CSS avancées

Vers HTML5 et CSS3

Raphaël Goetter

Préface de Daniel Glazman

EYROLLES

CSS avancées

Vers HTML5 et CSS3

R. Goetter

Webdesigner et gérant d'une agence web strasbourgeoise, **Raphaël Goetter** s'intéresse de près aux normes du Web et à l'accessibilité. Partageant ses connaissances via son site Alsacreations.com, il est déjà l'auteur du livre *CSS2 : Pratique du design web* et des mémentos *XHTML* et *CSS*, parus aux Éditions Eyrolles. Il fait également partie du collectif Openweb.eu.org, référence francophone en matière de standards du Web. Retrouvez-le sur Twitter @goetter !

Incontournable du design web moderne, les feuilles de styles CSS sont en pleine révolution avec l'adoption des nouveaux standards HTML5 et CSS3. Familier de CSS 2, allez plus loin en maîtrisant les techniques avancées déjà éprouvées dans CSS2.1 et découvrez les multiples possibilités de CSS3 !

Chaque jour mieux prises en charge par les navigateurs, les CSS sont sans conteste un gage de qualité dans la conception d'un site web élégant, fonctionnel et accessible, aussi bien sous Mozilla Firefox, Google Chrome, Opera ou Safari que sous Internet Explorer ou les navigateurs mobiles.

Vous croyiez tout savoir sur les CSS ? Grâce à ce livre, vous irez encore plus loin en apprenant à faire usage tout autant des technologies avant-gardistes de CSS3 et HTML5 que de pratiques avancées, concrètes et mal connues déjà utilisables en production, et ce, pour l'ensemble des médias reconnus par les styles CSS (écrans de bureau ou mobiles, messageries, mais aussi impression, médias de restitution vocale, projection et télévision). Maîtrisez tous les rouages du positionnement en CSS2.1, exploitez les microformats, optimisez les performances d'un site, gérez efficacement vos projets, ou contournez les bogues des navigateurs (hacks, commentaires conditionnels, HasLayout…). Enfin, profitez dès aujourd'hui des nouveautés de CSS3 : typographie, gestion des césures, colonnes, arrière-plans, dégradés, ombres portées, redimensionnement, rotations, transitions et autres effets animés, sans oublier les Media Queries, qui permettent d'adapter le site à son support de consultation.

Conseils méthodologiques, bonnes pratiques, outils, tests, exemples avec résultats en ligne, quizzes et exercices corrigés, tableaux récapitulatifs… rien ne manque à ce manuel du parfait designer web !

Au sommaire

À qui s'adresse cet ouvrage ?

- À tous les concepteurs de sites qui souhaitent exploiter les CSS au maximum de leurs possibilités ;
- Aux designers, développeurs et intégrateurs web impatients de découvrir et d'utiliser CSS3.

Conception : Nord Compo

www.editions-eyrolles.com
Groupe Eyrolles | Diffusion Geodif

CSS avancées

Vers HTML5 et CSS3

Chez le même éditeur

Du même auteur

R. Goetter. – **CSS 2 : pratique du design web.**
N°12461, 3e édition, 2009, 340 pages.

R. Goetter. – **Mémento CSS.**
N°12542, 2e édition, 2009, 14 pages.

R. Goetter. – **Mémento XHTML.**
N°12541, 2e édition, 2009, 14 pages.

Ouvrages sur le développement web

D. Cederholm. – **CSS 3 pour les Web designers.**
N°12987, 2011, 132 pages (A Book Apart).

J. Keith, préface de J. Zeldman. – **HTML 5 pour les Web Designers.**
N°12861, 2010, 90 pages (A Book Apart).

F. Daoust et D. Hazaël-Massieux. – **Relever le défi du Web mobile.**
Bonnes pratiques de conception et de développement.
N°12828, 2011, 300 pages.

J. Chable, D. Guignard, E. Robles et N. Sorel. – **Programmation Android.**
N°12587, 2010, 486 pages.

E. Sarrion. – **XHTML/CSS et JavaScript pour le Web mobile.**
Développement iPhone et Android avec et iUI et XUI.
N°12775, 2010, 274 pages.

J. Stark. – **Applications iPhone avec HTML, CSS et JavaScript.**
Conversion en natifs avec PhoneGap.
N°12745, 2010, 190 pages.

P. Chatelier. – **Objective-C pour le développeur avancé.**
Le langage iPhone/iPad et Mac OS X pour les développeurs C++/Java/C#.
N°12751, 2010, 240 pages.

E. Daspet et C. Pierre de Geyer. – **PHP 5 avancé.**
N°12369, 5e édition, 2008, 804 pages (Collection Blanche).

P. Borghino, O. Dasini, A. Gadal. – **Audit et optimisation MySQL 5.**
N°12634, 2010, 282 pages

R. Rimelé. – **Mémento MySQL.**
N°12720, 2e édition 2010, 14 pages.

C. Porteneuve. – **Bien développer pour le Web 2.0.**
Bonnes pratiques Ajax.
N°12391, 2e édition, 2008, 674 pages.

J.-M. Defrance. – **Premières applications Web avec Ajax, jQuery et PHP.**
N°12672, 2010, 474 pages.

A. Vannieuwenhuyze. – **Programmation Flex 4.**
N°12725, 2e édition, 2010, 604 pages.

G. Swinnen. – **Apprendre à programmer avec Python 3.**
N°12708, 2e édition, 2010, 410 pages.

Autres ouvrages

A. Boucher. – **Ergonomie web illustrée.** *60 sites à la loupe.*
N°12695, 2010, 302 pages (Design & Interface).

A. Boucher. – **Mémento Ergonomie web.**
N°12698, 2e édition, 2010, 14 pages.

E. Sloïm. – **Mémento Sites web.** *Les bonnes pratiques.*
N°12802, 3e édition, 2010, 18 pages.

M.-V. Blond, O. Marcellin et M. Zerbib. – **Lisibilité des sites web.**
Des choix typographiques au design d'information.
N°12426, 2009, 326 pages.

W. Lidwell, K. Holden et J. Butler. **Principes universels du design.**
N°12862, 2011, 272 pages.

O. Andrieu. – **Réussir son référencement web.**
N°12868, 3e édition, 2011, 462 pages.

T. Parisot. – **Réussir son blog professionnel.**
Image, communication et influence à la portée de tous.
N°12768, 2e édition, 2010, 322 pages.

