`. Voir l'équivalence du sélecteur de classe. |
| `balise[attr\|="prefix"]` | Sélecteur d'attribut. Correspond à tout élément `balise` dont l'attribut `attr` contient une série de valeurs séparées par des tirets (-), et dont la première est `prefix`. Utile pour l'attribut `lang`, par exemple. |
| `:lang(code)` | Pseudoclasse de langue. Sélectionne tout élément dont la langue correspond à `code`. |

## Groupement

Afin de limiter la duplication (cauchemar de la maintenance), on peut affecter une série de propriétés à plusieurs sélections (combinaisons de sélecteurs) disjointes, en les séparant simplement par des virgules.

```
h1, h2, h3, h4, h5, h6 {
    font-weight: normal;
}
```

Dans cet exemple, tous les titres utiliseront une épaisseur normale au lieu de la graisse par défaut.

## Pseudo-éléments

Un pseudo-élément correspond à une portion de texte à l'intérieur d'un élément, ou à un contenu généré dynamiquement. En tout état de cause, il s'agit d'un fragment du document qui ne correspond pas à un élément, mais fait néanmoins l'objet d'une sélection.

**Tableau B–6** Les pseudo-éléments de CSS 2.1

| Pseudo-élément | Description |
|---|---|
| `E:first-line` | Première ligne d'un corps de texte. E doit être élément de type bloc, `inline-block`, `table-caption` ou `table-cell`. On évitera donc `span`, par exemple. |
| `E:first-letter` | Première lettre d'un contenu textuel, sauf si cette lettre est précédée d'un autre contenu (image, tableau en ligne…). Toutes les propriétés CSS ne s'appliquent pas (voir section 5.12.2 de la recommandation). Principalement utilisé pour faire des lettrines. S'applique aux mêmes types d'élément que `:first-line`, plus au type `list-item`. |
| `E:before`<br>`E:after` | Permettent de générer un contenu avant ou après le texte normal de l'élément, à l'aide de la propriété CSS `content`. Très utilisés pour assortir les liens de petits glyphes ou d'informations sur la langue (attribut HTML `hreflang`)… Très pratique aussi pour une feuille de styles dédiée à l'impression (média `print`), qui peut ainsi afficher, à côté du texte des liens, les URL ciblées. |

# Tour d'horizon des propriétés

Il est temps de faire le tour des propriétés. Pour toutes celles qui ont des versions courtes, j'ai adopté une notation à lignes multiples, avec les possibilités entre crochets, listées dans l'ordre de leur apparition au sein de la propriété courte.

Chaque ligne supplémentaire constitue une composante optionnelle. La propriété `border` est la plus riche, puisqu'elle va d'une seule propriété consolidée (`border`) jusqu'aux propriétés très ciblées (par exemple `border-left-style`).

Je précise aussi une bonne fois pour toutes que toutes les propriétés peuvent avoir la fameuse valeur `inherit`. Je ne le préciserai pas dans les tableaux.

## De l'art de réaliser des CSS légères

CSS fournit de nombreuses syntaxes courtes pour simplifier et alléger vos feuilles de style. Nous avons déjà évoqué ce mécanisme dans le cadre des propriétés de marge, de bordure et d'espacement. Allez donc consulter la recommandation pour le détail des autres propriétés courtes, notamment `font`, une vraie merveille !

Par ailleurs, toute propriété de couleur peut avoir les syntaxes suivantes :

- `transparent`, mot réservé, utile entre autres pour préciser une couleur de fond chaque fois qu'on précise une couleur de texte (évitant ainsi les avertissements des validateurs).
- `#rrvvbb`, où l'on représente les composantes rouge, verte et bleue par une valeur hexadécimale comprise entre `00` et `ff`, c'est-à-dire allant de 0 à 255.
- `#rvb`, lorsque les composantes sont des doublons (par exemple `11`, `44` ou `ff`). Ces couleurs sont censées être plus fiables en rendu que les variantes détaillées (`#463`, équivalent de `#446633`, est censée être plus « garantie » que `#426431`, qui lui ressemble à s'y méprendre).
- `rgb(red, green, blue)`, où les composantes sont indiquées en base décimale, de 0 à 255. À mon sens la plus verbeuse, donc à éviter.
- Mot réservé de couleur, par exemple `white` ou `red`. La recommandation définit en section 4.3.6 les 17 noms autorisés et leurs valeurs exactes.

N'oubliez pas qu'une feuille CSS plus courte est aussi plus rapide à charger, et pas forcément plus difficile à lire !

## Propriétés du modèle de boîte : marges, espacements et bordures

**Tableau B–7** Propriétés du modèle de boîte en CSS 2.1

| Propriété | Description |
|---|---|
| `border -[top\|right\| bottom\|left] -[width\|style\|color]` | Bordures. L'épaisseur est exprimée en tant que taille ou à l'aide des mots réservés `thin`, `medium` et `thick`. On compte pas moins de 10 styles, les plus courants étant `solid`, `dotted`, `dashed` et `none` (qui diffère de `hidden` pour les cellules de tableaux !).<br>Au fait, MSIE 7 cesse de dessiner les bordures `dotted` d'épaisseur `1px` en `dashed`. Enfin ! |
| `margin -[top\|right\| bottom\|left]` | Marges, donc espacements extérieurs. |
| `padding -[top\|right\| bottom\|left]` | Espacements intérieurs. |

## Propriétés de formatage visuel : positionnement, largeur, hauteur, baseline

**Tableau B–8** Propriétés de formatage visuel en CSS 2.1

| Propriété | Description |
|---|---|
| `clear` | Contrôle le flux autour d'un élément flottant. Peut valoir `none`, `left`, `right` ou `both`. |
| `clip` | Permet de restreindre la portion affichée d'un élément. Par défaut `auto`, donc affiche toute la boîte. Sinon, peut définir un rectangle avec `rect(top, right, bottom, left)`. |
| `direction` | Indique le sens du texte dans l'élément. Vaut `ltr` (de gauche à droite, par défaut) ou `rtl`. |
| `display` | Indique le type de boîte de l'élément. Les valeurs les plus courantes sont `inline`, `block` et `none`, mais il y en a 13 autres, dont 10 relatives aux tableaux ! |
| `float` | Rend un élément flottant, c'est-à-dire l'extrait du flux normal du texte pour aller se caler quelque part contre les bords du conteneur, le flux du texte s'enroulant autour de lui. Suivant le côté du flottement, vaut `left`, `right` ou bien sûr `none`, sa valeur par défaut. |
| `height, min-height, max-height` | Régissent la hauteur d'un élément, en collaboration avec `overflow`. |

**Tableau B–8** Propriétés de formatage visuel en CSS 2.1 (suite)

| Propriété | Description |
|---|---|
| `line-height` | Hauteur minimale de ligne, ce qui inclut le texte et un certain espacement au-dessus et en dessous du texte. Vaut `normal` par défaut (basé sur les caractéristiques de la police de caractères), sinon un nombre ou un pourcentage (multiplicateur de `font-size`), ou encore une taille spécifique (avec des unités, par opposition à un simple nombre). |
| `overflow` | Gère le dépassement de contenu vis-à-vis des dimensions souhaitées de la boîte. Vaut par défaut `visible` : la boîte s'adapte, notamment en hauteur. Peut aussi valoir `hidden` (le contenu est tronqué), `scroll` (barres de défilement présentes qu'on en ait besoin ou non) ou `auto` (barres de défilement, si besoin). |
| `position` | Définit le positionnement de l'élément. Vaut `static` par défaut (positionnement défini par le navigateur suivant les contraintes actives), mais peut aussi valoir `absolute`, `relative` ou `fixed` (lequel est enfin pris en charge par MSIE 7). |
| `top`, `right`, `bottom`, `left` | Pour un élément positionné, fournit les positions de ses extérieurs de marge (absolues ou relatives, suivant `position`). |
| `unicode-bidi` | Permet à la valeur de direction de fonctionner également sur un élément en ligne. |
| `vertical-align` | Fournit l'alignement vertical du contenu en ligne (ou du contenu d'une cellule de tableau). Les valeurs ont toutes rapport aux métriques de typographie : le défaut est `baseline`, et on en a 7 autres dont `middle` et `text-top`, plus la possibilité d'une taille ou d'un pourcentage. |
| `visibility` | Affiche ou masque un élément, tout en conservant l'espace occupé (contrairement à `display`). Outre les valeurs `visible` et `hidden`, une valeur `collapse` a un sens spécial pour les lignes et groupes (de lignes ou de colonnes) des tableaux. |
| `width`, `min-width`, `max-width` | Régissent la largeur d'un élément. |
| `z-index` | Utilisée uniquement sur les éléments positionnés. Indique leur position « verticale », entre l'œil de l'internaute et le document si vous préférez. Peut valoir `auto` (défaut) ou un numéro. Règle les questions de recouvrement entre éléments se superposant, par exemple lors d'un glisser-déplacer. |

## Propriétés de contenu généré automatiquement

CSS 2 a introduit la notion de contenu automatique, principalement sur trois axes :

- Du contenu entièrement synthétisé par la CSS, généralement présent devant ou derrière le contenu natif de l'élément.
- Des indices incrémentaux ; principalement pour les listes, mais aussi pour les titres par exemple (enfin des titres numérotés automatiquement, et hiérarchiquement si on le veut !).
- Des symboles ou images destinés aux listes à puces.

**Tableau B–9** Propriétés de contenu automatique en CSS 2.1

| Propriété | Description |
|---|---|
| `content` | Remplace le contenu de l'élément. On l'utilise principalement sur des sélecteurs de pseudo-éléments `:before` ou `:after`, pour ajouter plutôt que remplacer. Peut valoir une foule de choses : un texte fixe, une ressource externe dont on fournit l'URI, l'état d'un compteur, la valeur d'un attribut de l'élément (comme `hreflang`), l'ouverture ou la fermeture des guillemets ou encore le contrôle du niveau d'imbrication de ceux-ci. |
| `counter-increment` | Incrémente un ou plusieurs compteurs déjà définis. Utilise des paires nom × incrément, l'incrément étant optionnel et valant par défaut `1` (un). Voir l'exemple très parlant de la section 12.4 de la recommandation. |
| `counter-reset` | Définit ou réinitialise un ou plusieurs compteurs, en précisant éventuellement leurs nouvelles valeurs. Même syntaxe que `counter-increment`. |
| `quotes` | Définit les paires de guillemets à utiliser dans l'élément, niveau par niveau (pour des utilisations imbriquées). La valeur par défaut dépend du navigateur. `none` supprime tout guillemet (pas très utile…). Sinon, on précise des paires de textes, par exemple `'«\00a0' '\00a0»' '"' '"'` pour le français (la séquence `\00a0` représente une espace insécable). À utiliser en combinaison avec les valeurs `open-quote` et `close-quote` de `content`. |
| `list-style-[type\|position\|image]` | Régit l'apparence d'une liste. Le type peut avoir pas moins de 15 valeurs dont 11 de numérotation (listes ordonnées), 3 de puces, et `none` pour retirer les puces ou numéros. La position vaut `inside` ou `outside` (défaut), indiquant si les puces ou numéros s'affichent à l'intérieur ou à l'extérieur de la boîte des contenus. Enfin, `image` permet de remplacer les puces classiques par une image quelconque, dont on fournit l'URI. |

## Propriétés de pagination

Ces propriétés ne s'appliquent que dans le cadre d'un *rendering* sur média paginé, ce qui revient à dire : à l'impression (média `print`). On dispose alors d'une règle spéciale nommée `@page`, qui désigne la page physique et non pas un élément du document.

**Tableau B–10** Propriétés de pagination en CSS 2.1

| Propriété / pseudoclasse | Description |
|---|---|
| `:first, :left, :right` | Pseudoclasses utilisables sur `@page` pour régler par exemple les marges indépendamment pour la première page, les pages gauches (paires pour un document se lisant de gauche à droite) et les pages droites (impaires). |
| `margin-[top\|right\|bottom\|left]` | Marges classiques. Je les remets ici car elles ont un sens particulier lorsqu'on les applique à `@page` : ce sont alors les marges d'impression. |

**Tableau B–10** Propriétés de pagination en CSS 2.1 (suite)

| Propriété / pseudoclasse | Description |
|---|---|
| `orphans` | Nombre minimum de lignes d'un bloc qui doivent apparaître en bas de page (lignes orphelines). Si le bas de la page est trop plein pour cela, le bloc démarre à la page suivante. La valeur est numérique, et vaut par défaut 2. |
| `page-break-after`<br>`page-break-before`<br>`page-break-inside` | Régissent les sauts de page. Appliquées à un élément, elles déterminent ce que le navigateur a le droit de faire après, avant et à l'intérieur de l'élément, respectivement. Les deux premières peuvent valoir `auto` (défaut), `always` (force le saut), `avoid` (éviter à tout prix), `left` ou `right` (forcer le saut jusqu'à une page gauche ou droite, par exemple pour un début de chapitre). La dernière ne peut valoir que `auto` ou `avoid`. |
| `widows` | Nombre minimum de lignes d'un bloc qui doivent apparaître en haut de page (lignes veuves). Si le bloc qui devait démarrer à la page précédente n'avait pas assez de lignes restantes pour ce haut de page, il démarre sur cette nouvelle page. La valeur est numérique, et vaut par défaut 2. |

## Propriétés de couleurs et d'arrière-plan

**Tableau B–11** Propriétés de couleurs et d'arrière-plan en CSS 2.1

| Propriété | Description |
|---|---|
| `background`<br>`-[color\|image\|repeat\|`<br>`  attachment\|position]` | Définit l'arrière-plan d'un élément. On a d'abord sa couleur, puis une image à utiliser, son mode de répétition (« mosaïque » : `repeat` par défaut, mais connaissez-vous `repeat-x`, `repeat-y` et `no-repeat` ?), son mode de défilement (`scroll` par défaut, mais connaissez-vous `fixed` ?) et sa position initiale dans l'élément (par exemple `top right`, ou `15% bottom`, ou `2cm top`). |
| `color` | Couleur du texte. |

## Propriétés de gestion de la police de caractères

**Tableau B–12** Propriétés de la police de caractères en CSS 2.1

| Propriété | Description |
|---|---|
| `font`<br>`-[style\|variant\|`<br>`  weight\|size\|family]` | Régit la police de caractères. Voilà un cas où bien apprendre la syntaxe consolidée de la propriété courte (`font`) est payant ! Le style est généralement `normal` ou `italic`, la variante `normal` ou `small-caps`, le poids `normal` ou `bold` (presque aucun système de polices ne gère plus de 2 degrés de graisse…), la taille a une syntaxe plus complexe (voir ci-après) et la famille aussi.<br>La propriété courte peut aussi utiliser juste un nom réservé de police système. |

### Taille de police

La taille peut être exprimée avec un mot-clef absolu, un mot-clef relatif, une taille classique ou un pourcentage de la taille de référence.

- Absolus : `xx-small`, `x-small`, `small`, `medium` (défaut), `large`, `x-large`, `xx-large`. Le rapport entre les valeurs successives n'est pas fixe, en particulier aux extrêmes.
- Relatifs : `smaller`, `larger`. Décale la taille sur l'échelle des absolus, et si on est déjà sur un extrême, interpole au mieux.

Dans le cas où la taille est précisée au sein de la propriété courte `font`, on dispose d'une syntaxe spéciale qui permet de faire d'une pierre deux coups en précisant à la volée le `line-height` : on utilise `font-size/line-height`, c'est très pratique et cohérent. Voir l'exemple un peu plus loin.

### Famille de polices

La famille de polices permet de définir une série de polices à tenter d'utiliser, par ordre décroissant de préférence. Il s'agit de noms de polices que le navigateur va chercher sur le système de l'internaute. Les noms à espaces doivent être entre guillemets. Il est fortement conseillé, pour des raisons d'accessibilité, de terminer la série par un des noms génériques :

- `serif` : police à empattements, par exemple Times ;
- `sans-serif` : police sans empattements, par exemple Arial ;
- `cursive` : police à pleins et déliés, par exemple Monotype Corsiva ou Zapf Chancery ;
- `fantasy` : police « délirante », décalée, amusante ;
- `monospace` : police à chasse fixe, par exemple Courier.

Voici un exemple :

```
font-family: "Bitstream Vera Sans Mono", Monaco, monospace;
```

### Tout spécifier d'un coup !

Enfin, voici un premier exemple de propriété courte, qui résume tout ce qu'on a besoin de dire sur la police :

```
font: 115%/1.4em "Bitstream Vera Sans Mono", Monaco, monospace
```

Ici, on ne précise ni le style, ni la variante, ni le poids, mais directement la taille (`115 %`), la hauteur de ligne (`1.4em`) et la famille.

Attention : certaines configurations partielles de style, variante et poids peuvent faire apparaître une ambiguïté : ainsi, si vous n'en précisez qu'une ou deux et utilisez la valeur `normal` pour la dernière, comment savoir de quelle propriété on parle ? Il faut alors être explicite, quitte à utiliser `inherit` pour maintenir les valeurs des propriétés qu'on ne souhaite pas affecter.

```
font: normal 1.5em/1.8em sans-serif;
```

C'est ambigu : est-ce le type, la variante ou le poids qui est normal ?

```
font: italic normal inherit/120%;
```

Et là, est-ce la variante ou le poids ?

```
font: italic inherit normal inherit/120%;
```

Ici, pas de doute : c'est le poids, et on ne touche pas à la variante.

Dernier point : les noms réservés de polices système, qui configurent toute la police d'un coup. Cela permet de réaliser une interface cohérente avec celle du système d'exploitation. Les valeurs possibles sont :

- `caption`, utilisée pour les contrôles (composants visuels) à libellés (par exemple les boutons, les listes déroulantes).
- `icon`, utilisée pour labéliser les icônes (par exemple sur le bureau).
- `menu`, utilisée dans les menus (barres ou menus contextuels).
- `message-box`, utilisée pour les boîtes de dialogue à message.
- `small-caption`, utilisée pour labéliser les petits contrôles (e.g. comme les boutons de barre d'outils).
- `status-bar`, utilisée par les barres d'état.

Il suffit donc d'utiliser par exemple :

```
div#status { font: status-bar; }
```

pour avoir un élément avec la fonte exacte des barres d'état sur le système de l'internaute.

## Propriétés de gestion du corps du texte

**Tableau B–13** Propriétés du corps du texte en CSS 2.1

| Propriété | Description |
|---|---|
| `letter-spacing` | Espacement entre les lettres. Vaut zéro par défaut. Je conseille vivement de n'utiliser que des tailles en unité `ex`, très adaptée. Rien qu'à `0.1ex`, on voit l'effet. |
| `text-align` | Alignement du texte dans un bloc. Peut valoir `left`, `center`, `right` ou `justify`. |
| `text-decoration` | Régit l'apparence de traits au-dessus ou en dessous du texte. Peut valoir `none` (par défaut), `underline` (soulignement), `overline` (trait au-dessus du texte), `line-through` (texte barré) ou… oserai-je le dire ? Bon, `blink`, mais gare au premier qui s'en sert ! C'est moche et tout le contraire d'accessible ! |
| `text-indent` | Indentation de la première ligne d'un bloc de texte. |
| `text-transform` | Gère la casse. Peut valoir `none` (défaut), `capitalize` (initiales en majuscules, autres lettres inchangées), `uppercase` (majuscules) ou `lowercase` (minuscules). Voir aussi `font-variant` dans le tableau B-12. |
| `white-space` | Gestion des espacements dans le corps du texte. Voir ci-après. |
| `word-spacing` | Espacement entre les mots. Même remarque que pour `letter-spacing`. |

### L'espacement dans le corps du texte

La propriété `white-space` mérite tout de même une petite explication.

En temps normal, le *rendering* d'un contenu textuel retire tous les espacements (espaces, tabulations, retours chariot, etc.) au début et à la fin du texte, ramène toute autre série d'espacements à une seule espace classique (y compris les retours chariot), et va à la ligne quand c'est nécessaire (quand le texte arrive en bout de largeur du bloc conteneur).