F.-X. et L. Bois. – **WordPress 3 pour le blogueur efficace.**
N°12829, 2011, 358 pages.

M. Blanchard. – **Magento.** *Réussir son site e-commerce*.
N°12515, 2010, 352 pages.

Y. Brault, préface d'E. Plenel. – **Concevoir et déployer ses sites web avec Drupal 6 et 7.**
N°12780, 2e édition, 2010, 420 pages.

V. Isaksen et T. Tardif. – **Joomla et Virtuemart.**
Réussir sa boutique en ligne.
N°12487, 2e édition, 2009, 336 pages.

N. Chu. – **Réussir un projet de site web.**
N°12742, 6e édition, 2010, 256 pages.

V. Messager Rota. – **Gestion de projet agile.**
Avec Scrum, Lean, eXtreme Programming...
N°12750, 3e édition, 2010, 272 pages.

CSS avancées

Vers HTML5 et CSS3

Raphaël Goetter

Préface de Daniel Glazman

EYROLLES

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

Remerciements de l'éditeur à Anne Bougnoux pour sa relecture et à Daniel Glazman, Jens O. Meiert et Jim Morrison pour leurs contributions.

© Groupe Eyrolles, 2011, ISBN : 978-2-212-12826-0

Préface

Au début étaient la pierre, le bois, l'argile, le métal, le papyrus et finalement le papier. Des supports pour lesquels fond et forme sont inextricablement mêlés. Séparer la lettrine de son enluminure ? Imaginer le Talmud sans son formatage si spécial permettant les commentaires ? Impossible ! Pire, les éléments de forme étaient fortement dépendants du support : la typographie ronde était difficile sur pierre et impossible dans l'écorce de bois, tandis que les barres supérieures de certaines graphies avaient pour utilité d'aider l'alignement.

La révolution technologique a non seulement séparé fond et forme dès la naissance du télégramme, mais elle a également séparé fond et format, les lettres et les chiffres n'étant plus des lettres et des chiffres, mais des signaux de morse transitant dans un fil métallique. Le Web, cette nouvelle révolution dont seuls nos descendants mesureront à sa juste valeur la portée, va encore plus loin et officialise enfin ce vieux leitmotiv des fanatiques de la documentation structurée : contenu et présentation sont deux notions quasi orthogonales. Un contenu donné peut être présenté de plusieurs manières différentes, une présentation peut être commune à plusieurs contenus sans rapport entre eux.

Lorsque le Web naît au CERN entre 1989 et 1991 sous l'impulsion de Tim Berners-Lee, rien de tout cela n'existe encore. Chaque élément de la *lingua franca* du Web, le langage HTML, véhicule intrinsèquement sa propre présentation et styler un contenu n'est pas encore une idée en vogue. On est encore bien loin de ce qu'offre la PAO...

C'est là qu'interviennent Håkon Lie, un Norvégien qui travailla au CERN avec Tim Berners-Lee et fut l'un des premiers employés du World Wide Web Consortium, et Bert Bos, un Néerlandais étudiant à l'Université de Groningen. Extrayant la substantifique moelle des technologies de style documentaire existantes et comprenant que le style peut se décliner en styles voulus par l'auteur du document, styles par défaut de l'outil de visualisation et enfin styles imposés par le lecteur, ils ont élaboré de 1993 à 1995 le concept de Feuilles de styles en cascade (en anglais, *Cascading Style Sheet*, d'où CSS).

Les débuts furent difficiles. Les éditeurs de navigateurs web se livraient une guerre acharnée et une solution standard, interopérable et, surtout, exigeant des changements fondamentaux dans leur code, n'était pas nécessairement bienvenue. Il fallut donc attendre un très grand virage sur l'aile, celui de Microsoft vers Internet et le Web, pour voir enfin les CSS implémentées de façon sérieuse et extensive dans un navigateur web. À titre de rappel, le premier navigateur à proposer le support intégral des CSS 1 fut Microsoft Internet Explorer pour Macintosh...

Netscape finit par abandonner son idée de styles fondés sur du code JavaScript (JSSS, *JavaScript-based Style Sheet*) et bascula vers les CSS. L'heure du succès était venue et la seconde mouture du standard, les CSS 2, s'aventura dans des champs encore inexplorés sur le Web : les polices de caractères téléchargeables, l'impression, le positionnement fin, et encore beaucoup d'autres nouveautés.

Mais les hommes n'étant finalement que des hommes, CSS 2 alla trop loin pour eux et l'implémentation des CSS 2 dans les navigateurs ne fut jamais à la hauteur des espoirs initiés par la spécification elle-même. Certaines fonctionnalités étaient sous-spécifiées, certaines autres posaient problème, certaines étaient même tout simplement impossibles à implémenter en l'état de l'art ou de la spécification.

Le World Wide Web Consortium (W3C) s'attacha donc à la révision des CSS 2 en même temps qu'il planchait sur la future mouture, CSS 3. Cela prit un certain temps, voire un temps certain. Malgré une certaine exaspération toute légitime du côté des éditeurs web, cela eut un effet très positif en laissant aux éditeurs de navigateurs le temps de profiter de nombreuses améliorations *hardware* et *software*. Un navigateur web de 2010 n'a plus rien à voir avec un navigateur web de 2000, même si l'usager ne s'en rend pas toujours compte.

Aujourd'hui, CSS 2.1 est enfin en phase finale de standardisation. Quant aux CSS 3, elles ne sont pas un rêve éthéré mais une réalité déjà utilisable dans tous les navigateurs du marché. Tous ? Oui, tous, y compris Internet Explorer 9.

Non seulement plus personne ne conteste le modèle et l'utilité des CSS, mais plus personne ne conteste non plus leur légitimité en tant que langage unique de feuilles de styles sur le Web. Les fonctions de formatage simples des CSS 1 ont été grandement étoffées, et les dégradés de couleurs, les transformations géométriques, le texte en colonnes, les polices de caractères téléchargeables, ou encore le texte vertical et l'internationalisation, promettent de servir des sites web encore plus modernes, plus réactifs, plus conformes aux standards, plus aisés à réaliser ou à maintenir et tout simplement plus beaux, à encore plus d'internautes dans le monde, sur ordinateur ou sur mobile.

Que vous soyez l'éditeur d'un grand site de presse ou celui d'un petit blog, la conception de votre site passe immanquablement par les CSS. Et continuera encore plus à l'avenir à passer par les CSS… Car l'histoire ne s'arrête pas là : le Groupe de Travail standardisant les CSS au W3C continue à avancer, à répondre de mieux en mieux aux demandes des designers web ou même des typographes. La convergence entre le Web et les autres métiers du design documentaire est en marche : grilles de design, modèle flexible de présentation, etc. Vous allez adorer ça autant que nous aimons le standardiser et l'implémenter.

Le livre de Raphaël Goetter est donc, pour tout auteur de site ou rédacteur d'une *newsletter* à envoyer par e-mail, un *must* qui lui permettra non seulement de tirer parti des nouvelles technologies du Web (CSS 2.1 et 3, HTML 5), mais également d'éviter les chausse-trapes.

Entre le hêtre (**bōkz* en proto-germanique) et l'*e-book*, le mot n'a que peu varié. Et la maîtrise du second requiert toujours la lecture du premier. Bonne lecture, donc !

Daniel Glazman

W3C CSS Working Group, Co-chairman

Table des matières

PREMIÈRE PARTIE

ANNEXES

Avant-propos

Vous croyiez tout savoir sur les CSS ? J'ai une bonne et une mauvaise nouvelle pour vous...

Ce projet d'ouvrage est né d'un constat évident : nul ne peut actuellement remettre en question les avantages d'une mise en page de site Internet à l'aide des feuilles de styles CSS. Du reste, les didacticiels en ligne, les organismes de formation et les livres d'apprentissage ont parfaitement intégré cette composante indissociable du design web.

Les ressources de base pour débuter et apprendre à utiliser les CSS sont légion, mais finalement peu d'outils osent aller plus loin et proposer des techniques véritablement avancées, pourtant chaque jour un peu plus accessibles grâce aux fréquences de renouvellement des navigateurs.

Ce livre se veut une passerelle entre des technologies avant-gardistes telles que CSS 3/HTML 5 et des pratiques avancées, concrètes et mal connues que l'on peut employer en production dès aujourd'hui.

Enfin, son objectif est de traiter tous les médias reconnus par les styles CSS : j'y évoque aussi bien les écrans de bureau que les mobiles, l'impression, les messageries, les médias de restitution vocale, de projection et de télévision.

Dans cet ouvrage, j'ai réuni avec soin de multiples aspects couverts par les feuilles de styles au sein d'un design web moderne :

- des fonctionnalités de CSS 2.1 oubliées à cause d'Internet Explorer 6 (sélecteurs d'adjacence, de parenté, d'attribut, génération de contenus en CSS, règles @) ;
- des schémas de positionnement classiques détaillés, avec astuces ;
- une introduction aux positionnement « avancés » méconnus (`inline-block`, modèle tabulaire, modèle de boîte flexible et autres positionnements CSS 3) ;
- l'usage des microformats ;
- la gestion de projet en CSS, l'optimisation des performances, les frameworks et autres outils CSS ;
- HTML 5 (syntaxe, attributs, formulaires, `audio`, `video`, `canvas`, API) et CSS 3 (propriétés, sélecteurs, préfixes, requêtes de média, transformations, animations) ;
- la prise en compte de la compatibilité des navigateurs et de la résolution de bogues (*hacks*, commentaires conditionnels, *HasLayout*) ;
- la gestion des médias avec CSS : CSS écran, CSS pour mobiles, CSS pour l'impression, CSS et messageries, média vocal, média de projection et média de télévision.

Bonne lecture !

Le site dédié au livre

Dans le but de prolonger votre expérience en dehors de ce support papier, une page web spécialement consacrée à cet ouvrage a été conçue et peut être consultée à l'adresse :

▸ http://www.goetter.fr/livres/css-avancees

Vous y trouverez une présentation et la table des matières du livre, un mot sur l'auteur, quelques extraits, mais aussi d'éventuels errata et des retours de lecteurs. N'hésitez pas à le consulter pour y laisser votre avis ou y dénicher une information.

Pour ce qui est des codes et exemples utilisés tout au long de ce livre, et notamment dans le cadre des exercices pratiques, c'est sur mon bac à sable personnel que vous les trouverez, en compagnie de bien d'autres cas d'étude, à l'adresse :

▸ http://www.ie7nomore.com

Conventions utilisées dans ce livre

Cela va sans dire – en tout cas je l'espère –, mais tous les contenus, méthodes et exemples décrits dans cet ouvrage respectent tant que faire se peut l'esprit général véhiculé par les standards W3C en termes de conformité de langage et d'accessibilité. Il se peut toutefois que certaines sections destinées à mettre en avant les prouesses des feuilles de styles s'écartent légèrement de certaines bonnes pratiques. Si tel est le cas, je ne manquerai pas de le signaler afin que vous puissiez juger l'intérêt de l'exercice en connaissance de cause.

W3C : le garant des standards

Le W3C (*World Wide Web Consortium*) est un organisme mondial créé par Tim Berners-Lee dont l'objectif est de garantir la pérennité et l'interopérabilité des standards de langage web. Composé de la plupart des acteurs de la toile (Microsoft, Mozilla, Opera, IBM, …), le W3C a pour charge de rédiger et proposer les spécifications HTML, XHTML, CSS, SVG, ECMAScript, etc.

▸ http://www.w3.org

Je traiterai généralement les notions de HTML et de XHTML indifféremment puisque, dans la pratique, seule la déclaration initiale (`Doctype`) diffère, mais que la méthodologie et la rigueur sont similaires.

Cet ouvrage balaye trois générations de spécifications CSS. A priori, le terme de « CSS » englobera naturellement toutes les générations, sauf si j'en indique précisément la version.

Les codes et exemples proposés ont systématiquement été testés sur diverses versions de navigateurs lors de la rédaction de l'ouvrage : Internet Explorer 6, 7, 8 et 9 (ce dernier est sorti en version bêta à la fin de l'année 2010), Firefox 3.6, Firefox 4, Opera 10.6 et 11, Chrome 7 à 9 et Safari 5.