On a donc trois comportements distincts : la réduction des espacements, le respect des retours chariot d'origine et le passage à la ligne pour honorer la largeur du conteneur (*wrapping*). Voici les définitions succinctes des valeurs possibles pour `white-space` :

**Tableau B–14** Valeurs de white-space

| Valeur | Réduction | Retours chariot | Wrapping |
|---|---|---|---|
| `normal` | Oui | Non | Oui |
| `pre` | Non | Oui | Non |
| `nowrap` | Oui | Non | Non |
| `pre-wrap` | Non | Oui | Oui |
| `pre-line` | Oui | Oui | Oui |

## Propriétés des tableaux

**Tableau B–15** Propriétés des tableaux en CSS 2.1

| Propriété | Description |
|---|---|
| `border-collapse` | Gère la fusion des bordures entre cellules adjacentes, et entre les cellules et la bordure du tableau. Désactivée par défaut (`separate`), peut être activée avec la valeur `collapse`. Je trouve ça beaucoup plus joli, personnellement… |
| `border-spacing` | Espacement entre bordures de cellules (si les bordures ne sont pas fusionnées). Peut contenir une ou deux tailles. Dans le second cas, distingue entre distances horizontale et verticale. |
| `caption-side` | Position du titre : au-dessus (`top`, défaut) ou en dessous (`bottom`). |
| `empty-cells` | Affiche ou masque les cellules vides. Par défaut, affiche (`show`). On les masque avec `hide`. |
| `table-layout` | Mode de calcul des largeurs du tableau et des cellules. Le mode par défaut, `auto`, est celui auquel on s'attend : il adapte les largeurs en fonction des contenus de cellules. L'autre mode, `fixed`, utilise uniquement les spécifications de largeur pour le tableau, les colonnes, les bordures et l'espacement entre cellules. Il est plus rapide mais rend généralement moins bien. |

## Propriétés de l'interface utilisateur

**Tableau B–16** Propriétés de l'interface utilisateur en CSS 2.1

| Propriété | Description |
|---|---|
| `cursor` | Détermine l'aspect du curseur souris à utiliser lorsque celui-ci survole l'élément. Extrêmement utile en terme d'ergonomie. Les valeurs sont détaillées à la section 18.1 de la recommandation, mais je cite les principales : `auto` (défaut), `default` (curseur classique du système), `pointer` (comme pour un lien), `move` (idéal pour glisser-déplacer), `help` (idéal pour `abbr` et `acronym`). |
| `outline -[color\|style\| width]` | Affiche une délimitation autour d'un élément. Diffère d'une bordure en ce qu'elle n'occupe pas de place dans le modèle de boîte : elle est dessinée au-dessus du bord extérieur de la bordure. Elle ne comprend donc pas les marges. Peut être utile pour signaler qu'un élément est prêt à recevoir un dépôt lors d'un glisser-déplacer, mais encore mal prise en charge… |

# Pour aller plus loin…

## Livres

*CSS 2 – Pratique du design web*
Raphaël Goetter
Eyrolles, juin 2005, 324 pages (de bonheur)
ISBN 2-212-11570-9

*Le zen des CSS*
Dave Shea
Eyrolles, novembre 2005, 296 pages
ISBN 2-212-11699-3

*Cascading Style Sheets: The Definitive Guide*
Eric Meyer
O'Reilly, novembre 2005, 508 pages
ISBN 0-596-00525-3

*Mémento CSS*
Raphaël Goetter
Eyrolles, novembre 2005, 14 pages (de mémento)
ISBN 2-212-11726-4

## Sites

- La recommandation CSS 2.1, évidemment. Attention, seule la version anglaise a valeur de référence :
  - http://www.w3.org/TR/CSS21/
  - Version française de la 2.0 : http://www.yoyodesign.org/doc/w3c/css2/cover.html
- L'excellent site géré par Raphaël Goetter (jetez-vous sur son livre !), Alsa Créations : http://www.alsacreations.com/
- Le CSS Zen Garden, pour se convaincre qu'avec le même XHTML, on peut changer complètement de tête : http://csszengarden.com/
- A List Apart brille aussi en CSS : http://alistapart.com/
- Des styles décalés, les frontières de l'impossible : CSS Play. http://moronicbajebus.com/playground/cssplay/
- Roger Johansson a plein de choses à vous raconter sur CSS, si vous allez au 456 Berea St. : http://www.456bereastreet.com/
- CSS Beauty : http://www.cssbeauty.com/

# C

# Le « plus » de l'expert : savoir lire une spécification

Il existe trois catégories de développeurs web. D'abord, ceux qui semblent toujours tout savoir, quitte à ne vous répondre que quelques instants plus tard, et qui expriment leur réponse avec un air d'autorité confiante dans l'exactitude de leur propos, laquelle se vérifie en effet à chaque fois. Ensuite ceux qui n'ont pas toutes les réponses, et semblent ne pas trop savoir où les chercher. Enfin, ceux qui manifestement n'ont qu'une compétence empirique : leurs pages « tombent en marche ».

Puisque vous avez ce livre entre les mains, on peut penser que vous souhaitez ne pas faire partie de la dernière catégorie, ni même avoir à travailler avec de telles personnes. Vous connaissez probablement un certain nombre de développeurs entrant dans la deuxième catégorie ; c'est peut-être d'ailleurs votre cas. Quant à ceux de la première catégorie, ces puits de connaissance apparemment sans fond, ils suscitent l'admiration de tous. Cette annexe vous propose modestement de tenter d'en faire partie.

# De Intérêt d'aller chercher l'information à la source

Il y a deux intérêts fondamentaux à être capable d'aller chercher l'information à la source. Le premier est parfaitement objectif et professionnel. Le second est plus subjectif et, comment dire... plus humain.

## Certitude et précision

Les bonnes spécifications ont plusieurs qualités. Intrinsèquement d'abord, elles constituent le document de référence pour une technologie : elles font donc autorité sur la question. Utiliser correctement la technologie revient à l'utiliser conformément à la spécification. Si cela ne fonctionne pas alors que c'est exactement comme la spécification le demande, on sait que c'est notre environnement de travail qui est fautif et non notre code.

Bien sûr, cela peut simplement vouloir dire qu'on utilise du CSS 2.1 sur MSIE 6, auquel cas on ne peut pas laisser les choses telles quelles, il faudra trouver une solution.

Une bonne spécification est par ailleurs précise : elle doit indiquer tous les cas particuliers, toutes les nuances, tous les problèmes potentiels. Elle ne doit pas laisser de zone d'ombre. Généralement, cela signifie que la spécification est aride, ou en tout cas particulièrement verbeuse. Les excellentes spécifications arrivent à conjuguer une précision totale et une bonne lisibilité.

Quiconque maîtrise un sujet sur le bout des doigts le sait bien : il est très agréable de discuter de quelque chose qu'on connaît parfaitement. En particulier s'il s'agit d'aider quelqu'un à comprendre, à utiliser, à mettre au point. L'expertise est agréable. Être véritablement spécialiste d'un domaine précis et mettre cette expertise en œuvre est très agréable.

Savoir utiliser les spécifications d'une technologie procure ce que les moyens de deuxième main (livres, didacticiels, articles, ateliers, etc.) ne peuvent que difficilement donner : l'accès à une maîtrise totale, ou en tout cas l'accès à l'information totale.

## « On m'a dit que là-dessus, c'est toi qui sais tout » : l'expertise

Il existe un deuxième avantage, plus humain celui-là. À force de faire preuve d'expertise sur un sujet donné, vous allez être connu pour cela. Un cercle toujours plus large de collègues et connaissances va faire l'association d'idées entre ce sujet et vous. Et de plus en plus, lorsqu'on aura besoin d'une information précise, pointue, fiable, on viendra vous voir.

Être un expert en XHTML, en balisage sémantique, en accessibilité, en CSS 2.1, en DOM niveau 2, en JavaScript et en Ajax ne vous apportera pas fortune et gloire (quoique...), mais dans votre travail, cela risque fort de vous apporter autre chose : *vous allez devenir indispensable*.

Rien que pour l'ego, c'est agréable. Mais cela peut aussi modifier vos prétentions salariales, embellir votre CV et vous ouvrir de nouvelles opportunités.

# Les principaux formats de spécifications web

Dans le cadre des technologies web, les spécifications utilisent essentiellement quatre formats.

## Les recommandations du W3C

Les technologies gérées par le W3C sont publiées sous la forme de recommandations. On trouve deux abréviations courantes : TR (*Technical Report*) et REC (*Recommendation)*. Il s'agit du statut finalisé d'une spécification, qui passe auparavant par plusieurs stades WD (*Working Draft*, ou ébauche).

La plupart des « langages descriptifs » du Web sont des technologies W3C. Citons principalement (X)HTML, XML, CSS, DOM, MathML, RDF, SMIL, SOAP, SVG, XPath et XSL/XSLT.

## Les grammaires formelles de langages à balises : DTD et schémas XML

Les langages à balises disposent d'une grammaire formelle, très pratique pour retrouver rapidement le détail des attributs et éléments autorisés dans un contexte précis.

Suivant le cas (principalement selon l'origine et l'ancienneté du langage visé), la grammaire utilise soit une DTD (*Document Type Definition*), qui est un document SGML de syntaxe assez facile, soit un schéma XML, un document... XML, potentiellement plus puissant mais souvent très, très verbeux.

Par exemple, HTML et XHTML 1.0 utilisent des DTD, tandis que XHTML 1.1+, WSDL et XLink utilisent des schémas XML.

Il est à noter qu'un juste milieu existe au travers de la syntaxe Relax NG. Bien que celle-ci gagne chaque jour en popularité, elle n'est pas encore adoptée par les principaux organismes de standardisation, notamment le W3C.

### Les RFC de l'IETF : protocoles et formats d'Internet

Enfin, la plupart des protocoles et formats de données du Web sont gérés et normalisés par l'IETF (*Internet Engineering Task Force*), un très large regroupement de professionnels qui est, véritablement, à l'origine d'Internet (premiers standards en 1969 !).

Les standards de l'IETF sont collectivement appelés les RFC (*Request For Comments*), et utilisent un format texte en 72 colonnes, très simple à lire. Il en existe plus de 4 600, dont plus de 300 ont vu le jour entre janvier et août 2006.

On y trouve notamment les spécifications pour HTTP, SMTP, POP, IMAP, FTP, Telnet, ICMP (la commande `ping`), SSL...

## S'y retrouver dans une recommandation W3C

Commençons par explorer la structure d'une recommandation W3C. Le site officiel du W3C est `http://w3.org`. Vous y trouverez toutes les spécifications dans leur version anglaise, seule à être garantie : des traductions existent souvent, mais leur qualité n'est pas validée en détail par le W3C, même si ce dernier fournit un lien vers celles-ci depuis la version originale.

### URL et raccourcis

Les recommandations ont souvent une URL assez longue, car elle contient quelques répertoires et surtout une date de version. Voici quelques exemples, qui donnent une idée des dégâts :

- http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/
- http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/
- http://www.w3.org/TR/2002/CR-css-mobile-20020725
- http://www.w3.org/TR/2001/REC-xhtml11-20010531/
- http://www.w3.org/TR/2003/REC-SVG11-20030114/
- http://www.w3.org/TR/1999/REC-xpath-19991116

Vous y observez le préfixe `/TR/` en début de chemin, comme pour toutes les recommandations. On a ensuite l'année de parution, puis le chemin des documents de la spécification, qui démarre généralement par `REC` (on a ici aussi `CR`, pour *Candidate Recommendation*, dernier stade avant finalisation. On peut aussi trouver `WD` : *Working Draft*, ou `NOTE`, pour les documents à valeur informative).

Les chemins des spécifications précisent toujours la date exacte de parution du document à la fin, au format `aaaammjj`. On peut donc déduire, par exemple, que la dernière version du DOM niveau 2 HTML date du 19 janvier 2003, alors que celle du DOM niveau 2 noyau date du 13 novembre 2000.

Pour de nombreuses spécifications, il existe toutefois une URL « raccourcie », qui amène automatiquement à la dernière version. Voici les équivalents des URL mentionnées plus haut, avec quelques autres :

- http://www.w3.org/TR/html401/
- http://www.w3.org/TR/xhtml1/
- http://www.w3.org/TR/CSS21/
- http://www.w3.org/TR/DOM-Level-2-Core/
- http://www.w3.org/TR/DOM-Level-2-HTML/
- http://www.w3.org/TR/css-mobile
- http://www.w3.org/TR/xhtml11/
- http://www.w3.org/TR/SVG11/
- http://www.w3.org/TR/xpath

En fait, cela revient le plus souvent à supprimer l'année, le préfixe REC et la date en fin de nom. Notez qu'il y a parfois des *slashes* (/) terminaux, et parfois non. Dans certains cas (HTML 4.01, XHTML 1.0, CSS 2.1...) cela n'a aucune importance, dans d'autres vous obtiendrez une page intermédiaire.

## Structure générale d'une recommandation

Une recommandation W3C a toujours la même structure générale.

D'abord l'en-tête :

1. titre avec version ;
2. statut (recommandation, ébauche...) et date de publication ;
3. liste de liens vers les formats disponibles (texte, HTML, PDF...) ;
4. liens vers la dernière version et la version précédente ;
5. liste des éditeurs, c'est-à-dire des responsables de la spécification.

La figure C–1 illustre l'en-tête de la recommandation DOM niveau 2 HTML.

Comme vous le voyez, la structure n'est pas toujours exactement identique à celle décrite plus haut, mais on retrouve très vite ses repères : ici, les formats sont simplement listés après la liste des éditeurs. Le lien vers les traductions, qui figure souvent après la partie introductive, se trouve ici en fin d'en-tête. Mis à part ceci, on reste dans le moule. Comparez avec l'en-tête de la recommandation pour HTML 4.01, qui correspond à l'ancienne façon de faire et reprend exactement notre liste (figure C–2).

**Figure C–1**
L'en-tête de la recommandation DOM niveau 2 HTML

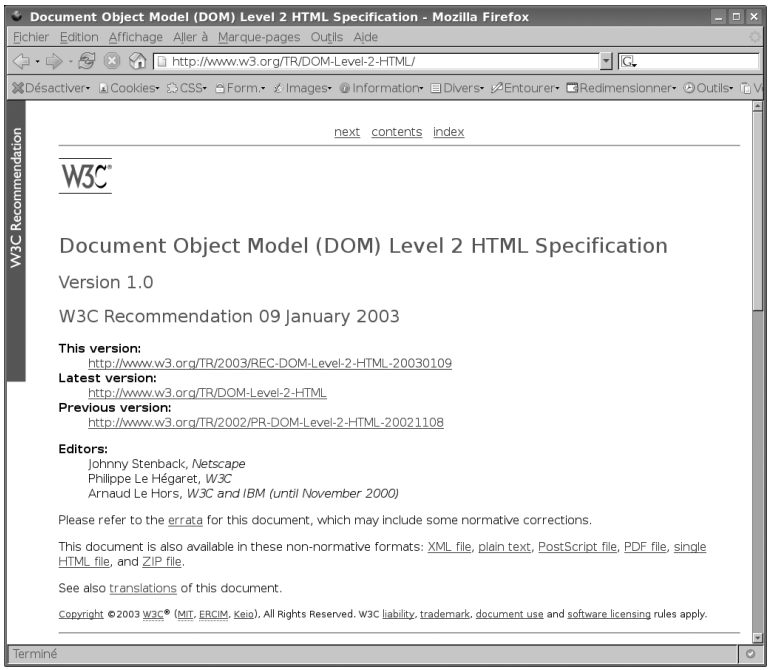
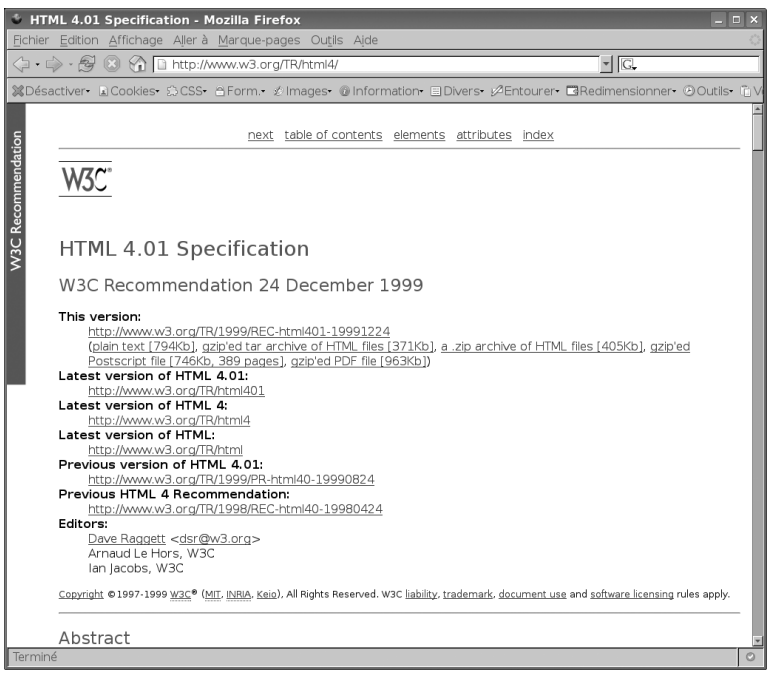



**Figure C–2**
L'en-tête de la recommandation HTML 4.01



Il a même une particularité, qui est de proposer les versions à jour et précédentes pour les variantes 4 et 4.01. Notez aussi que la liste des formats était bien moins lisible que la forme adoptée plus récemment.

On trouve ensuite la partie introductive, qui fournit :

1. l'*abstract*, qui décrit rapidement le rôle de la technologie spécifiée ;
2. le statut du document, qui contient toujours un texte plus ou moins pro forma sur le statut (recommandation, ébauche, etc.), l'état non normatif s'il s'agit d'une traduction, et fournit la liste des traductions connues (ou en tout cas un lien dessus, chercher le lien *translations* dans le corps du texte faute d'une section *Available languages*) ainsi qu'un lien vers les corrections ultérieures à la publication (errata) ;
3. la table des matières.

Suivant la spécification, on a alors plusieurs possibilités :

- Toute la spécification est sur la même page, le même document (cas de XHTML 1.0 ou XPath, par exemple).
- Seule la table des matières y figure et chaque section a une page dédiée (c'est le cas par exemple de HTML 4.01, XHTML 1.1 et CSS 2.1).
- La table des matières présentée ne référence que les parties incontournables (table des matières justement, copyright, annexes classiques, glossaire, références, index). Le cœur du sujet est exposé en une seule section (deux tout au plus), mais figure dans un sous-document (cas quasi systématique dans les spécifications du DOM, voir figure C-3).

On reconnaît vite la nature du document, rien qu'à la taille de l'ascenseur dans la barre de défilement verticale : s'il est très petit, il est probable qu'on ait toute la spécification sur une seule page !

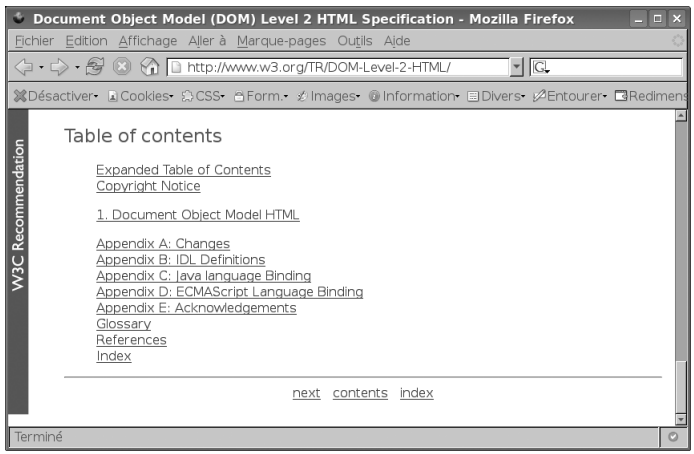



**Figure C–3**
Spécifications DOM : une table des matières courte et un sous-document

Le format premier d'une recommandation est le HTML. Les documents utilisent donc abondamment les hyperliens, ce qui rend leur consultation plus pratique. Toutefois, la simple taille des recommandations fait qu'on s'y perd facilement. Pour s'y retrouver, on a deux moyens. Si la spécification ne comporte que quelques pages (voire une seule), l'outil de recherche intégré au navigateur doit permettre de s'y retrouver rapidement. On l'appelle généralement avec Ctrl+F.

Pour des spécifications plus distribuées, comportant de nombreuses pages, il est bon de regarder si la recommandation fournit un index, qui figure alors généralement tout en bas de la table des matières. Chaque terme important (notamment tous les noms d'éléments, de propriétés, de méthodes, d'interfaces, etc.) fait l'objet d'un lien ; en cas de liens multiples, le premier porte le nom de l'élément et les suivants des numéros.

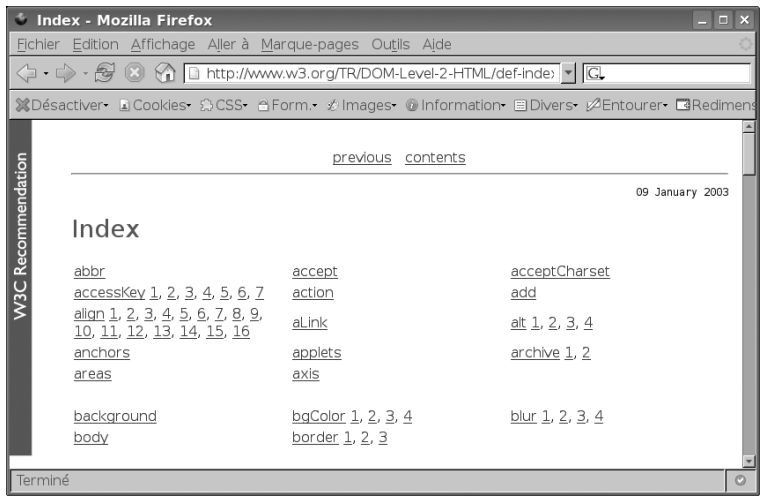



**Figure C–4**
Index d'une recommandation W3C

Astuce utile lorsqu'on cherche à déterminer si un aspect précis appartient à une version donnée ou à la précédente (ou simplement pour examiner rapidement en quoi la nouvelle version diffère de l'autre) : chaque recommandation dispose en annexe (et même dans les toutes premières annexes) d'une liste des changements. Suivant la taille de la recommandation, cette annexe est elle-même plus ou moins structurée. Ainsi, les changements pour CSS 2.1 (en réalité pour CSS 2.0 et 2.1) sont impressionnants (figure C–5).

En revanche, dans le DOM niveau 2 HTML, on est plus sobre : une simple entrée comme annexe A, nommée *Changes*. Il faut dire que le document détaillant les changements depuis DOM niveau 1 pour les éléments relatifs à HTML fait à peine défiler deux écrans.

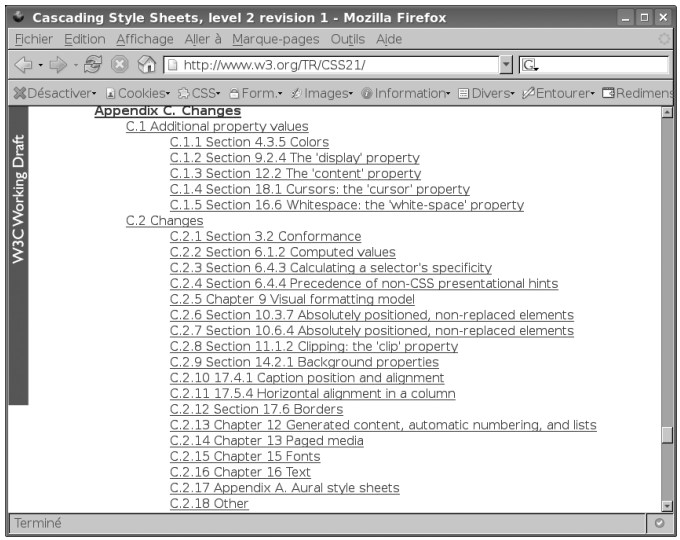



**Figure C–5**
Que de changements entre CSS 1 et CSS 2.1 !

## Recours à des syntaxes formelles

Suivant ses besoins, une recommandation W3C va s'appuyer sur des syntaxes formelles adaptées pour décrire des aspects techniques. Les principales syntaxes employées sont :

- DTD pour les éléments de balisage jusqu'à XHTML 1.0 Strict ;
- Schéma XML à partir de XHTML 1.1 ;
- IDL (*Interface Description Language*) pour les interfaces (essentiellement dans le cadre du DOM) ;
- EBNF (*Extended Backus-Naur Form*), Lex ou Yacc pour le reste.

En raison de leur fréquence, nous étudierons plus en détail DTD et les schémas XML dans les sections qui suivront.

IDL est très facile à lire, car elle ressemble à une déclaration de classe abstraite ou d'interface dans les principaux langages objets. C'est parfois aussi simple que dans le code suivant.

**Listing C-1 Déclaration IDL de l'interface HTMLElement (DOM niveau 2 HTML)**

```
interface HTMLElement : Element {
           attribute DOMString       id;
           attribute DOMString       title;
           attribute DOMString       lang;
           attribute DOMString       dir;
           attribute DOMString       className;
};
```

C'est parfois un peu plus compliqué, mais ça reste facilement lisible.

**Listing C-2 Déclaration IDL de l'interface HTMLSelectElement**

```
interface HTMLSelectElement : HTMLElement {
  readonly attribute DOMString       type;
           attribute long            selectedIndex;
           attribute DOMString       value;
  // Modified in DOM Level 2:
           attribute unsigned long   length;
           // raises(DOMException) on setting

  readonly attribute HTMLFormElement form;
  // Modified in DOM Level 2:
  readonly attribute HTMLOptionsCollection options;
           attribute boolean         disabled;
           attribute boolean         multiple;
           attribute DOMString       name;
           attribute long            size;
           attribute long            tabIndex;
  void               add(in HTMLElement element,
                         in HTMLElement before)
                             raises(DOMException);
  void               remove(in long index);
  void               blur();
  void               focus();
};
```

Quant à EBNF, il s'agit d'une des plus anciennes syntaxes textuelles de description de grammaire. Vous trouverez quelques explications sur cette syntaxe, plutôt simple, aux deux URL suivantes :

- http://developpeur.journaldunet.com/tutoriel/theo/050831-notation-bnf-ebnf.shtml
- http://fr.wikipedia.org/wiki/EBNF

Certaines spécifications utilisent des descriptions reposant plutôt sur les syntaxes Lex (ou Flex) et Yacc (ou Bison), bien connues des développeurs C. Ce n'est pas très éloigné d'EBNF. Voici un exemple tiré de la recommandation CSS 2.1, qui repose lui-même sur quelques définitions établies plus haut dans le document.

**Listing C-3 Définition Lex de syntaxe générale pour une feuille de styles CSS**

```
stylesheet  : [ CDO | CDC | S | statement ]*;
statement   : ruleset | at-rule;
at-rule     : ATKEYWORD S* any* [ block | ';' S* ];
block       : '{' S* [ any | block | ATKEYWORD S* | ';' S* ]* '}' S*;
ruleset     : selector? '{' S* declaration? [ ';' S* declaration? ]* '}' S*;
```

```
selector    : any+;
declaration : DELIM? property S* ':' S* value;
property    : IDENT;
value       : [ any | block | ATKEYWORD S* ]+;
any         : [ IDENT | NUMBER | PERCENTAGE | DIMENSION | STRING
              | DELIM | URI | HASH | UNICODE-RANGE | INCLUDES
              | DASHMATCH | FUNCTION S* any* ')'
              | '(' S* any* ')' | '[' S* any* ']' ] S*;
```

Il faut bien comprendre que, l'immense majorité du temps, les descriptions EBNF, Lex ou Yacc visent plus les développeurs de logiciels implémentant la technologie que ceux qui *utilisent* cette même technologie. Le texte environnant fournit généralement tous les détails nécessaires pour une utilisation courante.

## Les descriptions de propriétés CSS

La section « Déchiffrer une DTD », plus loin dans cette annexe, montrera comment lire et exploiter les fragments de DTD employés pour décrire les langages à balises, par exemple HTML. Avant d'en finir avec les recommandations W3C, je voudrais tout de même donner quelques informations supplémentaires sur deux formats très consultés dans la pratique : celui décrivant les propriétés CSS, et celui décrivant les propriétés et méthodes du DOM.

Commençons donc par les propriétés CSS. Par souci de référence, nous utiliserons la version originale, en anglais. Prenons par exemple la description de la propriété `white-space`, dans la section *Text*. La spécification utilise la représentation suivante pour décrire ladite propriété :

**'white-space'**

| | |
|---|---|
| *Value:* | normal \| pre \| nowrap \| pre-wrap \| pre-line \| inherit |
| *Initial:* | normal |
| *Applies to:* | all elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Computed values:* | as specified |

1 *Value* détaille la liste des valeurs possibles, séparées par des *pipes* (`|`), symbole fréquent pour indiquer une alternative. Les valeurs exprimées sont normatives et la casse est parfois significative. On a ici 6 valeurs possibles, dont l'incontournable `inherit` pour une propriété héritable.

2 *Initial* décrit la valeur par défaut, *none* s'il n'y en a aucune.
3 *Applies to* décrit les catégories d'éléments qui disposent de cette propriété. Ici elle s'applique à tous, mais des valeurs courantes sont *block-level elements*, *inline elements*, etc. Il s'agit des catégories déterminées par le modèle visuel CSS, évoqué à l'annexe B.
4 *Inherited* indique si la propriété est héritable ou pas ; une propriété héritable prend sa valeur par défaut dans son élément conteneur : c'est le principe de la cascade. Toutes les propriétés ne sont pas héritables (par exemple, `text-decoration` ne l'est pas).
5 *Percentages* est utile quand les valeurs possibles incluent une notation en pourcentage. C'est principalement le cas des tailles (de boîte ou de police de caractères). Cette valeur indique alors à quoi se réfèrent les pourcentages (par exemple, la largeur du bloc conteneur).
6 *Media* indique le ou les média CSS pour lesquels la propriété a du sens. Ici on est sur `visual`, ce qui couvre tous les média visuels : `screen`, `print`, `projection`, etc., tout en excluant par exemple les média comme `aural` (lecteurs d'écran).
7 *Computed values*, enfin, précise les ajustements à apporter à la valeur en fonction du contexte. C'est parfois simple et donc indiqué à la volée. Quand c'est plus complexe, on a généralement *as specified*, et les détails dans le texte qui suit. Par exemple, la valeur de la propriété `text-align` varie suivant celle de la propriété `white-space`.

## Les descriptions de propriétés et méthodes DOM

Autre format fréquemment consulté, les propriétés et méthodes DOM. On a déjà illustré le code IDL utilisé pour représenter une interface DOM complète, mais ces fragments de code sont suivis d'explications plus détaillées, bien entendu. Une interface DOM est toujours spécifiée en plusieurs morceaux :

1 le code IDL de l'interface ;
2 les descriptions des attributs éventuels ;
3 les descriptions des méthodes éventuelles.

Le code IDL fournit quelques informations précieuses. D'abord, au niveau de la déclaration de l'interface elle-même, si celle-ci étend une autre interface, on le voit immédiatement. Quelques exemples :

```
interface HTMLDocument : Document {
...
interface HTMLElement : Element {
...
interface HTMLImageElement : HTMLElement {
...
```

On voit clairement qu'un élément `img`, qui expose naturellement l'interface `HTMLImageElement`, expose donc aussi l'interface `HTMLElement`, et donc `Element`, et donc `Node`. Il fournit alors de nombreuses propriétés et méthodes !

Par ailleurs, le code IDL fournit une vue globale sur les types des propriétés et ceux des méthodes (types de retour, types des arguments). Quelques exemples là aussi, au travers du code IDL de `HTMLOptionsCollection`, reformaté :

```
// Introduced in DOM Level 2:
interface HTMLOptionsCollection {
  attribute unsigned long   length;
    // raises(DOMException) on setting
  Node item(in unsigned long index);
  Node namedItem(in DOMString name);
};
```

Que trouve-t-on ici ? D'abord, les attributs sont reconnaissables à ce qu'ils sont préfixés par `attribute`, et n'ont pas de parenthèses après leur nom. Le mot réservé `attribute` est parfois précédé de `readonly`, ce qui est très important : cela signifie qu'il est en lecture seule. Toute tentative d'écriture générera sans doute une exception.

Entre `attribute` et le nom de l'attribut, on a le type. IDL utilise des types primitifs issus du C, et des types objets qui sont soit d'autres interfaces du DOM, soit des types classiques définis par le DOM noyau.

Les méthodes n'ont pas `attribute` au début, mais un type de retour, un nom, et des parenthèses entre lesquelles on peut lister des paramètres, avec leur sens, leur type et leur nom. Toutes ces informations sont précieuses, mais le sens est généralement `in` et peut être ignoré. Il signifie simplement que la méthode utilise l'argument sans le modifier : c'est un paramètre local, passé par valeur, si vous préférez.

Voici une liste des principaux types utilisés dans l'IDL des spécifications DOM.

**Tableau C–1** Principaux types utilisés dans l'IDL du DOM

| Type | Description |
|---|---|
| `[unsigned] short` | Nombre entier sur 16 bits. Soit signé (-32 768 à 32 767), soit non signé (0 à 65 535). |
| `[unsigned] long` | Nombre entier sur 32 bits. Même remarque pour le signe. C'est le type numérique le plus courant. |
| `boolean` | Valeur booléenne : `false` ou `true`. |
| `void` | Aucune valeur. Type de retour des méthodes ne renvoyant rien. |
| `DOMString` | Chaîne de caractères Unicode (sur 16 bits). |
| `DOMTimeStamp` | Nombre entier positif sur 64 bits représentant un nombre de millisecondes depuis le début de l'ère (1er janvier 1970 00:00:00 GMT). De nombreux langages représentent ainsi les dates. |

**Tableau C–1** Principaux types utilisés dans l'IDL du DOM (suite)

| Type | Description |
|---|---|
| DOMException | Le type standard des exceptions levées par les méthodes. Contient simplement une propriété numérique entière nommée code, qui vaut l'une des constantes xxx_ERR décrites dans les recommandations du DOM. |
| Interface DOM (ex. Node) | Eh bien, l'interface en question... Reportez-vous à sa définition ! |

Les valeurs numériques signées sont assez rares : on utilise principalement `unsigned long`.

Après le code IDL, on trouve une section *Attributes* s'il y a des attributs, et une section *Methods*... s'il y a des méthodes.

Une section d'attribut reprend son nom, son type et son éventuelle contrainte de lecture seule. Un texte décrit l'attribut plus en détail, et si celui-ci a des contraintes pour son écriture, un paragraphe particulier précise l'information *Exception on setting* qui figurait déjà, normalement, en commentaire dans le code IDL.

Une section de méthode reprend son nom, puis décrit la méthode plus en détail avec un texte explicatif. Ce texte est obligatoirement suivi de trois sections : les paramètres, la valeur de retour et les exceptions potentielles. Les méthodes simplissimes se retrouvent donc avec trois embryons de sections. Ce qui donne par exemple :

**close**
Closes a document stream opened by `open()` and forces rendering.
**No Parameters**
**No Return Value**
**No Exceptions**

Ces sections fournissent souvent le petit détail qui explique pourquoi votre appel produit une erreur, ou ne donne rien, ou encore engendre un avertissement. Bien les lire est précieux.

Quand vous utilisez pour la première fois une interface DOM, lisez sa documentation complète. Ne vous contentez pas de quelques bouts qui semblent répondre à vos questions les plus immédiates. Prenez quelques minutes de plus pour lire l'ensemble, cela vous économisera souvent bien des heures de débogage par la suite.

# Déchiffrer une DTD

La DTD (*Document Type Definition*) constitue le premier format historique de grammaire pour les langages à balises. Une DTD est rédigée en SGML (*Standard Generalized Markup Language*), langage dont le HTML est en quelque sorte une réalisation.

Si vous rédigez correctement vos pages web, vous faites référence à une DTD tout au début de votre document, à l'aide d'une instruction `DOCTYPE`. Par exemple, la première ligne de votre page web dit normalement :

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

Comment ça, « non » ?! Mais c'est mal ! Vous ne faites pas du XHTML 1.0 Strict ? Vous n'avez pas honte ? Allez donc relire l'annexe A et faites pénitence.

Déclarer explicitement sa DTD de référence dans une page web a un double avantage. D'une part, cela permet d'utiliser correctement un validateur HTML : celui-ci pourra détecter votre grammaire au lieu d'essayer de la deviner, et vous assurera ainsi une vérification de grammaire appropriée. D'autre part, déclarer une DTD stricte (qu'elle soit HTML 4.01 ou XHTML 1.0) permet à votre navigateur de passer en mode « respect des standards », notamment en ce qui concerne les CSS. Pour plus de détails, cherchez donc l'expression *doctype switching* sur Google.

Mais revenons à nos DTD dans le cadre de spécifications. Une DTD constitue une spécification formelle pour un langage à balises (par exemple HTML 4.01). Elle n'explique pas le sens des balises ou des attributs, mais décrit quelles balises et quels attributs sont disponibles, où et quand.

Les spécifications HTML et apparentées ont pris l'habitude, au début de chaque section décrivant un élément, de présenter le fragment correspondant de la DTD. Observez par exemple la spécification de HTML 4.01 (sur laquelle repose XHTML 1.0, ne l'oubliez pas) pour l'élément `form`.

**Listing C-4 Le fragment de la DTD HTML 4.01 pour form**

```
<!ELEMENT FORM - - (%block;|SCRIPT)+ -(FORM) -- interactive form -->
<!ATTLIST FORM
  %attrs;                                  -- %coreattrs, %i18n, %events --
  action      %URI;          #REQUIRED -- server-side form handler --
  method      (GET|POST)     GET      -- HTTP method used to submit the form --
  enctype     %ContentType;  "application/x-www-form-urlencoded"
  accept      %ContentTypes; #IMPLIED -- list of MIME types for file upload --
  name        CDATA          #IMPLIED -- name of form for scripting --
  onsubmit    %Script;       #IMPLIED -- the form was submitted --
  onreset     %Script;       #IMPLIED -- the form was reset --
  accept-charset %Charsets;  #IMPLIED -- list of supported charsets --
  >
```

On a ici la description de l'élément `form` lui-même, avec son nom et la description de son contenu. Vous remarquez sans doute qu'ici, les noms d'éléments sont en majuscules, « à l'ancienne ». C'était la norme du temps de HTML (au siècle dernier, donc), mais depuis que XHTML est arrivé, on est sensible à la casse et les nouvelles normes officialisent la casse minuscule.

Les recommandations suivent d'ailleurs les pratiques de leur temps : même s'il est aujourd'hui communément admis que XHTML est incontournable, et qu'il faut donc respecter les règles de fermeture de balises, de valeurs d'attribut entre guillemets, et de balises en minuscules, la recommandation HTML foisonne toujours d'exemples « d'époque », pour ainsi dire, où beaucoup d'éléments ne sont pas fermés, sont en majuscules et ne protègent pas leurs valeurs d'attribut !

Détaillons rapidement la syntaxe de notre fragment. On a d'abord :

```
<!ELEMENT FORM - - (%block;|SCRIPT)+ -(FORM) -- interactive form -->
```

Le début, `<!ELEMENT`, indique qu'on définit un élément. Le nom suit : `FORM`.

Après les deux tirets séparés, on trouve une description du modèle de contenu : `(%block;|SCRIPT)+ -(FORM)`. On peut traduire ce modèle comme ceci : « soit le modèle *block*, soit l'élément `SCRIPT`, le tout autant de fois qu'on veut (mais au moins une fois), en revanche, pas d'élément `FORM` imbriqué ».

En effet, `%block` est ce qu'on appelle une référence d'entité, sur laquelle il est d'ailleurs possible de cliquer dans la recommandation. La définition de l'entité `%block` la décrit comme pouvant être réalisée par un certain nombre de balises connues, dont par exemple `P`, `H1`, `UL`, `PRE`, `DIV`, mais aussi `FORM`, ce qui amène justement l'exclusion explicite dans notre définition d'élément `FORM`.

La dernière partie, `-- interactive form --`, constitue un commentaire de fin de définition, qui nous signale que l'élément `FORM` permet de réaliser un formulaire utilisateur.

Passons maintenant à la liste des attributs pour l'élément `FORM`. Classiquement, elle figure dans la DTD juste après la définition de l'élément lui-même. J'ai retiré les alignements entre les champs pour améliorer la lisibilité individuelle des extraits. La liste des attributs débute par :

```
<!ATTLIST FORM
  %attrs; -- %coreattrs, %i18n, %events --
```

On déclare ici une définition d'attributs pour l'élément `FORM`. Les premiers attributs sont référencés par l'entité `%attrs`, dont le commentaire nous apprend qu'elle représente une combinaison des entités `%coreattrs`, `%i18n` et `%events`.

Ces entités regroupent les attributs dits noyau (`id`, `class`, `style` et `title`), ceux relatifs à la gestion des langues (`lang` et `dir`) et ceux relatifs aux événements JavaScript (par exemple `onclick`). Vous n'utiliserez bien entendu jamais ces derniers, puisque vous faites de l'*unobtrusive JavaScript*, n'est-ce pas ?

Que dit la suite ?

```
action %URI; #REQUIRED -- server-side form handler --
```

On a là un attribut `action`, dont le modèle de contenu est référencé par l'entité `%URI` (laquelle, idéalement, devrait décrire la syntaxe d'un URI, mais la syntaxe DTD étant pauvre, elle se contente, faute de mieux, de dire simplement « texte quelconque »...). On précise aussi que cet attribut est obligatoire (et je me suis rendu coupable d'infraction à cette règle dans certains exemples JavaScript de ce livre, je l'avoue...).

Deux autres exemples intéressants de définition d'attributs :

```
method (GET|POST) GET -- HTTP method used to… --
name    CDATA       #IMPLIED -- name of form for scripting --
```

On a ici une autre forme de modèle de contenu, qui dit en substance : « l'attribut `method` peut valoir `GET` ou `POST`, mais pas autre chose » (ce qui embête tant aujourd'hui les partisans d'une approche REST pour les API web). On voit aussi une valeur par défaut au lieu de `#REQUIRED` : si la méthode n'est pas précisée, elle vaudra `GET`.

La définition de l'attribut `name` indique qu'il s'agit d'un texte quelconque (type spécial `CDATA`, pour *character data*), et qu'il est optionnel sans valeur par défaut (`#IMPLIED`).

Voilà une introduction suffisante. Astuce intéressante : la spécification HTML 4.01 a jugé bon de fournir une présentation plutôt détaillée des syntaxes DTD utilisées, et cela directement dans la recommandation ! Voici l'adresse de la version française afin de vous faciliter le plus possible l'acquisition : http://www.la-grange.net/w3c/html4.01/intro/sgmltut.html#h-3.3. Si cela ne suffit pas, vous avez une introduction générique assez bien faite qui pourra peut-être mieux vous satisfaire sur http://www.w3schools.com/dtd/default.asp.

J'ai dit tout à l'heure que la DTD précisait quelles balises et quels attributs étaient disponibles, où et quand. Dit comme cela, on a l'impression que toute l'information peut être transmise par la DTD seule, et que le corps du texte de la recommandation n'est là que pour préciser le sens des éléments et des balises et ajouter quelques informations de contexte.

En réalité, une DTD n'est pas très précise. La syntaxe disponible ne permet pas de préciser de nombreux cas de figure courants, qu'il faut alors décrire dans le corps du texte. On ne peut pas formuler certaines règles simples, par exemple indiquer qu'un

élément A peut avoir au maximum 5 éléments fils B, sans même parler d'indiquer une fourchette allant de 2 à 5 éléments.