À propos de l'auteur

Papa de deux adorables bambins, passionné d'écriture puis de photographie, et pas franchement « geek », rien ne me prédestinait à atterrir dans l'univers de la conception web.

Mon parcours professionnel témoigne d'ailleurs d'un goût prononcé pour sortir des sentiers battus : après un baccalauréat scientifique, je me lance à corps perdu dans une faculté… de sport. Timide et peu enclin à botter les fesses des garnements n'ayant pas envie de courir autour d'un stade, je quitte en 1998 le cursus universitaire pour m'improviser « webmaster » d'une association sportive, sans véritable bagage dans ce domaine bien entendu. J'en profite là pour avouer avoir conçu mon premier site web à l'aide de la fonction Enregistrer pour le Web de Microsoft Publisher 98.

C'est en tant qu'emploi jeune dans le club sportif de la Constantia Strasbourg (www.slconstantia.com) que je commence véritablement à m'épanouir dans le Web en parfait autodidacte. Un diplôme universitaire de webmaster, anecdotique car déjà obsolète pendant la formation, vient sceller mon apprentissage.

C'est en 2003 très précisément que je tombe amoureux de la conception de sites conformes via les feuilles de styles CSS, puis de l'accessibilité du Web suite à des débats houleux sur un forum de discussion spécialisé où certains critiquaient violemment (qui l'eût cru ?) mon code source produit par Publisher.

Quelques mois plus tard, après une radicale remise en question de mes connaissances, les premiers tutoriels de mise en page CSS apparaissent sur mon nom de domaine fraîchement acquis, Alsacreations.com (je me ronge encore les doigts d'avoir choisi un nom se terminant par un « s » tant il est fréquemment oublié par les internautes !).

Embarqué dans cet élan créatif, un autre projet d'envergure se concrétise en 2005 : la rédaction et la publication d'un livre sur les CSS aux éditions Eyrolles. Cet ouvrage, *CSS 2 : Pratique du design web*, constitue un véritable pari puisqu'à cette époque, les mises en page via tableaux et cadres sont tellement légion que les standards web et les CSS peuvent passer pour des lubies (pensez qu'Internet Explorer 6 écrase littéralement le marché des navigateurs). Aucun véritable livre francophone n'avait alors été produit, hormis des traductions d'ouvrages américains.

Aujourd'hui, les navigateurs et nos pratiques de conception web ont évolué, les standards sont ancrés dans les mœurs et nous avons besoin d'aller au-delà d'un simple livre d'initiation à XHTML et CSS. Nous sommes prêts à passer la vitesse supérieure et à découvrir les techniques avancées et redoutablement efficaces offertes par ces langages en constante évolution. L'ouvrage que vous tenez dans vos mains est né de ce constat.

C'est d'ailleurs exactement le même discours que je tiens lors des différentes formations CSS que j'anime (http://formations.alsacreations.fr), puisque j'ai aujourd'hui le bonheur d'en avoir fait mon métier.

Si vous ne me rencontrez pas dans une salle de formation, ou lors du cycle de conférences annuel Paris-Web (événement que je vous recommande chaudement !), vous aurez beaucoup de chance de me croiser virtuellement sur le forum de discussion d'Alsacréations (http://forum.alsacreations.com) ou sur mes sites personnels goetter.fr (figure I-1) et IE7nomore.com, qui me servent à la fois d'exutoire et de bac à sable.

Féru des réseaux sociaux, je communique fréquemment mes découvertes CSS via mon compte Twitter @goetter et l'on peut me trouver pour des échanges professionnels sur LinkedIn et Viadeo.

Figure I-1

goetter.fr, mon site personnel, agrémenté de HTML 5 et CSS 3

À propos d'Alsacréations

Pour les lecteurs non familiers de mon univers, sachez qu'Alsacréations revêt une double facette : d'un côté le site communautaire www.alsacreations**.com** et de l'autre, l'agence web qui en a découlé, www.alsacreations.fr.

Si vous êtes pressé, passez cette section mais retenez simplement que « Alsacréations » s'écrit toujours avec un « s » final. Cela m'évitera de payer plusieurs noms de domaines à l'avenir...

Alsacreations.com

Alsacreations**.com** est une communauté d'échange et de partage autour des standards web et de la conception conforme aux normes, née en 2003. On peut y dénicher des astuces, des tutoriels,

un forum de discussion (depuis 2004) et un service emploi dans le Web très actifs (plus de 35 000 membres au sein du forum). Par philosophie personnelle, tous les contenus et apprentissages diffusés sur Alsacreations.com sont libres de droit et bien entendu gratuits.

Alsacréations communique également sous le compte Twitter @alsacreations, regroupant les membres de l'équipe de modération du forum. Chacun peut au nom d'Alsacréations poster ou relayer une information jugée notable sur le réseau Twitter.

Figure I-2

Alsacreations.com

Alsacreations.fr

Alsacréations est aussi devenue, en 2006, une agence web alsacienne multitâche à dimension humaine, actuellement composée de cinq passionnés et experts dans leurs domaines de prédilection. Le site de l'agence se trouve à l'adresse www.alsacreations.**fr** (figure I-3).

Figure I-3

Alsacreations.fr

Notre cœur de métier se situe bien évidemment dans le domaine de la conception web « propre », mais concerne un large panel de compétences : conception graphique, intégration HTML/CSS, mise en conformité, développements PHP/MySQL, Flash, CMS, hébergement, audits d'ergonomie et d'accessibilité, formations diverses dans le monde du Web.

Le site web Alsacreations.fr constitue un défi pour nous puisqu'il fut développé en HTML 5 et CSS 3 dès 2010.

Remerciements

Une fois n'est pas coutume, ma première pensée à la fin de cette aventure va tout naturellement à ma dévouée femme et à sa motivation dépassant parfois la mienne. Si ce livre a pu voir le jour, c'est en grande partie grâce à *sa* persévérance et à son aptitude à me convaincre de passer de trop longues soirées devant un écran d'ordinateur plutôt qu'en compagnie de ma petite famille.

Dans un second temps, je remercie humblement Daniel Glazman, membre influent du W3C et éminent co-responsable du groupe de travail sur les spécifications CSS de m'avoir fait l'immense honneur d'accepter de signer la préface de cet ouvrage.