On ne peut pas non plus exprimer des exclusions entre les attributs, ou indiquer que si tel attribut est présent, tel autre doit l'être aussi. Également, on ne peut pas indiquer que les valeurs de tel attribut, toutes balises comprises, doivent être uniques dans le document (cas flagrant, par exemple en HTML : l'attribut `id`). Bref, les DTD sont limitées.

C'est pourquoi la communauté s'est tournée vers une autre syntaxe. Hélas, on a alors poussé jusqu'à l'extrême inverse : pour pouvoir tout décrire, on a créé une syntaxe si verbeuse, que la plupart des cas simples prennent de nombreuses lignes à représenter. Ce sont les schémas XML.

## Naviguer dans un schéma XML

Les schémas XML ont pris la relève des DTD pour décrire formellement les possibilités de combinaison et d'inclusion d'éléments et d'attributs dans les langages à balises. Et quand on sait combien le W3C s'est épris des technologies XML (plus de vingt à ce jour, de XML lui-même à XSLT, en passant par XPath, XML Query, SVG ou encore MathML...), on imagine facilement quel usage intensif est aujourd'hui fait des schémas XML.

Un schéma XML est lui-même un document XML. La syntaxe a été conçue pour couvrir tous les besoins imaginables. Et comme souvent dans de tels cas, elle aboutit à quelque chose de très épais pour les cas simples...

Prenons par exemple un fragment du schéma pour XHTML 1.1 Strict.

**Listing C-5 Le schéma XML de l'élément form en XHTML 1.1**

```
<xs:attributeGroup name="xhtml.form.attlist">
    <xs:attributeGroup ref="xhtml.Common.attrib"/>
    <xs:attribute name="action" type="xh11d:URI" use="required"/>
    <xs:attribute name="method" default="get">
        <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="get"/>
            <xs:enumeration value="post"/>
        </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="enctype" type="xh11d:ContentType"
                  ➥ default="application/x-www-form-urlencoded"/>
    <xs:attribute name="accept-charset" type="xh11d:Charsets"/>
    <xs:attribute name="accept" type="xh11d:ContentTypes"/>
```

```
</xs:attributeGroup>
<xs:group name="xhtml.form.content">
    <xs:sequence>
        <xs:choice maxOccurs="unbounded">
           <xs:group ref="xhtml.BlkNoForm.mix"/>
           <xs:element name="fieldset" type="xhtml.fieldset.type"/>
        </xs:choice>
    </xs:sequence>
</xs:group>
<xs:complexType name="xhtml.form.type">
    <xs:group ref="xhtml.form.content"/>
    <xs:attributeGroup ref="xhtml.form.attlist"/>
</xs:complexType>
```

À qui aura le front de s'insurger : « c'est illisible ! », « c'est n'importe quoi ! », ou encore « beaucoup de bruit pour rien ! », je répondrai d'un air affable « vous avez bien raison ».

Seulement voilà, le W3C s'est amouraché des schémas XML et il n'est pas le seul, loin de là ! L'univers J2EE par exemple a basculé des DTD vers les schémas autour de l'apparition des Servlets 2.4 et de JSP 2.0, il y a déjà longtemps.

Parmi les concepts clefs d'un schéma XML, il faut retenir qu'on définit le plus souvent séparément un type (une grappe d'éléments et de balises) et les noms des éléments qui reposent sur ce type (ce qui permet certes une factorisation fort agréable). Les structures simples sont déclarées à l'aide de balises `<xs:simpleType>`, et les types complexes (en réalité, presque tous) au moyen de `<xs:complexType>`.

La notion de *groupe*, qui permet d'exprimer une exclusivité d'éléments ou d'attributs, ou de décrire des dépendances, est représentée par `<xs:group>` pour les éléments et `<xs:attributeGroup>` pour les attributs. Les éléments et attributs sont désignés respectivement par `<xs:element>` et `<xs:attribute>`, les contraintes exprimées soit avec des attributs d'occurrence (`minOccurs` et `maxOccurs`), soit avec l'attribut `use`, ou encore avec des descriptions complexes `<xs:restriction>`. Le fragment présenté en listing C-5 est sympathique, parce qu'il illustre au moins un cas possible pour presque toutes ces syntaxes.

Il me faudrait un ouvrage entier pour vous présenter toutes les possibilités des schémas XML, mais si le sujet vous intéresse, vous trouverez une présentation bien faite et assez détaillée, pas à pas, sur http://www.w3schools.com/schema/schema_intro.asp.

Pour terminer, sachez qu'une syntaxe alternative fait de plus en plus d'aficionados, car tout en étant basée elle aussi sur XML, et bien que permettant d'exprimer tout ce qui est exprimable en schémas XML, elle est incomparablement plus simple et plus lisible. Il s'agit de Relax NG (nouvelle génération).

Relax NG est suffisamment populaire pour que la plupart des bibliothèques de traitement de grammaires XML la prennent aujourd'hui en charge (qu'on travaille en Java, C#, Delphi ou Ruby...). Toutefois, elle n'a pas encore su gagner le cœur des grands organismes de standardisation. Pour vous faire une idée, consacrez donc quelques minutes à ce didacticiel sur son site officiel, qui permet de bien cerner la question : http://relaxng.org/tutorial-20011203.html. En guise de *teaser*, voici l'équivalent Relax NG du listing C-5.

**Listing C-6 La spécification Relax NG de l'élément form en XHTML 1.1**

```
<define name="form">
  <element name="form">
    <ref name="form.attlist"/>
    <oneOrMore>
      <ref name="Block.class"/>
    </oneOrMore>
  </element>
</define>

<define name="form.attlist">
  <ref name="Common.attrib"/>
  <attribute name="action">
    <ref name="URI.datatype"/>
  </attribute>
  <optional>
    <attribute name="method">
      <choice>
        <value>get</value>
        <value>post</value>
      </choice>
    </attribute>
  </optional>
  <optional>
    <attribute name="enctype">
      <ref name="ContentType.datatype"/>
    </attribute>
  </optional>
</define
```

L'exemple étant bien choisi, on ne voit pas une immense différence de taille (28 lignes dans les deux cas), mais sentez-vous combien la seconde version est plus lisible que la première ?

# Parcourir une RFC

Une RFC (*Request For Comments*) est un standard Internet élaboré par l'IETF (*Internet Engineering Task Force*). Je dis bien « Internet » et non « Web », car alors que le Web a vu le jour en 1992 avec la première page HTML, sous la houlette de Tim Berners-Lee, Internet existe, lui, depuis 1969 (même s'il s'appelait alors ARPANET).

Ce sont aujourd'hui des centaines de protocoles et de formats qui font fonctionner ce réseau mondial. Le grand public connaît les plus visibles : HTTP, POP, IMAP, SMTP, SSL, TLS, FTP... Ceux qui prêtent attention aux détails de leurs clients de messagerie connaissent peut-être aussi MIME. Il faut sans doute être dans l'informatique pour en connaître d'autres, comme SNMP, NTP, SSH, Telnet, RTSP et bien d'autres. Sans parler des formats, de CSV à Atom.

Au total, ce sont plus de 4 600 documents standardisés qui ont déjà été émis, et avec environ 500 documents par an ces derniers temps, le phénomène ne fait qu'accélérer.

## Format général d'une RFC

Le format d'une RFC a toutefois perduré à travers les âges et trahit aujourd'hui ses origines très modestes, à une époque où les imprimantes matricielles 8 points étaient le dernier cri, et où la notion même de réseau d'ordinateurs n'était encore qu'une vision pleine d'espoir.

Le format principal d'une RFC est un fichier texte utilisant le jeu de caractères US-ASCII sur 7 bits (pas d'accents, pas de signes diacritiques, etc.). Le texte est formaté à 72 caractères de large. Certains termes ont un sens particulier, par exemple *must*, *should*, *can*, *must not*. Leur usage est défini par la RFC 2219. Une RFC est obligatoirement en anglais et le fichier porte l'extension `.txt`. On a encore bien des règles.

La RFC 2223 donne des détails sur le format ; une ébauche en plan depuis 2004 après 8 révisions, la 2223bis, met à jour certains points. On trouve même une RFC décrivant comment configurer MS Word pour produire un fichier RFC valide : la RFC 3285 !

Sachez par ailleurs qu'il existe trois sous-catégories de RFC : les standards de premier plan (STD), les pratiques recommandées (BCP, *Best Current Practice*) et les notes informatives (FYI, *For Your Information*). Dernier point : les RFC publiées le 1er avril sont systématiquement des canulars. Faites donc une recherche, ça vaut le détour.

Enfin, le format textuel d'une RFC rend difficile la navigation à l'intérieur du document, les références n'étant pas des hyperliens. Aussi, sachez qu'il existe un accès HTML aux documents, dont la navigation est par conséquent facilitée. Pour obtenir la version HTML d'une RFC, c'est facile : prenez d'abord l'URL officielle de la version principale, par exemple : http://www.ietf.org/rfc/rfc4287.txt.

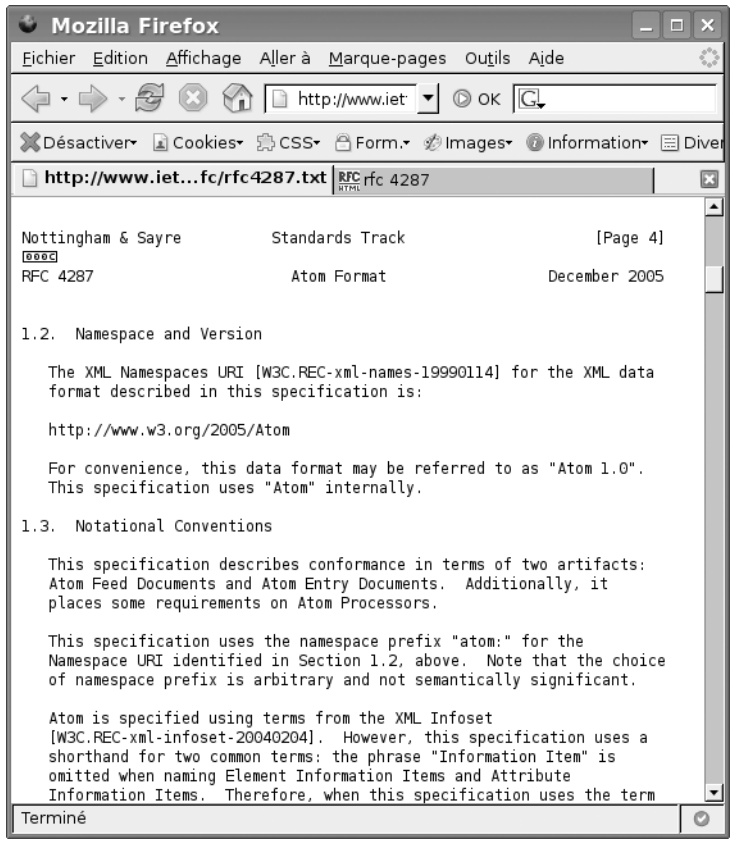



**Figure C–6**
L'aspect texte brut de la RFC du format Atom

Remplacez ensuite le « `www.` » par « `tools.` », le chemin `/rfc/` par `/html/` et retirez l'extension. Vous obtenez ceci : http://tools.ietf.org/html/rfc4287.

Et voilà une version HTML de votre RFC, avec des liens automatiques pour les numéros de page, les URL, les références et les renvois entre sections. Par ailleurs, certaines portions sont mises en gras (titres) ou en italique, et les en-têtes et pieds de page sont grisés. Jugez plutôt (figure C–7).

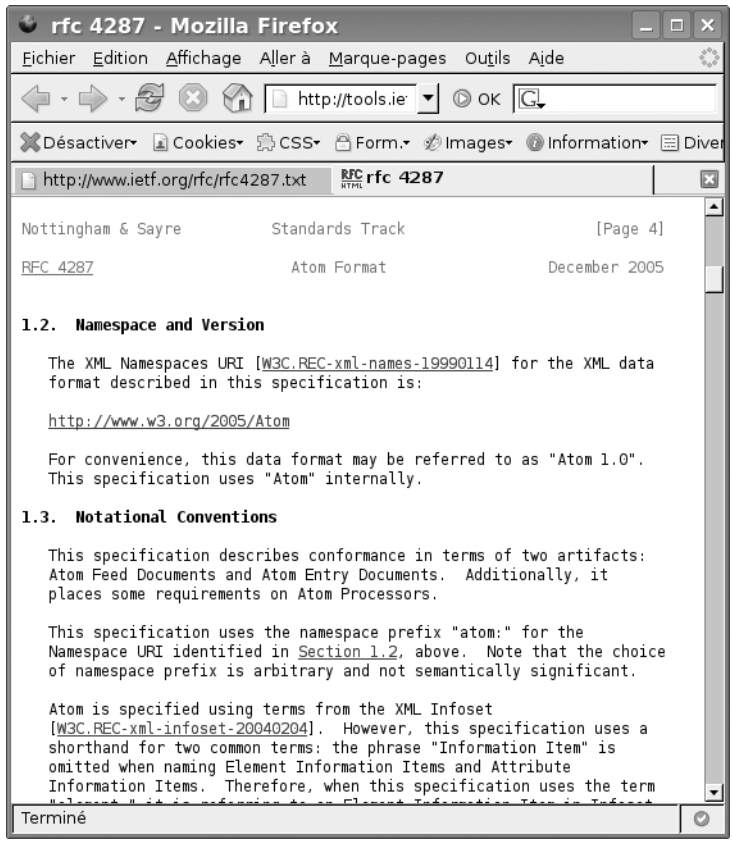



**Figure C–7**
La RFC passée à la moulinette HTML

## Structure générale d'une RFC

Une RFC commence toujours par son en-tête déclaratif, repris en en-tête de chaque page. Voici un exemple, raccourci en largeur pour tenir sur cette page :

```
Network Working Group                              M. Nottingham, Ed.
Request for Comments: 4287                              R. Sayre, Ed.
Category: Standards Track                               December 2005
```

On a sur la gauche le nom du groupe de travail (l'IETF en compte un certain nombre), le statut (RFC) et le numéro du document. Les RFC sont en effet le plus souvent référencées par leur numéro. La catégorie indique plus précisément le statut. Ici, le terme *Standards Track* confirme que le document a valeur de standard.

Sur la droite, on trouve la liste des auteurs principaux (le « `Ed.` » signifie *Editor*) et la date de publication du document, avec le mois et l'année.

L'en-tête des pages suivantes reprendra toutes ces informations ainsi que le numéro de page et le titre du document :

```
Nottingham & Sayre        Standards Track                [Page 1]

RFC 4287                    Atom Format             December 2005
```

Une RFC comporte obligatoirement une introduction, dont les premiers paragraphes doivent décrire avec concision le contexte et l'objectif du standard. L'IETF est ici généralement plus efficace que le W3C dans ses résumés...

L'introduction comprend obligatoirement, généralement vers la fin, une section du type *Notational Conventions*, qui détaille les notations propres au document, et précise systématiquement le sens formel de termes comme *must*, *should*, *cannot*, etc. en faisant une référence à leur définition exacte dans la RFC 2119.

Une RFC se clôt généralement par une liste de références normatives (autres standards) ou informatives, une liste des contributeurs au standard (auteurs n'ayant pas un statut « principal »), et souvent des versions consolidées de grammaires formelles ou autres informations techniques fragmentées dans le corps du document.

Il n'est pas rare de voir, dans les derniers chapitres d'une RFC, un *IANA Considerations*, si le standard entraîne le dépôt de nouveaux types MIME ou réquisitionne certains numéros de ports réseau, ainsi qu'un *Security Considerations*, qui détaille toute faille de sécurité potentielle envisagée par les auteurs.

Enfin, les RFC font une utilisation intensive de la syntaxe EBNF, évoquée plus haut. Prenez par exemple la RFC 2616 (http://tools.ietf.org/html/rfc2616), celle qui décrit HTTP/1.1. La syntaxe des en-têtes de requête et de réponse est décrite en EBNF. En voici un extrait dans le listing suivant.

**Listing C-7 Syntaxe EBNF de l'en-tête de requête Accept en HTTP/1.1**

```
Accept         = "Accept" ":"
                 #( media-range [ accept-params ] )

media-range    = ( "*/*"
                 | ( type "/" "*" )
                 | ( type "/" subtype )
                 ) *( ";" parameter )
accept-params = ";" "q" "=" qvalue *( accept-extension )
accept-extension = ";" token [ "=" ( token | quoted-string ) ]
```

Comme souvent, l'ensemble de la syntaxe est expliquée clairement en début de RFC, dans la section *Notational Conventions*, pour ne pas perdre les lecteurs.

Notez que la RFC citée en exemple ci-dessus, celle du format de flux Atom, innove en utilisant des schémas Relax NG pour décrire son balisage.

## Vos spécifications phares

En tant que développeur web, voici les principales spécifications qui devraient vous intéresser :

**HTML 4.01 (éléments et attributs autorisés)**

- Référence : http://www.w3.org/TR/html4/
- Version française : http://www.la-grange.net/w3c/html4.01/cover.html

**XHTML 1.0 (repose sur HTML 4.01)**

- Référence : http://www.w3.org/TR/xhtml1/
- Version française : http://www.la-grange.net/w3c/xhtml1/

**CSS 2.1**

- Référence : http://www.w3.org/TR/CSS21/
- Version française (2.0 !) : http://www.yoyodesign.org/doc/w3c/css2/cover.html

Attention, la version 2.1 a beaucoup modifié la version 2.0...

**JavaScript 1.5**

- DevMo : http://developer.mozilla.org/fr/docs/JavaScript
- ECMA : http://www.ecma-international.org/publications/standards/Ecma-262.htm

**DOM niveau 2**

- Noyau : http://www.w3.org/TR/DOM-Level-2-Core/
- HTML : http://www.w3.org/TR/DOM-Level-2-HTML/
- Événements : http://www.w3.org/TR/DOM-Level-2-Events/
- Style : http://www.w3.org/TR/DOM-Level-2-Style/

# D

# Développer avec son navigateur web

C'est à ses outils qu'on reconnaît le bon artisan... Il est effarant de constater combien, encore aujourd'hui, les développeurs web persistent à sous-utiliser leurs navigateurs dans leurs développements. Le navigateur est relégué au rang de visualiseur, ce qui est aberrant !

Il ne s'agit pas seulement de savoir configurer le cache pour être certain de toujours utiliser la dernière version d'une ressource qui change fréquemment. Les principaux navigateurs savent explorer les méandres internes de la page, son DOM, ses CSS, son accessibilité... Ils fournissent des pelletées d'outils informatifs, d'analyseurs, et même des débogueurs JavaScript !

La question délicate du débogage JavaScript fait l'objet d'un chapitre dédié : le chapitre 5. Pour tout le reste (DOM, CSS, etc.), il y a cette annexe.

Il ne tient qu'à vous d'ajouter une bonne dose de confort et d'efficacité, en somme, de *productivité*, à votre méthodologie de développement. Si j'étais vous, je lirais attentivement cette annexe *avant tout le reste*.

# Le cache peut être votre pire ennemi

Surtout alors que vous suivez les exemples de ce livre. Le cache, c'est très pratique : ça nous évite de passer notre vie à recharger tout le temps des images, feuilles de style, etc. Mais en développement, cela peut poser problème. On change fréquemment la CSS, le JavaScript, les feuilles XSLT...

Chaque navigateur a sa façon bien à lui de gérer ses caches, qu'il s'agisse du disque ou de la mémoire. Certains seront plus réactifs aux changements de ressources accédées directement qu'à ceux affectant les données obtenues par Ajax ; d'autres rechargeront plus volontiers du JavaScript que des CSS ; etc.

## Le rafraîchissement strict

La plupart des navigateurs fournissent un mécanisme clavier pour effectuer un **rafraîchissement strict**, c'est-à-dire un rechargement effectif de l'intégralité des ressources utilisées par la page :

- **Mozilla, Camino, Firefox, MSIE** : au lieu de presser simplement *F5* ou de cliquer sur le bouton idoine, maintenez *Ctrl* ou *Cmd* enfoncée pendant ce temps. Hors MSIE, vous pouvez aussi utiliser *Maj* avec le bouton de rechargement ou *F5*.
- **Safari** : utilisez *Cmd+Maj+R* au lieu de *Cmd+R*, ou maintenez *Cmd* enfoncée en pressant le bouton de rafraîchissement.
- **Opera** : il n'y a pas de mécanisme de rafraîchissement strict ! C'est ahurissant, mais c'est comme ça. En théorie, Opera ignore le cache à chaque rafraîchissement explicite avec *F5* ou son bouton de rechargement. Dans la pratique, sur les exemples de ce livre, j'ai dû configurer le cache. On peut donc décider de faire de même, pour qu'il vérifie soigneusement les nouvelles versions des ressources (voir un peu plus bas), ou vider complètement le cache (voir ci-dessous).
- **Konqueror** : ignore le cache à chaque rechargement explicite avec *F5* ou son bouton de rechargement.

## Vider le cache

Il est parfois nécessaire de vider complètement le cache, ou en tout cas les parties du cache associées à la page en cours. Voici comment faire.

- **Firefox** : allez dans les options (*Outils > Options* sous Windows, *Édition > Préférences* sous Linux, *Firefox > Préférences* sous Mac OS X) et choisissez la catégorie *Vie privée* puis l'onglet *Cache*. Cliquez sur le bouton *Vider le cache*. Depuis sa version 1.5, Firefox permet également de supprimer instantanément tout ou partie de votre état « privé » en enfonçant *Ctrl+Maj+Suppr* (*Cmd+Maj+Ret.Arr.* Sur Mac OS X) ou

en allant dans *Outils* > *Effacer mes traces*. Attention, par défaut cela sélectionne bien plus que le cache ! Vous pouvez le configurer depuis la catégorie *Vie privée* des options, avec le bouton *Paramètres*.

- **Mozilla** : allez dans *Édition* > *Préférences* > *Cache* et choisissez *Vider le cache*.
- **MSIE 6** : allez dans *Outils* > *Options Internet* > *Général* > *Fichiers Internet temporaires*. Le bouton *Supprimer les fichiers...* permet de vider le cache (pensez à cocher la case *Supprimer tout le contenu hors-ligne*). Validez.
- **MSIE 7 et 8** : allez dans *Outils* > *Options Internet* > *Général* et dans la section *Historique de navigation* activez le bouton *Supprimer…* Cliquez alors sur le bouton *Supprimer les fichiers…* et validez.
- **Safari** : dans le menu *Safari*, choisissez *Vider le cache...*, ou pressez *Cmd+Option+E*.
- **Opera** : allez dans *Outils* > *Préférences* > *Avancé* > *Historique* et choisissez le bouton *Vider maintenant*.
- **Konqueror** : allez dans *Configuration* > *Configurer Konqueror*. Dans la liste de gauche, faites défiler pour choisir la catégorie *Cache*. Cliquez sur le bouton *Vider le cache*.

## Configurer le cache

Pour éviter de trop peiner avec le cache, il est important de le configurer correctement. En développement, le mieux est de lui demander de vérifier à chaque requête (à l'aide d'une requête HTTP `HEAD`), pour chaque ressource, si cette dernière a changé depuis sa dernière version. On peut parfois le désactiver complètement, mais c'est moins utile.

Voici les modes opératoires :

- **Mozilla, Camino, Firefox** : c'est une mauvaise idée, mais vous pouvez demander au navigateur d'allouer 0 Ko au cache dans les options. Toutefois, il ne s'agit que du cache disque : un cache mémoire existe tout de même. Il est désactivable via les options « expert » de `about:config`. Là aussi, mauvaise idée...
- **MSIE 6** : allez dans *Outils* > *Options Internet* > *Général* > *Fichiers Internet temporaires*. Cliquez sur *Paramètres...* Choisissez l'option *À chaque visite de la page*. Ainsi, MSIE vérifiera à chaque fois, et récupérera toute nouvelle version.
- **MSIE 7 et 8** : allez dans *Outils* > *Options Internet* > *Général* et dans la section *Historique de navigation* activez le bouton *Paramètres*. Choisissez l'option *À chaque visite de la page*, comme sur les versions 6 et 7.
- **Safari** : à n'utiliser qu'à vos risques et périls... Fermez totalement Safari, ouvrez un terminal (outil *Terminal* dans *Applications* > *Utilitaires*), et tapez deux lignes : d'abord `rm -fr ~/Library/Caches/Safari`, puis `touch ~/Library/Caches/Safari`. Quittez le terminal.

- **Opera** : allez dans *Outils > Préférences > Avancé > Historique* et configurez les listes déroulantes *Vérifier les documents* et *Vérifier les images*, en les définissant à *Toujours*. Vous pouvez aussi choisir de désactiver le cache en définissant *Cache mémoire* et *Cache disque* à *Off*, mais ce n'est pas une très bonne idée...
- **Konqueror** : allez dans *Configuration > Configurer Konqueror*. Dans la liste de gauche, faites défiler pour choisir la catégorie *Cache*. Si la case *Utiliser le cache* est cochée, assurez-vous que l'option *Assurer la synchronisation du cache* est sélectionnée. Pour désactiver complètement le cache, décochez simplement la case. Mais sur Konqueror, c'est *vraiment* une mauvaise idée : sa gestion du cache est bien adaptée au développement.

# Firefox, favori du développeur grâce aux extensions

D'entrée de jeu, Firefox fournit une console JavaScript plutôt correcte (*Outils > Console JavaScript*), mais pas folichonne. Nous l'avons vue au chapitre 5.

Firefox est sans doute néanmoins le meilleur navigateur pour développer des applications web, tant l'univers de ses extensions est riche d'outils fabuleux ! Je n'en cite que deux qui me semblent incontournables : la **Web Developer Toolbar** de Chris Pederick, et **Firebug** de Joe Hewitt.

## Firebug

Nous avons déjà vu les possibilités JavaScript de Firebug en détail au chapitre 5. Petite extension légère, il fournit néanmoins une console JavaScript efficace, un objet `console` utilisable dans vos scripts, un pisteur de requêtes Ajax et un débogueur JavaScript amplement suffisant. Il nous reste à examiner un superbe inspecteur multivue :

- code source, très utile ;
- DOM, plus classique ;
- layout, parfois irremplaçable...

Ces inspecteurs sont de petits bijoux.

L'inspecteur de code source de Firebug est accessible via l'onglet *HTML* de la zone principale. Le code source qui apparaît représente en réalité le DOM de la page ; il est mis à jour en temps réel ! On peut par ailleurs déplier/replier chaque élément. Sélectionner un élément à gauche permet par ailleurs de visualiser à droite le détail des styles qui lui sont appliqués, regroupés par règle CSS, et mettant en évidence les

propriétés qui ont été « écrasées » par d'autres au fil de la « cascade » (les propriétés ainsi ignorées apparaissent barrées).

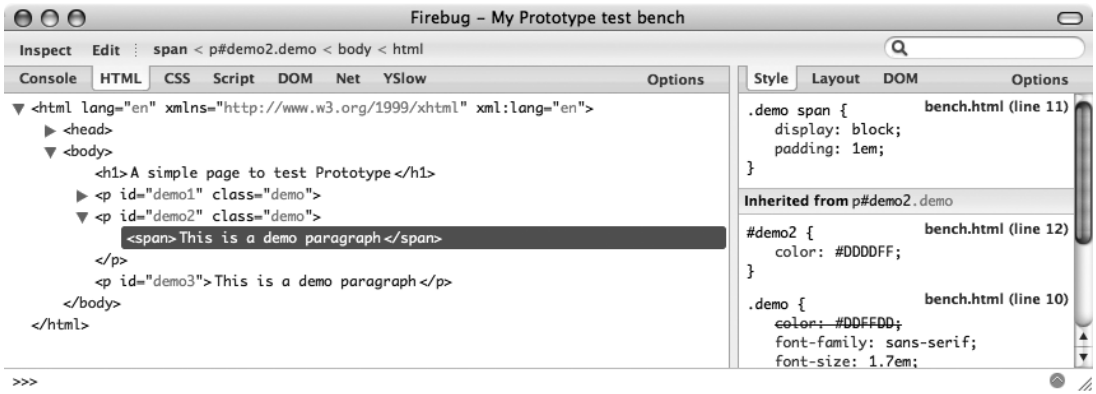



**Figure D–1** L'inspecteur de code source de Firebug, en mode « style »

Lorsqu'on survole un élément dans le listing, celui-ci est mis en évidence dans la page elle-même, ainsi que ses marges (en jaune) et ses espacements internes (*padding*, en violet). C'est extrêmement utile !

Sur la droite, outre l'onglet *Style*, on dispose aussi d'un onglet *Layout* qui se révèle indispensable dans bien des cas : il indique la position et l'ensemble des dimensions de l'élément sélectionné dans l'onglet *HTML*, en pixels. Cela permet de vérifier le bon résultat des règles de positionnement qu'on a mises en place, et en cas de doute, un tour par l'onglet *Style* pour vérifier les propriétés `margin`, `padding`, `width`, `height`, `left`, `top`, etc. appliquées permet de découvrir l'éventuel problème.

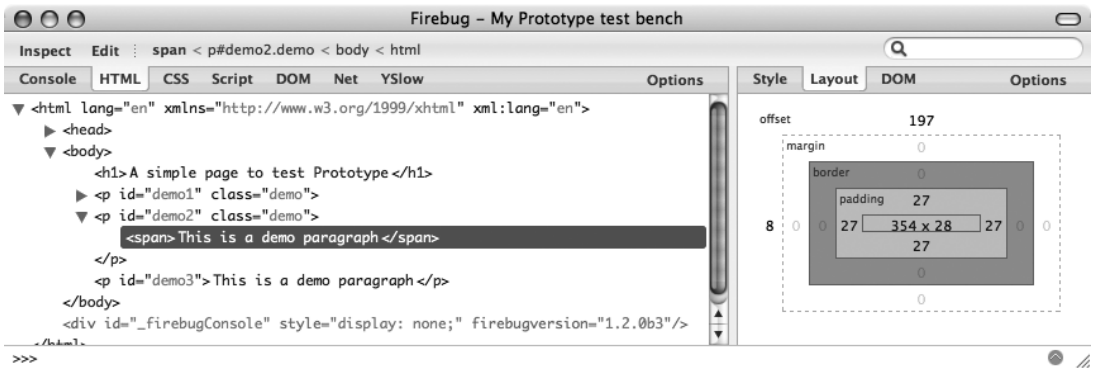



**Figure D–2** L'onglet Layout de Firebug se révèle vite incontournable !

Enfin, Firebug propose deux façons d'accéder au DOM :

- À partir de la racine (l'objet `window`), donc des variables dites « globales » ; on y retrouvera par conséquent tous les objets mis en place par les bibliothèques JavaScript chargées (par exemple Prototype), plus les objets « standards » (`window`, `navigator`, `screen`…), mais aussi tous les objets racine plus spécifiques au navigateur (par exemple, sur Firefox, on trouvera `crypto`, `globalStorage`…).
  Ce mode correspond à l'onglet *DOM* de la partie gauche, et n'est que rarement utilisé.
- Pour ce qui concerne l'élément sélectionné dans l'onglet *HTML*, ce qui est évidemment beaucoup plus utile, on y retrouvera les propriétés et méthodes de l'élément, chaque propriété pouvant être « déroulée » si c'est un objet (on liste alors le contenu de cet objet, et ainsi de suite sans limite de profondeur). On y accède par l'onglet *DOM* de la partie droite, et c'est le mode le plus employé.

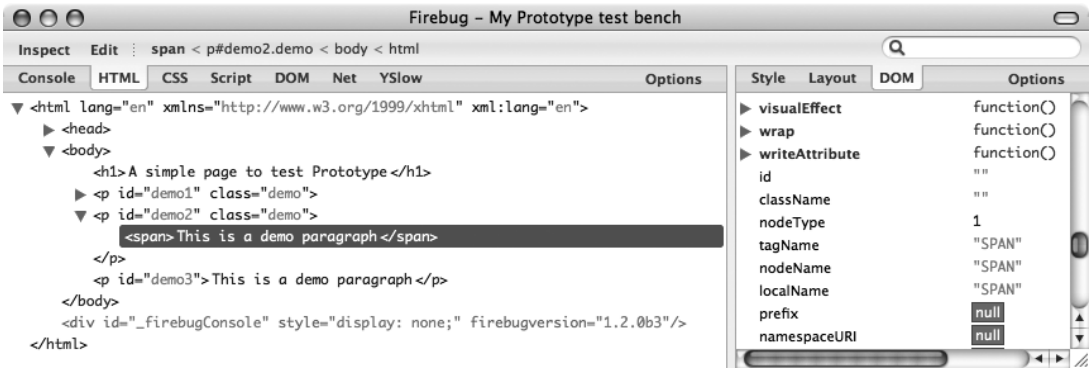



**Figure D–3** L'inspecteur DOM pour l'élément sélectionné dans Firebug

Un dernier conseil : pensez à explorer les menus *Options* qui figurent généralement à droite des barres d'onglets. Il y en a pour de nombreux onglets, avec des options dédiées à chaque fois. On y trouve beaucoup de réglages utiles suivant la circonstance…

## Web Developer Toolbar

La Web Developer Toolbar de Chris Pederick est devenue une véritable légende. Une fois installée grâce au système d'extensions de Firefox (qui s'est énormément amélioré avec Firefox 3), on obtient la barre d'outils suivante (en français si on l'a téléchargée chez JoliClic[1]) :

1. http://joliclic.free.fr/mozilla/webdeveloper/

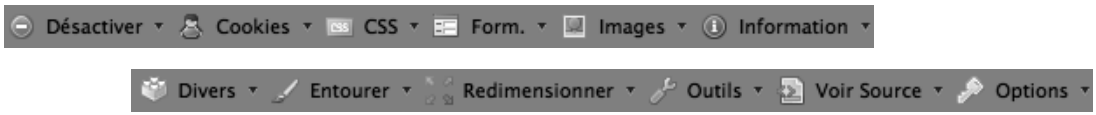



**Figure D–4** La Web Developer Toolbar avec icônes et libellés

Parmi la pléthore de fonctionnalités qu'elle propose, je recommande notamment celles-ci, dont je me sers pratiquement au quotidien :

- *Désactiver > Désactiver le cache*. Indispensable lorsque vous souhaitez être absolument sûr(e) que les prochains rafraîchissements, ou les requêtes Ajax, iront directement à la couche serveur.
- *Désactiver > Désactiver JavaScript > Tout le JavaScript*. Très utile lorsque vous suivez les préceptes d'amélioration progressive pour le développement de vos pages, et devez donc vérifier qu'elles fonctionnent convenablement sans JavaScript. N'oubliez pas de rafraîchir ensuite la page pour garantir qu'aucun code JavaScript n'aura été exécuté au chargement.
- *Cookies > Désactiver les cookies > Tous les cookies*. Utile pour vérifier le comportement des sessions et attributions de cookies par le serveur. Ce menu abrite aussi plusieurs options pour nettoyer les cookies par type (session, domaine, chemin…) et consulter/modifier la liste des cookies actifs pour la page.
- *CSS > Désactiver les styles css > Tous les styles*. Une page balisée sémantiquement doit être « lisible » intuitivement sans aucune feuille de style. Ce mode, accessible avec *Ctrl+Maj+S* sur Windows et Linux, *Cmd+Maj+S* sur Mac OS X, se révèle vite indispensable pour « auditer » le balisage d'une page, par exemple dans un contexte de référencement naturel.
- *Images > Remplacer les images par l'attribut alt* (tout en bas). Une page accessible, c'est aussi des attributs `alt=` correctement renseignés pour toutes les images non décoratives (images dites « de contenu »). À l'inverse, des images purement décoratives ont en général un attribut `alt=` défini mais vide. Cette fonction permet de se rendre compte tout de suite de la pertinence des attributs `alt=` en place, et de ceux qui manquent manifestement.
- *Information > Afficher les AccessKeys*. Toujours dans un contexte d'accessibilité, on tente de définir au mieux les raccourcis clavier pour des zones critiques de la page (recherche, navigation, contenu principal, plan du site, bascule de langue ou de taille de texte…). Il n'existe aujourd'hui aucun standard formel sur la question mais des conventions un peu éparses (par exemple http://clagnut.com/blog/193/, qui documentent plusieurs schémas dont le standard gouvernemental au Royaume-Uni, l'un des plus complets sur le sujet). Certains navigateurs affichent automatiquement les *accesskeys* sur enfoncement de la touche commande de raccourci (*Alt* sur Windows,

*Alt* ou *Ctrl* sur Linux, *Cmd* sur Mac OS X), mais Firefox non : cette option est donc bien utile…

- *Information > Afficher les index de tabulation*. Une page accessible, c'est aussi un ordre de tabulation intelligent, parfois différent de l'ordre naturel des éléments dans le code source du document. C'est particulièrement vrai pour un formulaire, mais toute page gagne à avoir un ordre « utile » de tabulation, par exemple avec le contenu principal en premier, la navigation principale en deuxième et la navigation générique en dernier. Je rappelle que l'attribut `tabindex=` est autorisé sur tout élément visuel (notamment les liens), et pas seulement sur les champs de formulaire.
- *Information > Afficher la profondeur des tableaux*. Lorsqu'on audite la qualité de balisage d'une page, par exemple dans une optique de référencement naturel, la profondeur d'imbrication des tableaux éventuellement utilisés pour la mise en page (horreur !) est d'une importance significative. Au-delà de 3 ou 4 niveaux, le « poids » du contenu est tellement minoré qu'il en devient presque insignifiant. À l'heure où des « recettes » fiables et sans balisage superflu permettent de se passer de tableaux pour pratiquement tous les cas de figure (et de gagner ainsi en élasticité), des mises en page trop lourdes en tableaux imbriqués doivent être détectées et remaniées.
- *Information > Plan du document*. Cette commande très pratique permet d'obtenir, dans un onglet à part, la structure des titres du document, basée sur les balises de titre `h1` à `h6`. On peut ainsi voir en un clin d'œil si la hiérarchie de titres de la page est cohérente ou non. La cohérence de cette hiérarchie influe fortement sur la pertinence de l'indexation par les moteurs de recherche et, dans une optique d'accesibilité, sur la navigation alternative. Voici les deux principales erreurs de cohérence :
  - hiérarchie ne démarrant pas par le niveau `h1` ;
  - niveaux « sautés » (par exemple des `h4` sous des `h1`).
- *Entourer > Entourer les tableaux > Cellules de tableaux*. Afin de déterminer si une page ayant recours aux tableaux pour tout ou partie de sa mise en page en fait un usage raisonnable ou non, cette commande affiche, avec des codes couleur par niveau d'imbrication, l'ensemble des cellules de tableau du document. On voit immédiatement si celles-ci ne sont utilisées que sporadiquement pour implémenter rapidement la disposition principale de la page (utilisation acceptable) ou si les tableaux sont utilisés à foison et à toutes les sauces (pratique obsolète et dommageable pour le contenu).
- *Redimensionner > 1024 × 768* (nécessite une personnalisation). Inexplicablement, Web Developer Toolbar n'a pas de tailles prédéfinies pour cette commande ! Il faut aller dans la commande Modifier les dimensions… pour définir les combinaisons qui nous intéressent. La taille minimale acceptée aujourd'hui est le 1024 × 768.

- *Options* > *Fonctionnalités persistantes*. Hormis la désactivation du JavaScript, du cache et des cookies, l'ensemble des fonctionnalités de la Web Developer Toolbar « disparaissent » au rafraîchissement ou en naviguant sur une autre page. Si vous souhaitez maintenir une même configuration de visualisation tout au long de ces opérations, cochez cette option.

Une fois qu'on est habitué(e) à la barre, on a souvent tendance à retirer les libellés (champ *Afficher la barre d'outils avec…* de la section *Général* des *Options*).

**Figure D–5**
La Web Developer Toolbar en mode « icônes sans libellés »

Il me faudrait un chapitre entier pour tout décrire, allez plutôt regarder son site web, téléchargez-la si ce n'est déjà fait, et émerveillez-vous !

Vous trouverez pour Firefox énormément d'extensions et d'outils complémentaires, qui correspondront peut-être davantage à vos besoins. Je cite rapidement la XML Developer Toolbar, GreaseMonkey, Platypus, Aardvark...

Les URL correspondantes :

- Web Developer Toolbar : http://chrispederick.com/work/webdeveloper/
  - En français : http://joliclic.free.fr/mozilla/webdeveloper/
- Firebug : http://www.joehewitt.com/software/firebug/
- XML Developer Toolbar : https://addons.mozilla.org/firefox/2897/
- GreaseMonkey : http://greasemonkey.mozdev.org/
- Platypus : http://platypus.mozdev.org/
- Aardvark : http://karmatics.com/aardvark/

# Les trésors du menu Développement caché dans Safari

Safari 3 dispose d'un menu *Développement*, caché par défaut, qui regorge d'options intéressantes pour le développeur web. Pour l'activer, il suffit de réaliser les manipulations suivantes :

1. Fermez totalement Safari.
2. Ouvrez un terminal (*Applications* > *Utilitaires* > *Terminal*).
3. Tapez `defaults write com.apple.Safari IncludeDebugMenu 1`.
4. Vous pouvez relancer Safari.

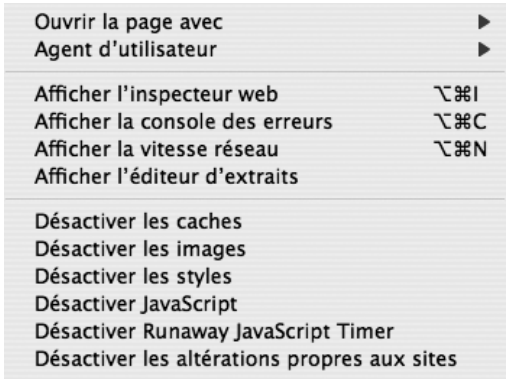



**Figure D–6**
Le menu Développement de Safari 3

Dans sa version 2, Safari disposait d'un menu *Debug*, non traduit, qui contenait une pléthore d'options parfois résolument « internes » (du genre *Use ATSU For All Text*, *Use Threaded Image Decoding* ou encore l'incongru *Populate History*). On a désormais quelque chose de plus orienté développeur web. Et par-dessus le marché, ils l'ont traduit !