Je suis enfin extrêmement reconnaissant envers les multiples contributeurs francophones que je connais dans le monde de la conception web et qui, par leurs articles, billets de blog et autres messages sur Twitter, partagent librement leurs connaissances et contribuent, chacun à son niveau, à l'enrichissement de notre profession. Je pense notamment à Laurence Vagner, Pascale Lambert-Charreteur, Monique Brunel, Karl Dubost, Élie Sloïm, Tristan Nitot, Eric Daspet, Bruno Bichet, Benjamin De Cock, Florent Verschelde, Jean-Pierre Vincent, Jérémie Patonnier, Antony Ricaud, Aurélien Levy, Philippe Le Mesle, Denis Boudreau et tellement d'autres que la liste ne sera jamais exhaustive.

Et bien entendu, je remercie mon *tourmenté* collègue Rodolphe Rimelé, mon premier relecteur et bientôt auteur d'un admirable livre sur HTML 5, et rends un hommage sincère à la maison d'édition Eyrolles qui m'accompagne depuis le début, toujours aussi efficacement, dans mes différents projets d'écriture.

1

État des lieux

Ce chapitre retrace brièvement les dernières évolutions des standards du Web, à travers les évolutions des usages, des différentes normes et des navigateurs. Vous y découvrirez plus particulièrement les améliorations des dernières versions du navigateur Internet Explorer, notre bête noire tout au long de cet ouvrage.

Les standards du Web, une longue mise en place

Le Web a connu un développement révolutionnaire entre la première page HTML mise en ligne en 1990 et aujourd'hui. En une petite vingtaine d'années, les normes et outils dédiés au monde virtuel ont tellement évolué que les années 1990 sont parfois qualifiées de « préhistoire » du Web. Dans cet univers en progression constante, certaines technologies – pourtant nées durant cette préhistoire – semblent avoir connu une très lente mise en application : les standards web. L'histoire des jeunes années du Web et de ce que l'on a nommé « guerre des navigateurs » facilite la compréhension de cette difficile approbation des différentes normes.

En 1994, lorsque le consortium W3C, ayant pour vocation de rendre le Web universel, naît sous l'impulsion de Tim Berners-Lee, le langage HTML se limite alors à des hyperliens, des titres, sous-titres, listes et du texte brut.

Cette même année, le marché des navigateurs est complètement monopolisé par Netscape, issu du développement de NSCA Mosaic. Ce n'est que l'année suivante que Microsoft lance Internet Explorer, après une tentative avortée de concurrencer purement et simplement Internet via un réseau parallèle nommé MSN (Microsoft Network). À cette période, Netscape Navigator, payant, fédère plus de 80 % des utilisateurs.

En quelques années, une véritable bataille économique est lancée entre les deux ténors Netscape et Microsoft, chacun inventant à tour de bras de nouveaux éléments HTML, voire de nouvelles technologies. Ainsi, Netscape (figure 1-1) propose les éléments de formulaires, les balises `<img>`, `<font>`, `<center>`, `<blink>` et `<layer>`, mais aussi les cadres (*frames*) et un concept propriétaire de feuilles de styles (JSSS – *JavaScript-based Style Sheet*) entre 1994 et 1996. La première version de JavaScript émane également des laboratoires de Netscape. Internet Explorer introduit quant à lui les éléments `<table>`, `<marquee>` et diverses techniques pour rendre les pages plus dynamiques (VBScript, ActiveX, filtres CSS).

Figure 1-1

Aperçu du site Wikipédia sur le navigateur Netscape 4.78

Il va sans dire que l'adoption de chaque technologie concurrente ne se fait ni sans heurts ni sans réticences. Les premières versions d'Internet Explorer ne prennent pas en charge JavaScript. Pire, Microsoft développe et intègre son propre langage de script : JScript.

Ce n'est pas plus glorieux du côté des feuilles de styles, car Netscape se refuse longtemps à employer les CSS normalisées par le W3C (et intégrées sur Internet Explorer 3) au profit de ses styles propriétaires. En définitive, Netscape Navigator 4 prend en charge deux versions de styles conjointes : JSSS et CSS, avec une implémentation assez bâclée. Ce n'est finalement qu'avec la version Netscape 6, basée sur un nouveau moteur (Gecko), que la prise en charge des styles CSS 1 est véritablement acquise.

C'est en plein cœur de cette guerre de monopoles et de technologies propriétaires que le consortium W3C tente de réguler le Web, son usage et sa compatibilité. Les différentes versions de HTML et CSS se succèdent en intégrant au fur et à mesure certaines inventions de chacune des forces en présence.

Aux alentours de l'an 2000, après le rachat de Netscape par AOL, Internet Explorer domine largement le marché des navigateurs, selon des estimations dépassant les 95 % d'utilisateurs. Cela entraîne, de fait (par la désaffection de Microsoft pour ce marché conquis), la fin des

innovations dans ce navigateur : il n'y a pas eu de nouvelle version d'Internet Explorer entre 2001 (IE6) et 2006 (IE7).

Plutôt que son monopole, c'est principalement la stagnation d'Internet Explorer qui est rapidement décriée par de nombreux concepteurs web. L'avènement et le dynamisme des navigateurs dits « alternatifs » (Mozilla Firefox, Apple Safari, Opera, puis Google Chrome) engendrent des mouvements de contestation et une renonciation progressive au navigateur dominant. En 2010, certaines statistiques montrent que la version 3.6 du navigateur Firefox est la plus utilisée en Europe, devant Internet Explorer 8.

Cela va-t-il nous mener vers une deuxième guerre des navigateurs ?

ALLER PLUS LOIN **Comment tout a commencé ?**

Toutes les sources s'accordent à penser que l'histoire du Web commence en 1989, lorsque Tim Berners-Lee rédige un document nommé *Hypertext and CERN* (ce document peut être visionné à l'adresse ci-après). Le premier outil ressemblant à un navigateur et portant le nom de « WorldWideWeb » est conçu en 1990 et le premier serveur externe à l'Europe est configuré à l'Université de Stanford en 1992.

▸ http://www.w3.org/History/1989/proposal.html

L'année suivante, en juin 1993, nous dénombrions… 130 sites web dans le monde !