Parmi les différentes options, la plus utile, et de loin, est *Afficher l'inspecteur web* (*Cmd+Alt+I*). Cet inspecteur, qui peut sembler au premier abord un peu limité, deviendra incontournable en un ou deux clics.

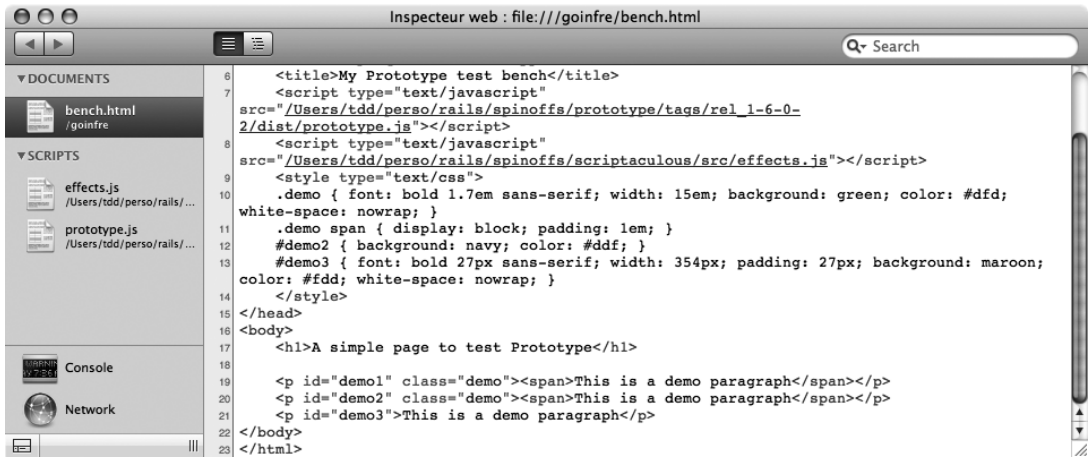



**Figure D–7** L'inspecteur web de Safari 3 en mode basique

L'inspecteur web s'affiche par défaut dans une fenêtre à part ; c'est mon mode préféré car en mode « panneau » (le petit bouton en bas à gauche bascule entre les modes fenêtre et panneau), l'inspecteur ne permet pas de redimensionner sa hauteur explicitement, ce qui peut le rendre minuscule…

Autre point important : cet inspecteur peut basculer entre un mode « code source » (celui de la figure D-7) et un mode « arborescence », bien plus utile et similaire à celui de l'onglet *HTML* de Firebug. On bascule de l'un à l'autre à l'aide des deux boutons centraux de la barre supérieure.

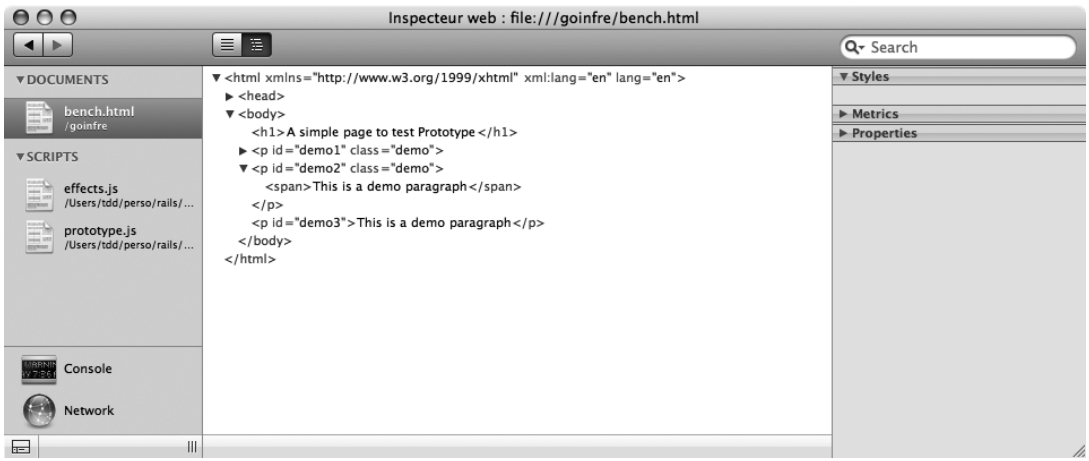



**Figure D–8** L'inspecteur web en mode arborescent.

C'est là que se révèle toute la richesse de l'inspecteur : en sélectionnant un élément dans l'arborescence, on dispose des fonctions suivantes :

- Mise en évidence de l'élément dans la fenêtre de navigation par ombrage de tout le reste de la page.
- Affichage des styles actifs dans le panneau supérieur droit
- Affichage des métriques (homologue de l'onglet *Layout* de Firebug) dans le panneau central droit.
- Affichage des propriétés, *regroupées par prototype* (le bonheur !), dans l'onglet inférieur droit. Il faut cliquer sur le prototype qui nous intéresse pour voir les propriétés et méthodes qu'il fournit (pour bien des développeurs qui ne connaissent pas leur *DOM Level 2 Core* et leur *DOM Level 2 HTML* sur le bout des doigts, c'est très instructif !).

Jugez plutôt sur les figures D-9 à D-11 !

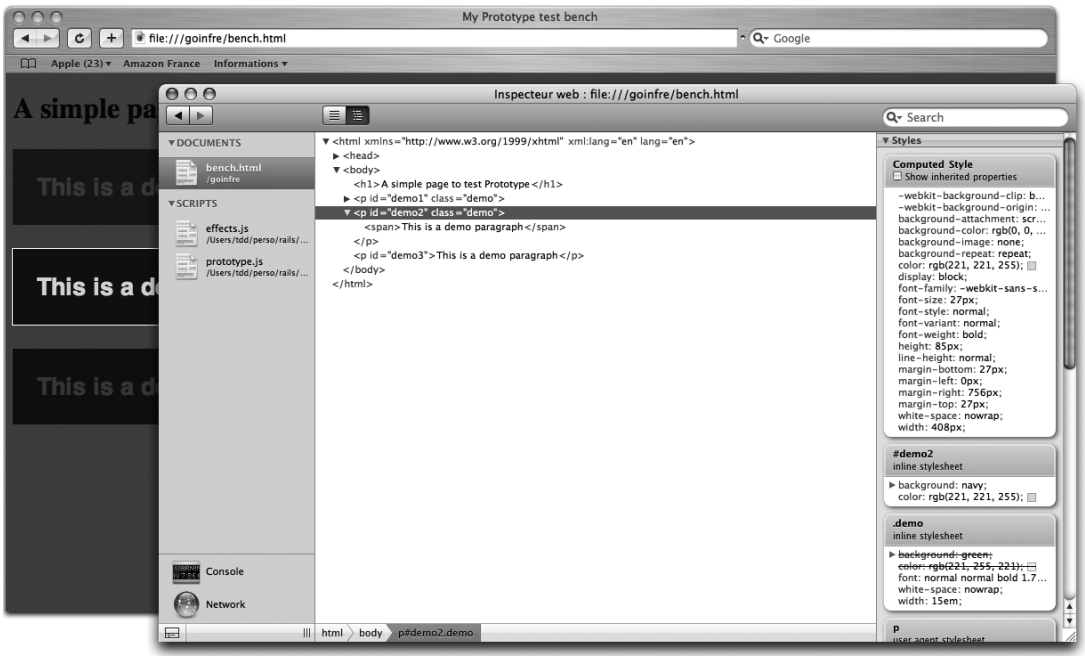



**Figure D–9** L'inspecteur web de Safari 3 explore le style d'un élément

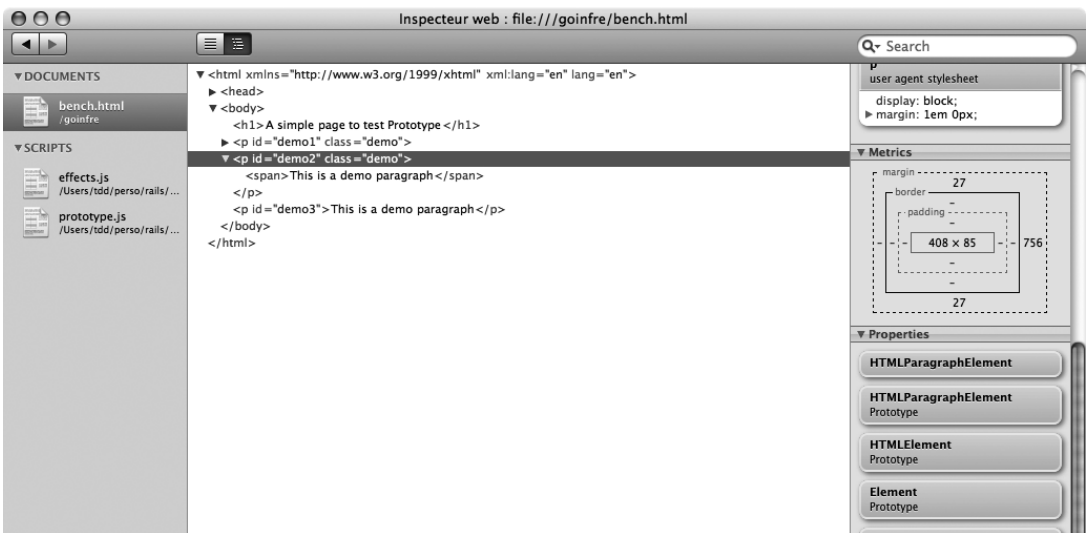



**Figure D–10** L'inspecteur web de Safari 3 affiche les métriques d'un élément

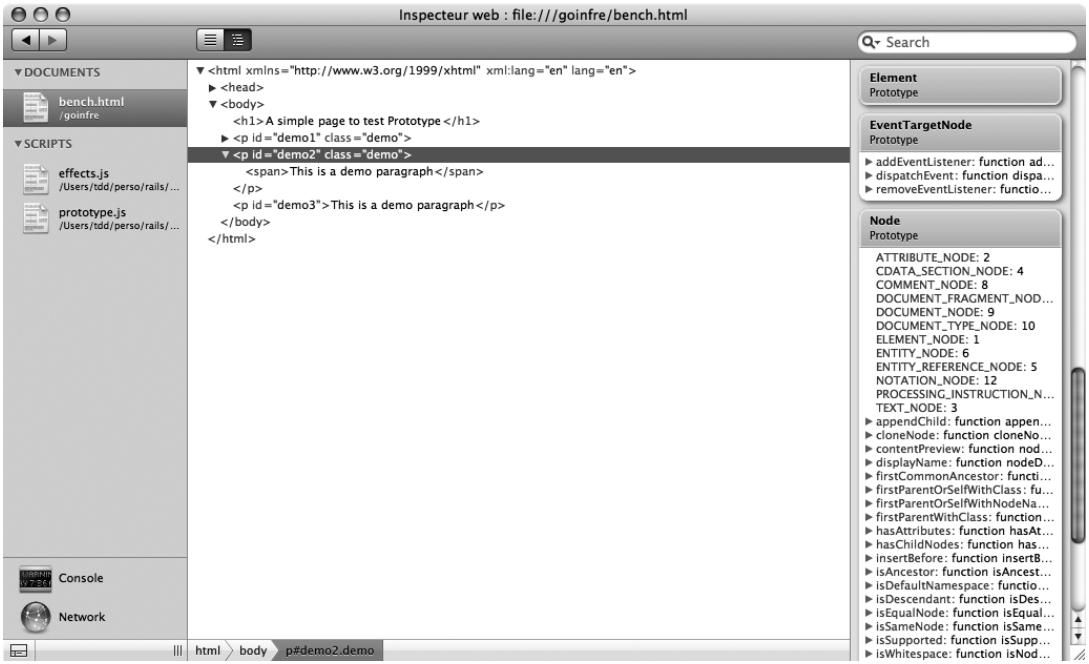



**Figure D–11** L'inspecteur web de Safari 3 affiche le DOM d'un élément

Seule ombre au tableau : tout ceci n'est pas modifiable ! On est vraiment en consultation, ce qui ralentit le travail de débogage/prototypage… C'est bien dommage.

Notez également le « chemin » de l'élément dans la page, affiché en bas de l'inspecteur, qui permet de restreindre la vue au contenu d'un élément précis. On trouve le même concept en haut de Firebug dans la vue *HTML*.

Cette fenêtre n'est en réalité par seulement l'inspecteur web, mais l'outil de développement général. Dans le panneau de gauche, les icônes *Console* et *Network* correspondent aux options *Afficher la console des erreurs* (*Cmd+Alt+C*) et *Afficher la vitesse réseau* (*Cmd+Alt+N*) du menu *Développement*.

La « console des erreurs » est en réalité une console JavaScript complète avec possibilité de saisie et d'évaluation immédiate, associée à un objet JavaScript `console` proposant quatre méthodes `error`, `info`, `log` et `warn`, exactement comme dans Firebug. Vos débogages de script à base de `console.log`, par exemple, fonctionnent donc aussi dans Safari 3 ! Attention quand même : l'exécution directe de code JavaScript depuis la console exécute d'abord le code, pour le « loguer » dans la console seulement après coup.

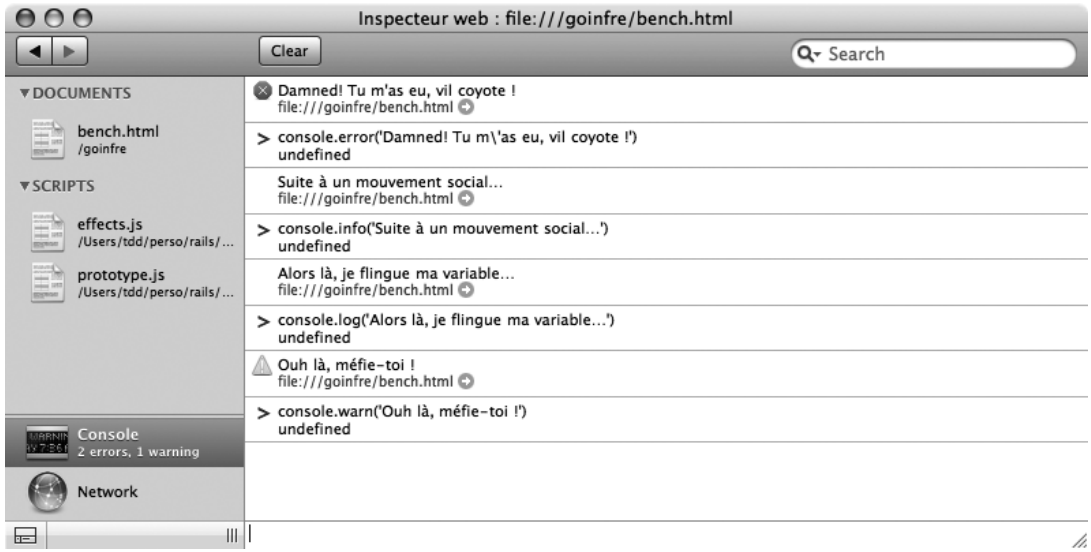



**Figure D–12** La console JavaScript de Safari 3 avec des appels à console.

Enfin, l'inspecteur de vitesse réseau affiche le diagramme de chargement des ressources pour la page sur un axe de temps, avec un codage couleur par type de ressource (documents/(X)HTML, images, feuilles de style, autres) et des cumuls. L'axe de temps est calculé automatiquement, sa borne droite correspondant au temps total de chargement de la page.

La figure D-13 montre les résultats de cet inspecteur sur l'accueil du site web de Paris-Web 2008.


**Figure D–13** Inspecteur de vitesse réseau sur Safari 3.

On peut par ailleurs cliquer sur n'importe quelle ligne pour avoir le détail des en-têtes de réponse pour la ressource concernée (et donc examiner l'éventuelle compression par le serveur, la taille ,le type MIME, et j'en passe). Vous en voyez un exemple sur la figure D-14.

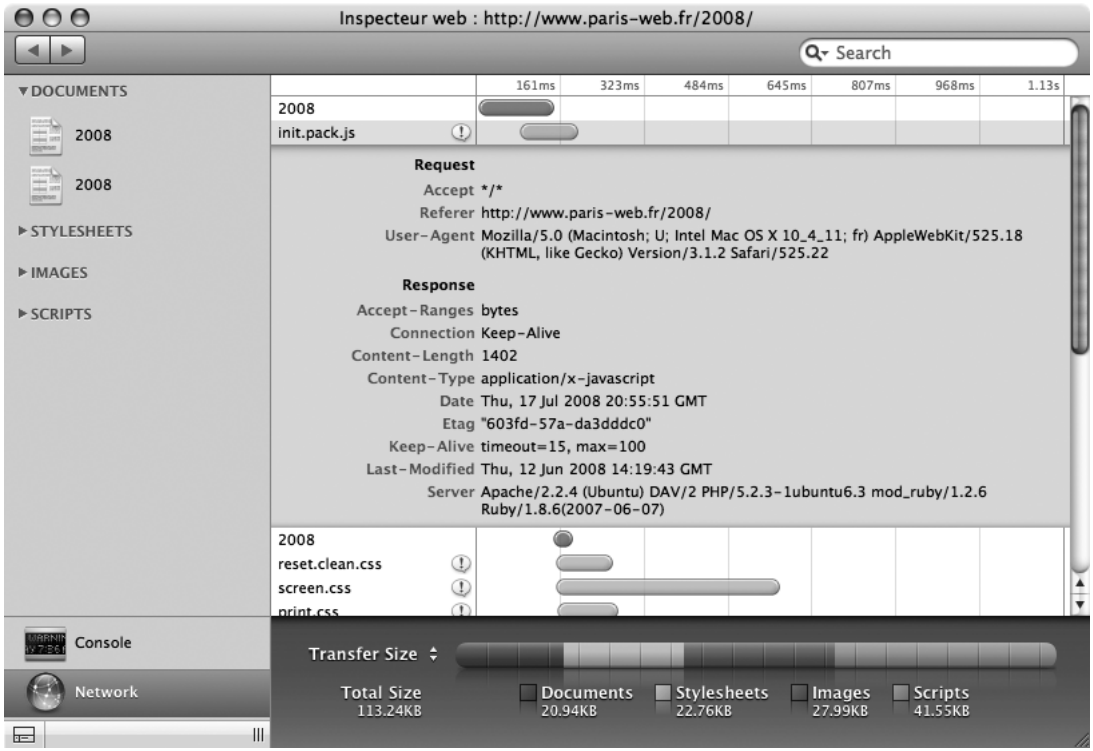
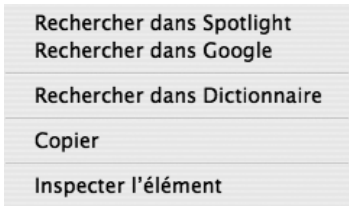



**Figure D–14** Inspecteur de vitesse réseau sur Safari 3 avec examen des en-têtes

Pour finir, sachez que l'inspecteur web peut être directement lancé sur n'importe quel élément d'une page : dans Safari 3, quand vous cliquez avec le bouton droit (ou *Ctrl+Clic*) sur un élément, vous disposez d'une option *Inspecter l'élément*, comme on peut le voir en figure D-15. Activer cette option ouvre automatiquement l'inspecteur web sur l'élément concerné.

**Figure D–15**
Le menu contextuel de Safari 3 permet d'inspecter directement un élément



En somme, Safari 3 n' a pas grand-chose à envier à Firebug (si ce n'est le suivi des requêtes Ajax) !

## MSIE 6-7 et la Internet Explorer Developer Toolbar

MSIE ne brille guère par ses possibilités natives. Pas de console JavaScript, une gestion des plus pénibles pour les erreurs et avertissements JavaScript... Toutefois, on peut améliorer la situation avec la **Internet Explorer Developer Toolbar**.

Très fortement inspirée de la Web Developer Toolbar de Chris Pederick (voir pour s'en convaincre http://www.thinklemon.com/weblog/stuff/WebDevelopervsDeveloperToolbar.jpg), elle se révèle cependant infiniment moins pratique à l'usage. On la télécharge, comme d'habitude, via une URL ahurissante :

http://www.microsoft.com/downloads/
details.aspx?familyid=e59c3964-672d-4511-bb3e-2d5e1db91038

On peut accéder à la barre en question *via* une petite icône supplémentaire en fin de barre d'outils, que vous pouvez voir en figure D-16.

**Figure D–16**
L'icône de la Internet Explorer Developer Toolbar.


Activer cette icône affiche le panneau correspondant en bas de page, comme on peut le voir en figure D-17.

On retrouve ici une petite partie des fonctions de Firebug et de la Web Developer Toolbar : arborescence du DOM, sélection d'un élément par clic sur la page, mise en avant de l'élément sélectionné, désactivation des images, du cache ou de JavaScript, entourage de certains types d'éléments.

C'est déjà bien, mais l'exécution est particulièrement inefficace.

- On ne peut que *partiellement modifier* le DOM (les attributs seulement, ni ajout, ni retrait, ni modification d'élément).
- L'affichage des styles affiche forcément toutes les propriétés, sans indiquer l'origine de leurs valeurs.
- Une autre vue permet de lister toutes les correspondances de règles CSS pour la page, mais seulement par règle et non par élément, ce qui n'est pratiquement d'aucune utilité…

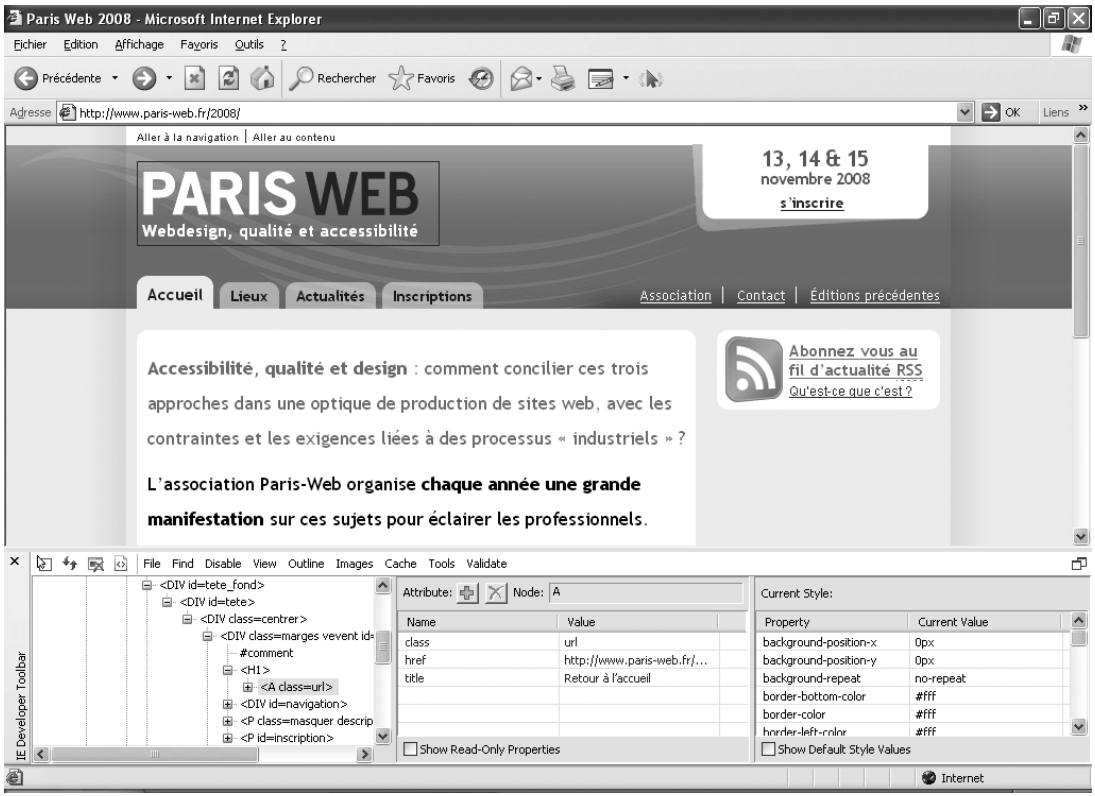



**Figure D–17** La IE Developer Toolbar en action

- Aucun débogueur JavaScript n'est présent, on n'a que la console, d'ailleurs très limitée. Il faut recourir à un débogueur externe, soit le Microsoft Script Debugger évoqué plus haut et au chapitre 5, soit des « poids lourds » de la famille Visual Studio (Express ou non), ce qui est démesurément lourd si vous ne les utilisez pas déjà par ailleurs.

Heureusement, la situation s'est un peu améliorée avec la prochaine version 8…

# MSIE 8b1 et son outil de développement intégré

La toute dernière version de MSIE a été dotée d'un outil intégré situé, en termes fonctionnels, quelque part entre la Internet Explorer Developer Toolbar, la Web Developer Toolbar de Chris Pederick, et Firebug. Il y a incontestablement des progrès depuis l'ancienne mouture, mais on est encore loin de l'efficacité et de l'ergonomie de Firebug.

On accède à l'outil depuis les menus ou *via* l'icône de barre d'outils qui est identique à celle d'Internet Explorer Developer Toolbar (la petite flèche bleue). On obtient alors le panneau de l'outil en bas de la fenêtre. La figure D-18 montre ce que ça donne en mode « fenêtre séparée ».

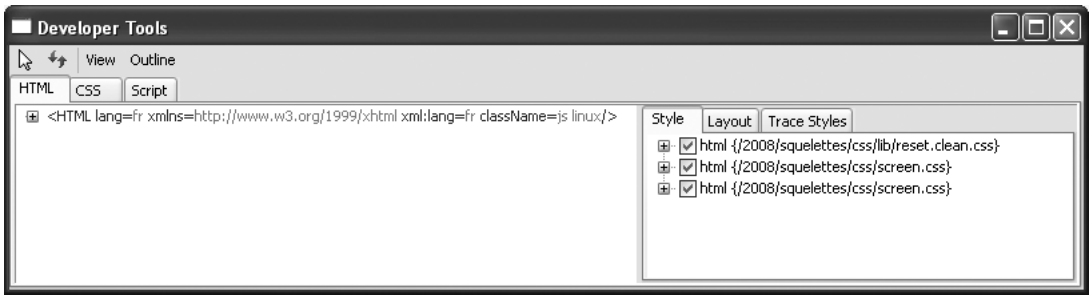



**Figure D–18** L'outil de développement intégré à IE8

À gauche, trois onglets :

- *HTML* est similaire à l'onglet homonyme de Firebug : il représente le DOM de la page sous format « code source », dépliable et repliable.
- *CSS* présente l'ensemble des CSS de la page, un format trop global pour être utile, un peu comme c'était le cas dans MSIE 6 et 7.
- *Script* est le débogueur JavaScript léger que nous avons déjà étudié en détail au chapitre 5.

À droite en mode HTML, trois autres onglets :

- *Style*, qui liste l'ensemble des règles appliquées à l'élément sélectionné dans l'onglet *HTML*. Ce n'est à mon sens pas le bon cheminement dans la consultation (des règles vers l'élément), ça rappelle un peu l'ancienne mouture…
- *Layout*, qui correspond exactement à l'onglet homonyme de Firebug et aux métriques de Safari 3, comme on peut le voir dans la figure D-19.
- *Trace Styles* pour terminer, qui fournit une autre manière d'examiner les styles de l'élément sélectionné : par propriété, avec les origines des valeurs (comprendre : les fichiers et règles qui les ont fournies). Hélas, à voir ce que ça donne (figure D-20), on en vient à se demander si les créateurs de l'outil ont jamais eu à déboguer leurs CSS ! Ce n'est guère utile : pas d'information claire sur quelle règle a « gagné » par rapport à quelle autre, pas d'aperçu des couleurs ni des images… On est bien loin de l'efficacité immédiate de l'onglet *Style* de Firebug. Espérons seulement que les choses iront mieux d'ici la sortie officielle.

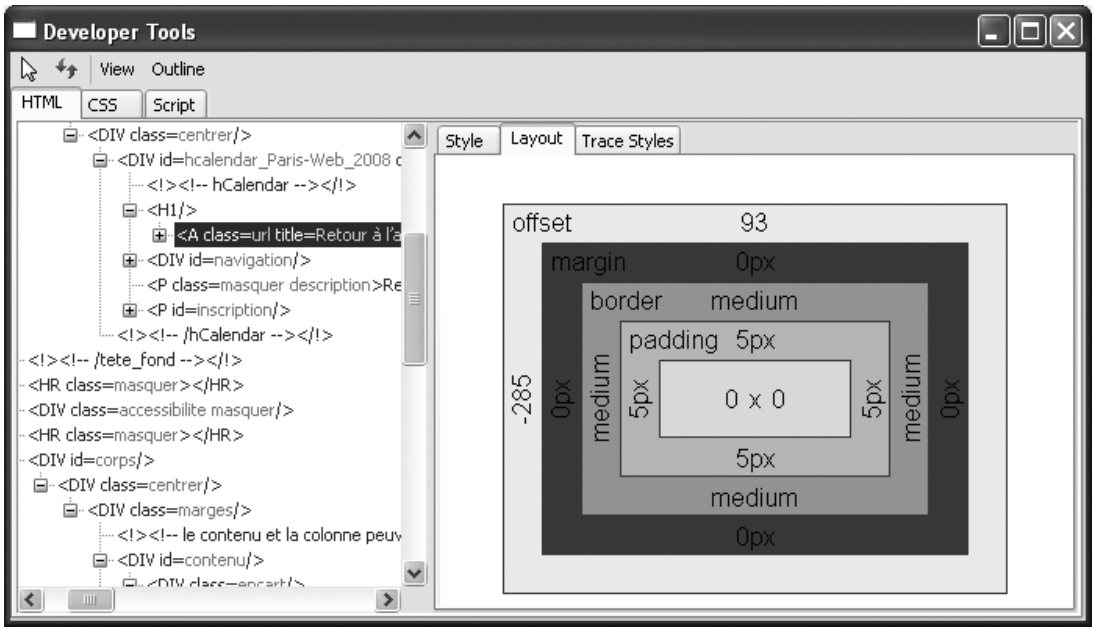



**Figure D–19** L'outil de développement intégré à IE8 et son onglet Layout

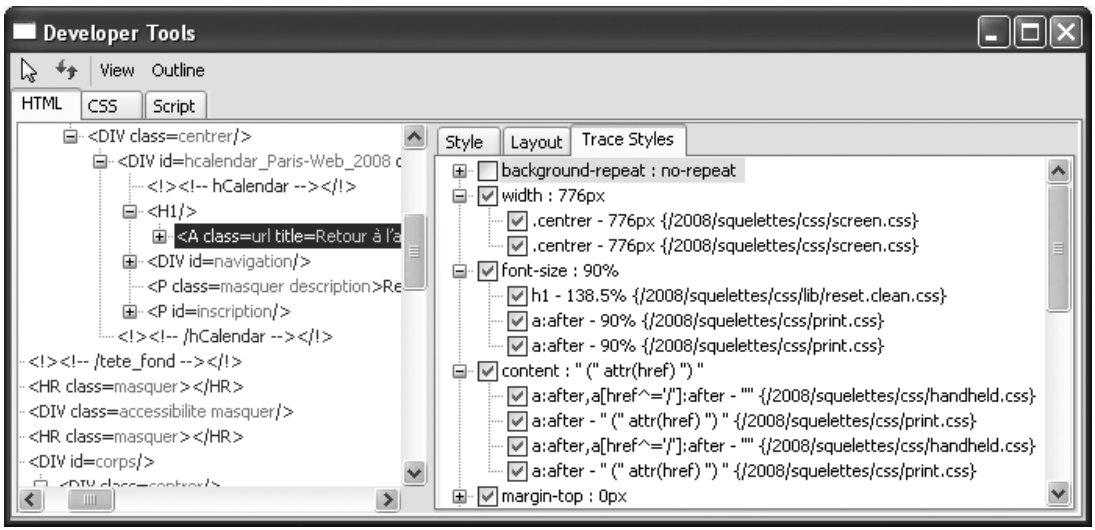



**Figure D–20** L'outil de développement intégré à IE8 et son onglet Trace Styles

Quelques précisions complémentaires :

- Dans l'onglet *HTML*, les noms et valeurs d'attributs sont désormais modifiables à la volée, plutôt que de devoir recourir à un encombrant panneau central comme

auparavant. En revanche « c'est œil pour œil, dent pour dent » : on a perdu la possibilité d'ajouter ou supprimer des attributs. Et on ne peut toujours pas ajouter, supprimer ou renommer des éléments.

- La sélection d'un élément dans l'arborescence l'entoure dans la page web correspondante pour faciliter son identification (comme pour les autres navigateurs, la palme revenant à Safari 3 qui met tout le reste de la page dans la pénombre…).
- Le menu *View* de l'outil permet de basculer entre les trois modes de rendu IE :
  - L'ancien mode 100 % propriétaire, baptisé *Quirksmode*, du temps de MSIE 5.x et MSIE 6/7 sans DOCTYPE.
  - Le mode « Strict » introduit par MSIE 6 lorsque la page déclare un DOCTYPE, et qui respecte mieux le modèle de boîte du W3C.
  - Le mode « Standard » de MSIE 8, beaucoup plus respectueux notamment du standard CSS. C'est le mode par défaut de ce navigateur !
- Enfin, le menu *Outline* permet d'entourer rapidement sur la page certains types d'éléments (blocs, cellules de tableaux, éléments positionnés, etc.).

## Opera et ses nouveaux outils de développement

Opera est très, très riche de fonctionnalités, notamment en ce qui concerne l'accessibilité. Mais côté développement, il n'offrait pas grand-chose jusqu'à récemment. La firme nordique a fini par monter une équipe dédiée à la réalisation d'outils intégrés s'adressant aux développeurs web, et à force d'être à l'écoute de leurs besoins, a enfanté un rejeton prometteur, baptisé « Dragonfly ».

On y accède par *Outils* > *Avancé* > *Outils du développeur* (*Ctrl+Alt+I*). Particularité potentiellement ennuyeuse toutefois : l'outil en question est accessible uniquement en ligne. Inutile donc d'espérer pouvoir déboguer un éventuel souci sur Opera lorsque votre ordinateur ne dispose pas d'une connexion réseau rapide (donc, pour le moment, dans l'écrasante majorité des trains et avions, par exemple).

Ceci dit, il faut garder un peu de perspective : la nécessité de déboguer une page (que ce soit sur XHTML, CSS ou le DOM et JavaScript) sur Opera alors qu'elle « passe » impeccablement sur Firefox et Safari est *extrêmement rare*. C'est là tout le bénéfice d'un bon respect des standards ! On peut sans problème concevoir que ces rares occasions auront lieu au bureau, avec une connectivité efficace…

La figure D-21 montre l'outil dans son état initial, une fois déporté dans une fenêtre séparée (ce qui est préférable vu sa disposition interne). On a plusieurs onglets principaux. Nous avons vu l'onglet *Script*, ouvert par défaut, dans le chapitre 5.

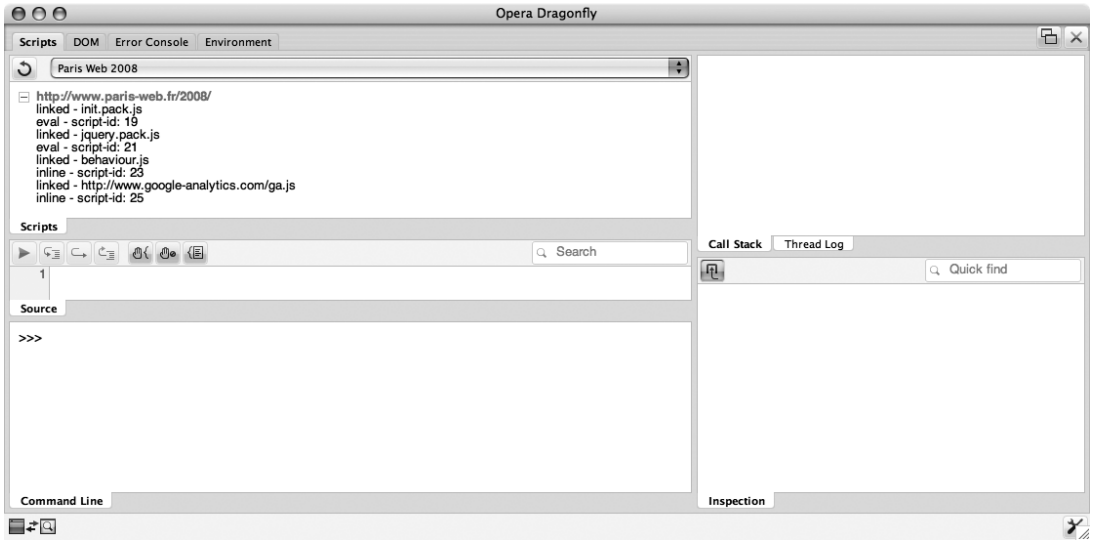



**Figure D–21** Opera Dragonfly et sa vue par défaut : le débogueur JavaScript

L'onglet *DOM*, visible plus en détail sur la figure D-22, permet de visualiser d'un seul coup d'œil l'arborescence du document et les propriétés CSS pour l'élément sélectionné (en prenant soin de distinguer celles qui ont été écrasées par d'autres et de les regrouper par feuille de style, façon Firebug).

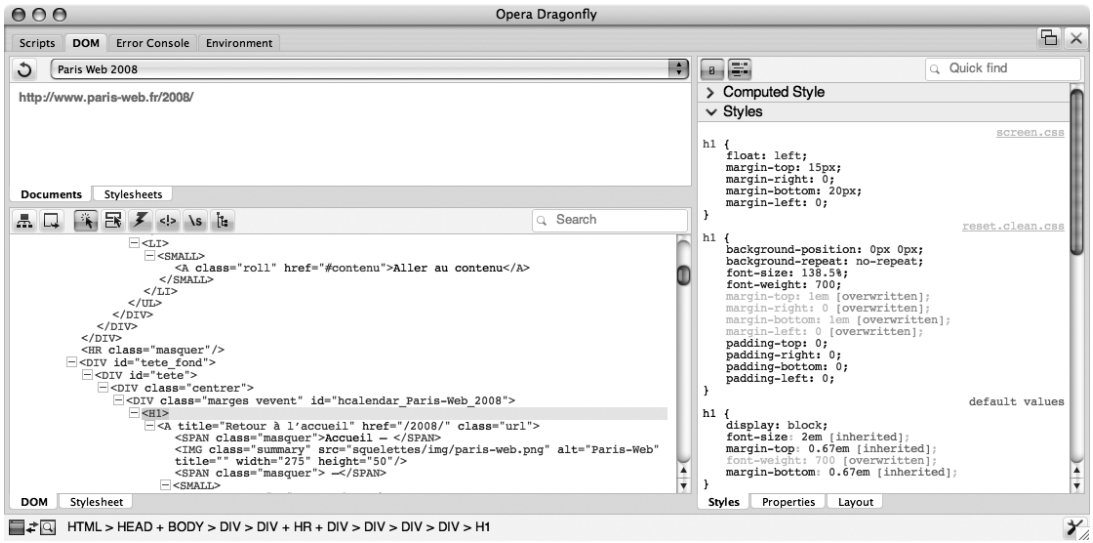



**Figure D–22** La vue DOM d'Opera Dragonfly : un parfum de Firebug…

Quelques remarques importantes :

- La vue DOM peut afficher ou masquer les commentaires et espacements superflus ; en revanche, elle n'est qu'en lecture seule.
- Par défaut, lorsque cette vue est active, cliquer dans le document lui-même affiche l'élément cliqué dans la vue DOM, mais ce comportement est désactivable, afin qu'on puisse continuer d'utiliser la page en même temps qu'on examine le DOM.
- Il y a en fait trois onglets à droite : les styles bien sûr, mais aussi *Properties* (les propriétés DOM de l'élément) et *Layout*, montré en figure D-23. C'est exactement l'organisation de la vue HTML de Firebug, mais la vue *Layout* est encore plus détaillée, car elle affiche toutes les propriétés de positionnement, et pas seulement celles du DOM…

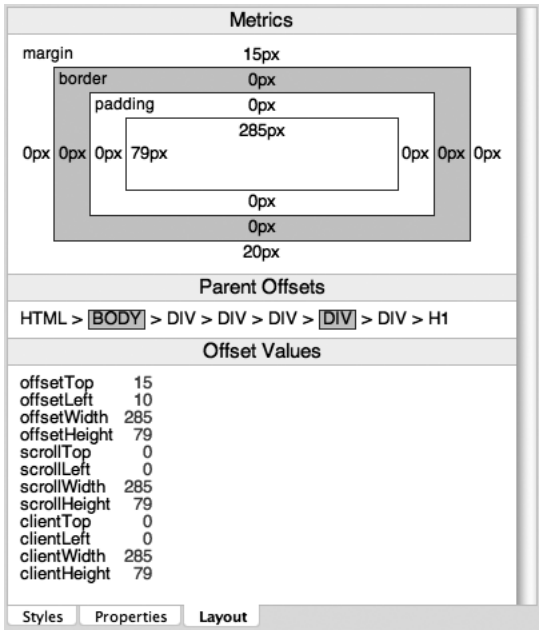



**Figure D–23**
L'onglet Layout de la vue DOM d'Opera Dragonfly : tout sous la main !

- La vue *Error Console* est la console d'erreurs classique. Je trouve toutefois la console d'erreur générale d'Opera beaucoup plus puissante (*Outils* > *Avancé* > *Console d'erreur*).
- La vue *Environment* n'est utile que lorsqu'on signale un bogue ou qu'on cherche de l'aide sur les forums : elle donne l'ensemble des informations sur les versions d'Opera et de Dragonfly qu'on utilise.

Avec Dragonfly, Opera a tapé très fort ! Et les habitués de Firebug trouveront immédiatement leurs marques dans l'onglet *DOM*…

# E

# Tour d'horizon des autres frameworks JavaScript

Lorsqu'il s'agit de bibliothèques JavaScript, ce livre se concentre sur les deux les plus populaires, Prototype et script.aculo.us. Toutefois, il en existe bien d'autres plus ou moins populaires, aux mérites variés. Personnellement, si je devais ponctuellement délaisser Prototype, ce serait sans doute pour YUI, si j'avais des besoins du genre « poids lourd »… En tous les cas, cette annexe vise à faire avec vous un rapide tour d'horizon des principales bibliothèques du marché.