Le W3C a publié en 2004 un diaporama en ligne retraçant l'historique d'Internet et du consortium à partir des années… 1945. Ce document est consultable à l'adresse :

▸ http://www.w3.org/2004/Talks/w3c10-HowItAllStarted/

Où en est-on aujourd'hui ?

Les usages évoluent

Beaucoup d'encre a coulé (et beaucoup d'ouvrages sur CSS sont parus) depuis la publication en 2005 de mon livre *CSS2 : pratique du design web* : non seulement les spécifications ont évolué, mais les divers navigateurs se sont approprié, au fil du temps, les propriétés avant-gardistes d'hier, en attendant d'assimiler les plus actuelles.

En relisant aujourd'hui mon ouvrage, je me rends compte que de nombreuses barrières ont été levées et de nombreux progrès réalisés. Même si le discours est toujours d'actualité, l'avancée des techniques d'intégration et des méthodes d'apprentissage a permis à notre génération de webdesigners de passer un nouveau cap prépondérant dans le monde de la conception de sites web.

Le positionnement à l'aide des styles CSS (par opposition à la mise en page basée sur les tableaux HTML) est devenu plus robuste et surtout mieux assimilé par les designers. Les techniques et les méthodes d'application ont évolué en parallèle : la nouvelle génération de concepteurs web délaisse dorénavant les notions de tableaux de mise en page, de `spacer.gif`, de *layer* et autres *frames* au profit d'une meilleure séparation entre le fond (HTML) et la forme (CSS).

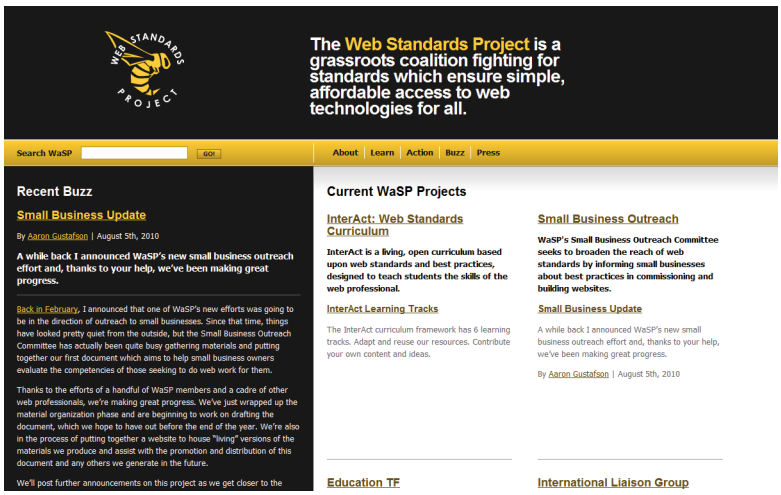
Ces nouveaux comportements s'expliquent principalement par le fait que les générations de navigateurs de l'an 2000, devancés par Internet Explorer 6 (premier navigateur à reconnaître

partiellement CSS 2) ont démontré qu'il était possible de concevoir des mises en page web à la fois esthétiques et complexes, tout en séparant la structure de la présentation et en positionnant les divers éléments via les feuilles de styles CSS.

Le début des années 2000 voit naître des mouvements tendant à prêcher la « bonne parole » des CSS (on parle même « d'évangélisateurs »), le plus médiatique étant le Web Standards Project fondé par Jeffrey Zeldman (figure 1-2). En France, c'est le collectif Openweb (co-fondé, entre autres, par Tristan Nitot, actuellement président de Mozilla-Europe) qui se charge de promouvoir les normes du W3C et plus particulièrement le positionnement CSS dès 2002.

Figure 1-2

Le site du Web Standards Project



Les normes évoluent

Les premières versions des spécifications HTML et CSS évoluent très vite : trois petites années à peine sont nécessaires pour passer de CSS 1 au premier brouillon de CSS 3. Il en va de même pour HTML, dont les versions se succèdent à bon rythme, suivant de près la guerre des navigateurs.

Entre 1998 et 2000, le monopole grandissant d'Internet Explorer s'accompagne d'une période de ralentissement général. Tous les acteurs du Web marquent le pas et font le point sur cette révolution fulgurante. La première « bulle Internet » explose et de nombreuses start-ups mettent la clé sous la porte.

Le W3C s'interroge quant à l'avenir de la Toile et à la pertinence des différents langages. Il officialise tour à tour les versions HTML 4, HTML 4.01, XHTML 1.0 et XHTML 1.1, puis s'embourbe dans de longues expectatives et d'imposants projets stationnaires.

La norme XHTML 1, une reformulation XML du langage HTML, naît en 1998 sans vraiment faire l'unanimité, en raison de sa complexité tentaculaire et de son champ de couverture

allant bien au-delà de l'usage web. Les divers acteurs du marché rechignent à l'intégrer (Internet Explorer, jusqu'à sa version 8 incluse, ne le prend d'ailleurs toujours pas véritablement en charge). Cependant, l'un des avantages de XHTML va séduire la nouvelle génération de créateurs de pages web et les formateurs : sa grande rigueur par rapport à HTML.

Tandis que tout porte à croire que la norme XHTML va devenir le futur standard du langage de conception web, un tournant s'opère peu après les années 2004. Un nouveau groupe de travail « dissident », le WHATWG (*Web Hypertext Application Technology Working Group*), se forme parallèlement au W3C pour redynamiser l'évolution de HTML.

Le W3C annonce en 2007 que le groupe de travail WHAT intègre officiellement le consortium afin de créer HTML 5 et, deux ans plus tard, que les travaux en cours sur la norme XHTML 2 sont définitivement suspendus. Le dernier né de la lignée XHTML, bien que demeurant tout à fait standard, n'aura pas de successeur et le prochain langage du Web sera bel et bien HTML 5.

Les navigateurs évoluent

Du côté des navigateurs web également, on observe de nombreux remous depuis la publication de mon premier livre : Microsoft sort de son mutisme et lance IE7 en 2006, puis IE8 en 2009, tout en annonçant une version IE9 encore plus au fait des normes. Trois versions en quelques années, alors que le géant nous avait habitué à IE6 durant plus de six ans !

À partir de 2005, on observe une baisse régulière de l'usage d'Internet Explorer au profit des navigateurs dits « alternatifs », dont principalement Mozilla Firefox, qui jusqu'alors se contentaient de parts de marché négligeables.

Fin 2008, Google surprend en publiant son propre navigateur : Chrome (figure 1-3). Basé sur le même moteur que Safari, Chrome est à la fois rapide et très conforme aux standards récents. Il est vite adopté (surtout chez les « geeks ») grâce à une mise en avant par Google et s'installe dès la première année sur la troisième place d'un podium très disputé.

OUTIL Un indice de conformité aux standards : Acid Test

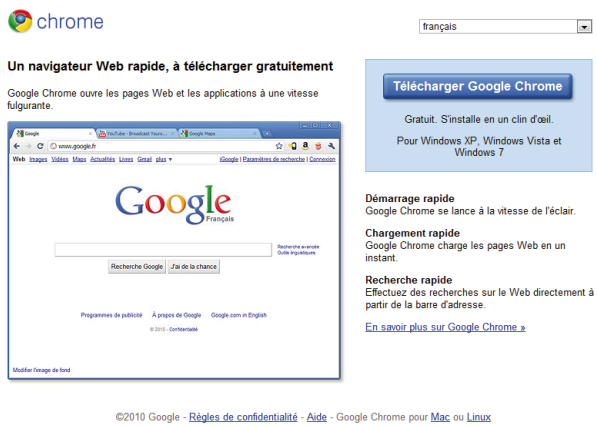
Acid1, publié en 1998, teste la conformité de la mise en œuvre de certaines fonctionnalités du niveau 1 des feuilles de styles en cascade (CSS). Le principe du test a évolué avec les standards du Web, donnant les versions *Acid2* et *Acid3* (*Acid4* est en préparation).

Pour passer le test, un navigateur doit, avec ses réglages par défaut, afficher le rendu fluide d'une animation dont l'image finale doit correspondre exactement à une image de référence, avec un score de 100/100. Pour cela, le navigateur doit implémenter correctement certains aspects du DOM2, d'ECMAScript, des CSS, du SVG, du XML et des URI. Le test *Acid3* n'est donc pas un test de conformité globale à ces spécifications, comme le sont en revanche les test suites du W3C.

Source : Wikipédia.

Figure 1-3

Page de téléchargement de Chrome



À l'heure actuelle, nous distinguons quatre groupes de navigateurs web, classés selon leur moteur de rendu :

- ceux basés sur le moteur de rendu Trident (dont Internet Explorer et Maxthon) ;
- ceux basés sur le moteur de rendu Gecko (dont Mozilla Firefox, Seamonkey, K-Meleon, Camino, Epiphany, Flock) ;
- ceux basés sur Presto (Opera, Opera Mobile, Opera Mini, Opera Wii, Opera pour Nintendo DS) ;
- et ceux basés sur WebKit (Apple Safari, Apple Safari iPhone, Google Chrome, Adobe AIR, Palm), KHTML (Konqueror).

En début d'année 2011, selon une moyenne de diverses statistiques mondiales, cinq navigateurs se partagent l'essentiel du marché : Internet Explorer (environ 50 %), Mozilla Firefox (30 %), Google Chrome (15 %), Apple Safari (5 %) et Opera (2 %) (source : http://fr.wikipedia.org/wiki/Navigateur_web).

Le « cas » Internet Explorer

Les mises à jour d'Internet Explorer constituent une étude intéressante dans la mesure où nous avons l'opportunité d'être les témoins d'un cas de figure inédit : quatre générations du même navigateur vont coexister, pour le plus grand désarroi des concepteurs web.

Internet Explorer 6

À sa sortie en 2001, Internet Explorer 6 disposait de nombreux atouts en sa faveur : plus rapide, plus sûr et plus conforme aux standards que ses concurrents directs. Il marque en outre la supériorité écrasante de Microsoft sur tous ses rivaux de l'époque et scelle la fin de la première « guerre des navigateurs » puisque IE6 peut se targuer d'avoir été utilisé par plus de 95 % des internautes autour des années 2003.

Ce monument de l'histoire du Web accompagnera une génération de concepteurs web pendant plus de six années durant lesquelles l'immobilisme de Microsoft causera de sérieux dégâts à l'évolution du Web et de ses usages. Parmi ces dommages, comptons les nombreuses erreurs de rendu CSS (dont une liste est tenue à jour sur le site *Position Is Everything* : http://www.positioniseverything.net/explorer.html), ses défauts d'interprétation des normes, ses technologies ou éléments propriétaires et ses failles de sécurité accumulées au fil des ans.

En 2010, IE6 représente encore une part non négligeable dans le parc des navigateurs web : près de 5 % des internautes surfent avec cette version, délibérément ou non. Certaines applications industrielles lourdes et basées sur IE6 sont encore déployées chez de grosses entreprises et le coût de la migration de ces outils freine l'agonie de ce navigateur désuet.

Des sites web d'envergure tels que Google, YouTube, Amazon ou encore Yahoo! ont annoncé qu'ils cessaient dorénavant leur prise en charge et leurs développements pour cette version d'IE, trop limitée pour leurs produits d'avenir. En France, c'est le site commercial de la Redoute (www.laredoute.fr) qui lance le mouvement à la fin de l'année 2010 (figure 1-4).

HUMOUR Mort de IE6

De plus en plus de sites web humoristiques jouent avec l'âge avancé et le déclin de ce navigateur et poussent à le mettre à jour. Citons ripie6.com, ie6nomore.com, dieie6.com, ie6death.com, ie6funeral.com, stopie6.com, idroppedie6.com, etc.

Figure 1-4

La Redoute et Microsoft Internet Explorer 6

Internet Explorer 7

2006 marque le réveil du géant Microsoft dans le monde du Web et, par la même occasion, du microcosme des concepteurs et défenseurs des standards : après une hibernation de plusieurs longues années, de nouvelles perspectives s'ouvrent enfin à nouveau avec la sortie d'Internet Explorer 7 (figure 1-5).

À peine publié, IE7 essuie aussitôt les premières critiques : huit ans après la finalisation de la version CSS 2, le navigateur ne prend toujours pas complètement en charge cette norme, contrairement à tous ses camarades alternatifs, et présente un certain nombre d'autres lacunes notables.