## jQuery

Cette petite bibliothèque modulaire, dotée d'un bon système d'extensions, est développée par l'excellent John Resig, qui travaille pour Mozilla. Contrairement à bon nombre des bibliothèques qui suivent, elle se place exactement sur le créneau de Prototype : quelque part entre le simple lissage des incompatibilités JavaScript/DOM et le mastodonte façon SDK…

Ses performances sont bonnes, et elle bénéficie d'une forte croissance de popularité due en bonne partie à son excellent « éco-système », qui se développe au travers de son site officiel et de nombreux sites et forums communautaires, plus quelques livres.

jQuery se concentre sur la concision de la syntaxe et la facilité d'apprentissage. Je trouve pour ma part qu'en dépit d'avantages certains, elle manque de cohérence syntaxique d'une fonctionnalité à l'autre, ce qui rend l'API moins « découvrable » : ce n'est pas parce que tel groupe de fonctions est structuré de telle façon que tel autre suivra les mêmes principes de conception…

J'ajouterai pour finir qu'on a souvent dit que jQuery et Prototype étaient des rivaux acharnés. C'est faux. Les relations entre les concepteurs des deux bibliothèques sont des plus cordiales, au point que John a fait intégrer les suites de tests de Prototype dans le cycle de compilation de Firefox !

Quelques ressources à connaître :

- Le site officiel avec sa documentation, ses démonstrations, ses modules complémentaires et des liens vers les listes de discussions officielles et canaux IRC : http://jquery.com/
- Une communauté francophone assez active : http://www.jquery.info/
- Bear Bibault, grand fan de bibliothèques JavaScript, a écrit *jQuery in Action* chez Manning : http://www.manning.com/bibeault/
- De leur côté, Karl Swedberg et Jonathan Chaffer ont sorti l'excellent *Learning jQuery* : http://www.packtpub.com/jQuery/book/mid/1004077zztq0.

Eh oui, il n'existe pour le moment aucun livre sur jQuery en français !

## YUI (*Yahoo!* User Interface)

Là aussi, on a assisté cette année à une modification radicale du paysage : il n'y a qu'à voir comment YUI s'est taillé une belle place dans le panthéon des bibliothèques JavaScript.

La grosse différence entre YUI et les autres, c'est qu'il s'agit d'une bibliothèque « corporate », entièrement développée et financée par *Yahoo!*. Rien à voir avec le monde du logiciel libre : seul *Yahoo!* peut la développer. En revanche, elle répond aux besoins très concrets de *Yahoo!* pour ses nombreux sites web et services en ligne ; on y trouve donc un grand soin de la performance, de la modularité, et bien sûr une portabilité à toute épreuve.

C'est en effet YUI qui est au cœur des nombreuses applications *Yahoo!* comme Flickr, del.icio.us ou Upcoming.

YUI est constituée de nombreux modules, qui peuvent être chargés individuellement et dynamiquement grâce à un module de base spécifique. On est là dans la catégorie des poids lourds, avec un noyau dur, des modules multiples allant des effets visuels à

la gestion de sources de données Ajax en passant par la manipulation de l'historique, des cookies, de JSON, et de nombreux *widgets* (calendrier, sélecteur de couleur, menus, éditeur de texte formaté, onglets…). Et j'en passe !

Évidemment, vu les ressources considérables allouées à YUI, on trouve aussi une documentation de tout premier choix et la possibilité de récupérer les modules sur le réseau de distribution rapide (*Content Delivery Network*, ou CDN) de *Yahoo!*.

C'est vraiment un excellent système, surtout si l'on a des besoins extrêmement étendus. Je lui préfère la légèreté de Prototype et j'aime réaliser moi-même mes éléments visuels, mais j'imagine sans peine que pour beaucoup de développeurs web, YUI constitue une solution plus qu'alléchante.

Quelques ressources :

- Le site officiel, un vrai portail : http://developer.yahoo.com/yui/
- Un accès direct aux diverses ressources communautaires, notamment les groupes de support : http://developer.yahoo.com/community/
- Packt Publishing, qui a sorti *Learning jQuery*, a également permis à Dan Wellman de faire *Learning the Yahoo! User Interface Library* : http://www.amazon.com/Learning-Yahoo-User-Interface-library/dp/1847192327/

## Dōjō

Attention, mammouth (quoique raisonnablement petit une fois compacté puis compressé) ! Dōjō a démarré dans son coin avec son fondateur Alex Russel puis les gens de SitePen, notamment Dylan Schiemann. Depuis, Dōjō a fait un sacré chemin : la Dōjō Foundation comprend SitePen, bien sûr, mais aussi AOL, IBM et Sun Microsystems, pour ne citer qu'eux ! Gros support *corporate*, donc.

Dōjō est un *framework* complet modulaire, comme YUI. Il se prête particulièrement bien à la réalisation d'applications « desktop-like ». Depuis sa version 0.9, produit d'une refonte à partir de zéro, il se découpe en trois grands modules eux-mêmes très structurés :

- **dojo core**, le noyau. Équivalent à Prototype ou jQuery.
- **digit**, la partie interface utilisateur/*widgets* (système de thèmes visuels, éléments d'interface utilisateur, internationalisation, etc.)
- **dojoX**, qui regroupe les parties expérimentales et les plug-ins.

Le site officiel est superbe, et la documentation de référence est d'excellente facture. On trouve aussi beaucoup de didacticiels en ligne dont le *Book of Dōjō*, assez long et récemment remis à jour.

Dōjō brille particulièrement sur certains aspects innovants comme la gestion des données hors ligne ; Brad Neuberg a présenté il y a plus d'un an déjà Dōjō Offline sur la base de la technologie Google Gears…

Les ressources fondamentales et les bons bouquins :

- Le site officiel, la documentation, les listes de discussion, etc. : http://dojotoolkit.org/
- *Mastering Dōjō*, par l'équipe noyau de Craig Riecke, Rawld Gill et Alex Russel : http://www.pragprog.com/titles/rgdojo/mastering-dojo
- *Dōjō: Using the Dōjō JavaScript Library to Build Ajax Applications* par James E. Harmon : http://www.amazon.com/dp/0132358042/
- *Dōjō: The Definitive Guide* par Matthew A. Russel : http://www.amazon.com/dp/0596516487

## Ext JS

Ext JS est en fait un dérivé de la version 1.0 de YUI, porté par Jack Slocum, qui a démarré cette branche en 2006. On lui doit notamment les fondations des moteurs de sélection DOM par CSS qu'on retrouve aujourd'hui dans la plupart des autres frameworks.

Ext JS a toujours fourni toute une série de modules applicatifs, bien écrits, avec un système de thèmes, des grilles de données, des onglets, des menus, des gestionnaires de disposition… Tout à fait dans la cour de Dōjō et, logiquement, de YUI.

L'un des points les plus notables de Ext était qu'il avait été écrit de manière si modulaire et découplée qu'on pouvait le baser sur divers noyaux tiers comme jQuery ou Prototype ! Une performance impressionnante…

La version 3.0, prévue pour l'hiver 2008/2009, devrait inclure de nombreuses innovations et nouveautés comme la prise en charge de Comet/Bayeux (un point fort traditionnel de Dōjō) et une meilleure prise en charge d'ARIA, ce que je ne peux qu'encourager pour cette gamme de frameworks.

Aujourd'hui, Ext a grandi et une société a vu le jour. Elle s'appelle Ext LLC et édite deux variantes du framework : Ext JS qui est 100 % JavaScript, et Ext GWT qui fournit la même fonctionnalité mais en se basant sur GWT, dont l'approche est radicalement différente, comme nous le verrons plus loin. La société propose aussi un support technique commercial. Je ne suis pas contre ce modèle en soi, mais il est dommage que cette bascule commerciale ait, dans la pratique, sonné le glas d'une bonne partie de l'aspect « logiciel libre » du projet, et notamment de certains projets dérivés d'encapsulation GWT…

- Le site officiel est bien fait, les documentations de référence sont de bonne qualité et on trouve pas mal de didacticiels et d'exemples : http://extjs.com.
- En revanche, la récupération commerciale semble avoir engendré un léger désintérêt des auteurs face à une communauté (pourtant florissante encore récemment) qui est devenue un peu captive de l'entité commerciale : impossible de trouver le moindre livre sur Ext JS, même en anglais…

# Base2

Base2 nous vient de Dean Edwards. Et Dean, tous les *JavaScript ninjas* le révèrent. Développeur britannique en *freelance*, il est un peu comme un ermite qui, occasionnellement, descend de sa montagne et nous balance négligemment une pure petite merveille de JavaScript, qu'on met tous des heures, si ce n'est des jours, à comprendre. Ensuite, on s'en inspire et on sort les frameworks plus connus ; du script de Dean, ça se distille doucement, mais sûrement. Au bout de la cornue, le grand public…

Base2 est une sorte de plus petit dénominateur commun pour les autres frameworks : il se concentre sur le lissage des incompatibilités entre les navigateurs, et la mise au point d'un système d'héritage de classes. Pour tout le reste (Ajax, le glisser-déplacer, les effets, etc.) il laisse les autres s'amuser, ce n'est pas son problème : pour Dean, une fois qu'on a envoyé paître les innombrables petits obstacles de fond, on peut « laisser l'exercice au lecteur »…

Base2 fournit donc une sélection DOM rapide basée sur CSS, un DOM enfin conforme au niveau 2 officiel du W3C, l'événement `DOMContentLoaded` (équivalent du `dom:loaded` de Prototype), quelques méthodes fort utiles d'obtention de style calculé ou de position, et tout ça… à partir de MSIE 5.0 !

D'ailleurs, Dean est tellement fort qu'il est aussi l'auteur de l'immensément célèbre **IE7.js** (qui justement servait à torturer IE5 et IE6 pour qu'ils soient équivalents à IE7 ; il a depuis ajouté des trucs dans IE8.js, le malin), du très bon compresseur de script **/packer/**, et du premier système de sélection DOM par CSS, **cssQuery**, sur les ferments duquel tous les autres (y compris Jack Slocum pour Ext JS) ont basé leur travail (au moins au début).

Évidemment, Base2 ne séduit pas tellement les foules car il est trop « bas niveau », au sens où il ne fournit aucun service avancé côté utilisateur. Comme les autres frameworks réimplémentent l'équivalent, les développeurs web ne voient pas trop l'intérêt de Base2. Du coup, pas un bouquin en vue, et la documentation, bien que présente et complète, est succincte et sans exemples.

Mais je ne pouvais pas passer Base2 sous silence. Chaque nouvelle version suscite l'admiration et crée l'inspiration chez les développeurs des *autres* bibliothèques.

Le site officiel : http://code.google.com/p/base2/.

## MooTools

MooTools est arrivé un peu après la plupart des bibliothèques que j'ai citées ici. Commençons par les bonnes nouvelles : ça marche plutôt bien, la documentation est correcte, et c'est disponible *via* le CDN de Google, tout comme jQuery, Prototype et script.aculo.us.

MooTools est très modulaire et joue beaucoup sur ce point, en proposant deux *builders* en ligne qui permettent de choisir les modules que l'on veut, le type de compression que l'on souhaite (YUI Compressor, JsMin ou rien), et d'obtenir un script unique qui répond à tout ça. Le seul autre framework ayant un système de ce type aussi facile d'emploi, c'est YUI.

Enfin, MooTools, c'est une petite équipe motivée dirigée par le talentueux Valerio Proietti.

Toutefois leur popularité ne décolle pas vraiment. MooTools se classe plutôt dans la catégorie de jQuery et Prototype, et n'a pourtant pas la moitié de leur popularité suivant les sondages ; YUI le devance de loin. À quoi cela est-il dû ?

Je ne peux pas répondre à cette question avec certitude, mais peut-être est-ce dû à ceci (je parle pour moi, et seulement moi) : MooTools n'a, à l'heure où j'écris ces lignes, rien de significatif qu'on ne retrouve pas dans jQuery, Prototype ou YUI, mais surtout, *il est généralement en retard*.

Il y a évidemment des gens talentueux dans l'équipe MooTools (au premier rang desquels Valerio), mais le schéma qu'on ne cesse de constater est le suivant : une nouvelle fonction apparaît dans un des trois frameworks cités plus haut, et dans MooTools quelques jours ou quelques semaines plus tard. Pas forcément du copiage à proprement parler, mais tout de même, il y a eu quelques cas édifiants, qui ont nui à la réputation du framework et de son équipe.

Ajoutez à cela que certains anciens membres de l'équipe MooTools ont parfois dénigré un peu trop fort les autres frameworks en public, et vous comprendrez pourquoi le framework subit une réputation qui ne lui fait guère de bien. C'est dommage, parce qu'on y trouve du très bon code et que l'équipe a de nombreux membres de valeur, mais c'est ainsi.

- Le site officiel fournit la documentation de référence, les fameux *builders*, le blog de l'équipe et des exemples et démonstrations : http://mootools.net
- Je cite aussi un micro-moteur d'effets visuels (3 Ko compressé !) écrit par Valerio, qu'on peut utiliser avec MooTools ou Prototype : **moo.fx²**, téléchargeable sur http://moofx.mad4milk.net/.

# Et deux autres, très liés à la couche serveur…

Jusqu'à présent, nous avons vu des frameworks 100 % JavaScript, qui sont agnostiques quant à la couche serveur : on peut s'en servir aussi bien sur du PHP que du Ruby on Rails, du Java EE, de l'ASP.NET, du ColdFusion, et même de bons vieux CGI en Perl ou en C !

Il existe cependant deux frameworks assez répandus, ou en tout cas assez connus, qui méritent d'être cités. L'un d'eux est spécifique à une couche serveur ASP.NET, et l'autre à une couche serveur Java.

## GWT

Le *Google Web Toolkit* est une API Java extrêmement riche dont le principe fondamental est simple : vous écrivez votre interface web en Java, un peu comme vous décririez une interface en Swing. GWT se charge de compiler tout ça pour pondre du HTML, des CSS et du JavaScript qui passent partout, sont performants et interagissent de façon transparente avec votre couche serveur, en gérant les problématiques de persistance d'état, de flux de données, etc.

Autant je suis le premier à regarder avec avidité toutes les nouvelles API que nous sort Google (j'ai notamment déjà travaillé avec les API Maps, Maplets, Static Maps, Chart, Feedburner, Analytics, Checkout, Custom Search, Gears, OpenSocial, Sitemaps et YouTube), autant je suis plutôt du genre à fuir GWT.

C'est tout à fait personnel : étant moi-même féru de JavaScript avancé et très perfectionniste sur la sémantique et l'accessibilité de mon balisage, l'idée de laisser tout ça à une boîte noire (enfin non, c'est du logiciel libre, mais pour se farcir la masse de code et l'internaliser, bonjour…) qui, pour passer *vraiment* partout, fait forcément bon nombre d'entorses à mes idéaux, m'est plutôt douloureuse.

Il appartient à chacun de se faire sa propre opinion, comme toujours. La promesse de GWT a bien des aspects alléchants pour un chef de projet : des performances potentiellement excellentes, l'intégration avec de nombreux EDI (notamment pour

le débogage), la capitalisation des notions issues de Swing (gestionnaires de disposition, etc.), et de très nombreuses fonctionnalités.

Et *surtout*, ça évite d'avoir besoin de fortes compétences en JavaScript ; on peut réutiliser nos développeurs Java confortablement encadrés par le typage statique et les génériques, les avertissements du compilateur, la complétion automatique, la refactorisation, et tous les autres garde-fous.

Mais finalement, c'est fonctionnellement presque identique à Dōjō ou YUI… Avec en plus une contrainte forte côté serveur. D'ailleurs, sur les sondages de l'univers Ajax, GWT arrive loin derrière tous les frameworks cités précédemment.

On parle ici d'un projet Google massif et avec de forts investissements *corporate*, donc forcément, un très gros site et d'innombrables documentations : http://code.google.com/webtoolkit/.

## ASP.NET AJAX (ex-Atlas)

Étant doté d'une éthique de fer, je cite ASP.NET AJAX.

Bon, je vide mon sac tout de suite : je suis contre l'idée même d'un serveur web tournant sur plate-forme Windows (c'est le cas pour l'écrasante majorité des applications .NET), et je refuse l'idée qu'il me faut plusieurs Go de disque pour installer un EDI qui servirait à faire du Web (sans parler du peu de confiance que j'accorde, personnellement, à des gens qui sont à l'origine de Windows, Access, Notepad, Visual Basic et Paint).

Maintenant, je comprends parfaitement l'intérêt que peut présenter pour une équipe de développement le haut niveau d'intégration que propose Visual Studio.NET, ou même sa version Visual Studio Web Express (« juste » 1,6 Go pour l'installation, une paille). Et C# est vraiment très loin au-dessus des autres langages pris en charge par la CLR (ce qui est notamment dû au fait qu'il est piloté par l'immense Anders Hejlsberg, qui nous avait auparavant donné Delphi). Si vous faites déjà principalement du .NET en C# (par pitié, rendez-nous service, laissez tomber VB.NET…), capitaliser là-dessus semble être du simple bon sens.

Un petit rappel quand même : ASP.NET ne sait pas vraiment, sauf erreur de ma part, faire du MVC (on bidouille, mais on n'a pas un vrai découpage), et sa première tentative vers Ajax, à savoir les premières moutures d'Atlas, était tout simplement catastrophique de dysfonctionnements.

Tout le monde a droit à une deuxième chance, et justement le système a été renommé récemment en ASP.NET AJAX (au moins, on sait tout de suite de quoi ça parle), intégré à ASP.NET 3.5 et téléchargeable en complément de la version 2.0 (et pour une fois c'est petit, à peine 1,4 Mo !).

Très honnêtement, je ne sais pas si c'est pratique, utilisable et sans bogue, mais sa popularité semble arriver juste après celle de YUI, avant MooTools. Ça doit donc être plutôt pas mal !

Le site officiel, évidemment truffé de vidéos, didacticiels et documentations variées (l'avantage d'émaner du plus gros éditeur de logiciels au monde) : http://ajax.asp.net.

# Petit tableau récapitulatif

Voici un petit tableau qui récapitule les différents frameworks purement JavaScript (j'exclus donc GWT et ASP.NET AJAX, qui jouent sur un autre tableau).

**Tableau 5–1** Tableau E-1 Comparaison rapide des principaux frameworks

| | Prototype | jQuery | YUI | Dōjō | ExtJS | Base2 | Mootools |
|---|---|---|---|---|---|---|---|
| **Version** | 1.6.0.3 | 1.2.6 | 2.5.2 | 1.1.1 | 2.2 | 1.0β2 | 1.2 |
| **Taille noyau** | 140 Ko | 98 Ko | 31 Ko | 260 Ko | 160 Ko | 42 Ko | 90 Ko |
| **Taille noyau compressé** | 32 Ko | 14 Ko | 11 Ko | 24 Ko | 25 Ko | 6 Ko | 20 Ko |
| **CDN** | Google | Google | Yahoo! | Google | - | - | Google |
| **Espace de noms** | - | $/jQuery | YUI | dōjō | -/Ext | plusieurs | - |
| **Licence** | MIT | MIT/GPL | BSD | BSD/AFL | commerciales et GPL | MIT | MIT |
| **Taille équipe** | 10 | 5 | 10 | moult | 5 | 1 | 9 |
| **Éditeur** | Sam Stephenson | John Resig | Yahoo! | Dōjō Foundation | Ext JS, LLC | Dean Edwards | Valerio Proietti |

## La vraie fausse question de la taille…

Pour terminer, un mot sur la préoccupation inopportune qu'ont beaucoup de gens au sujet de la *taille* du framework, en termes d'impact sur le temps de téléchargement, et donc d'initialisation, de leurs pages.

Cette préoccupation est totalement infondée. Ne choisissez *en aucun cas* votre framework sur cette base. En effet, vos pages font régulièrement appel à des tas de ressources bien plus lourdes que le framework, et qui varient beaucoup plus souvent. Une petite mise en perspective s'impose…

Sur vos pages, il y a souvent :

- Une image JPEG 16/9 « pleine largeur » sur un site en 940 pixels de large, format extrêmement classique ne serait-ce que pour un fond ou un bandeau d'accueil, va peser dans les 70 Ko, déjà compressée (vers 75 % de qualité JPEG, ce qui est relativement nominal).
- Une animation Flash pour un menu ou une page d'accueil pèse régulièrement au moins 40 Ko, déjà compressée.
- Ces images varient souvent fréquemment, voire d'une page à la suivante, nécessitant un rechargement fréquent.

Et côté JavaScript ?

- Le plus gros framework du tableau précédent, une fois compressé correctement, pèse 32 Ko.
- La plupart de ces frameworks sont disponibles *via* un CDN, qui va donc décharger votre serveur et garantir un téléchargement optimal en vitesse.
- Ces frameworks ne changent que lorsque vous passez à une nouvelle version ; une fois téléchargés une première fois, si vous utilisez un CDN ou avez configuré correctement votre propre serveur, les navigateurs les gardent en cache aussi longtemps que vous le souhaitez (un mois, un an…). Ils sont donc disponibles instantanément, contrairement à vos animations, images et pages web en général.
- Même sur une mauvaise connexion ADSL (1 Mbit/s en descente, soit entre 5 % à 15 % d'une bonne connexion métropolitaine), l'écart maximal de temps de téléchargement entre les versions compressées des frameworks ci-dessus est de l'ordre *d'un dixième de seconde*. Même en version *décompressée*, on ne dépasse pas 1,8 s d'écart. Et ce, sur la première page uniquement.

La conclusion s'impose d'elle-même : oubliez les questions de taille.