Toujours est-il que la version 7 d'Internet Explorer corrige une bonne quantité d'erreurs de son prédécesseur et apporte des améliorations significatives :

- Plusieurs sélecteurs CSS 2.1 sont désormais reconnus, tels que le sélecteur d'adjacence (symbole `+`), le sélecteur d'enfant (symbole `>`) et le sélecteur d'attribut (`[attr]`).
- De nouvelles propriétés sont prises en compte, entre autres `min-width`, `min-height`, `max-width` et `max-height`.
- Le pseudo-élément `:first-child` est dorénavant pris en charge et `:hover` est enfin interprété sur tous les éléments et non plus uniquement sur les éléments hypertextes.
- La position fixée (`position: fixed`) est interprétée.
- Enfin, la transparence « alpha » (niveaux d'opacité) du format d'image PNG 24 est reconnue.

Pour résumer, les principaux développements CSS d'IE7 portent sur les sélecteurs CSS 2. En revanche, d'autres domaines très attendus sont laissés pour compte : peu de nouvelles propriétés reconnues et, surtout, aucune mise en œuvre des séduisantes possibilités de positionnement et de rendu proposées par CSS 2.1.

Même si la venue au monde d'IE7 témoigne de la bonne volonté de Microsoft de se rapprocher à nouveau des standards, les concepteurs web restent frustrés.

Figure 1-5

Microsoft Internet Explorer 7

Internet Explorer 8

La huitième variante du navigateur phare de Microsoft se dévoile au début de l'année 2009 et comble les défaillances critiquées de ses ancêtres sur au moins un point essentiel : IE8 comprend (enfin) toute la spécification CSS 2.1 (figure 1-6).

Cela pourrait presque suffire à le résumer, mais le fait d'avoir eu à négocier si longtemps avec des navigateurs défectueux fait que les concepteurs web ne savent plus exactement quels sont les apports de CSS 2.1 à présent reconnus par Internet Explorer.

Voici donc une liste non exhaustive mais néanmoins attrayante des corrections d'IE8 :

- Le mécanisme *HasLayout* a été abandonné (nous le verrons plus en détail dans le chapitre 6 dédié à la résolution des bogues) et les flottants ont été améliorés.
- Toutes les valeurs de la propriété `display` (dont `table`, `table-cell` et `table-row`) sont prises en compte et la valeur `inline-block` est corrigée.
- La fusion de marges (*margin collapsing*) est désormais conforme aux spécifications CSS 2.1.
- Les pseudo-éléments `:before` et `:after`, qui permettent d'ajouter du contenu avant et après un élément, sont reconnus.
- Les pseudo-classes `:focus` et `:lang` sont comprises.
- IE8 prend en compte toutes les valeurs CSS 2.1 pour `list-style-type`, `background-position`, `font-weight`, `white-space`, `word-spacing`, `border-collapse`, `border-style` et `empty-cells`.
- La propriété `outline`, qui permet d'ajouter une bordure sans affecter la taille de l'élément, avec ses dérivés `outline-color`, `outline-style` et `outline-width`, est également reconnue.
- IE8 améliore sa prise en charge du média imprimante, avec `page-break-inside` (comment un saut de page doit se comporter s'il survient dans la boîte de rendu d'un élément), `widows`, `orphans`, `@page` avec les sélecteurs `:first`, `:right`, `:left` et les valeurs `avoid`, `left` et `right` pour `page-break-before` et `page-break-after`.
- Quelques maigres propriétés CSS 3 : `border-spacing`, `text-overflow`, `box-sizing`, `word-wrap`, sans oublier `font-face` (reconnue depuis IE5.5) sont dorénavant mieux prises en compte.

Figure 1-6

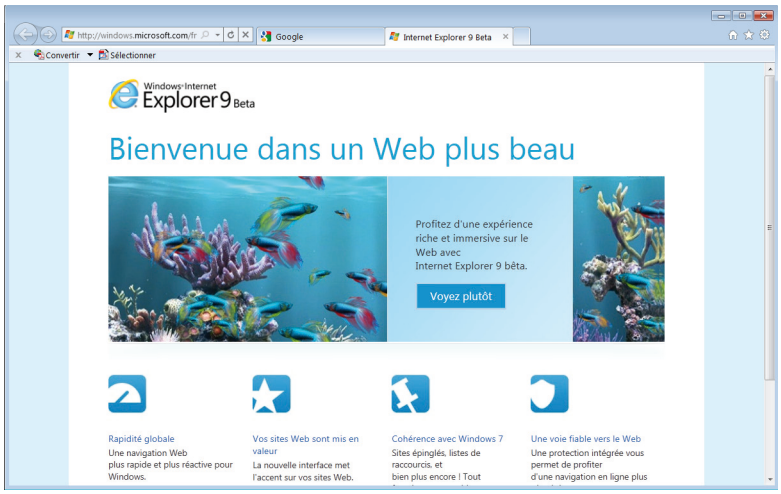
Microsoft Internet Explorer 8

Internet Explorer 9

À l'heure de la publication de cet ouvrage, le très attendu IE9 (figure 1-7) n'en est encore qu'à un stade de développement bien avancé mais non finalisé (la version bêta publique est sortie en septembre 2010). De nombreuses rumeurs circulent à ce sujet depuis le début de l'année 2010, notamment via le blog officiel de Microsoft ou le compte Twitter @ie.

Figure 1-7

Microsoft Internet Explorer 9



Le géant annonce une « prise en charge de HTML 5 et de CSS 3 » que l'on ne peut malheureusement résumer pour l'instant qu'à quelques prudents préliminaires dans ces domaines. Cependant, la communication de Microsoft sur ce point est claire : le rapprochement vers les derniers standards en date est devenu une priorité manifeste.

Retenons de la version bêta de ce nouvel opus les points suivants :

- Seules quelques propriétés CSS 3 sont reconnues, mais pas des moindres : `border-radius`, `opacity`, `box-shadow`, `media queries`, ainsi que les images d'arrière-plan multiples et la gestion de la transparence avec `RGBa` et `HSLa`.
- Tous les sélecteurs CSS 3 sont pris en compte, dont les pseudo-éléments `:target`, `:enabled`, `:disabled`, `:checked`, `:not`, `:nth-child`, `:last-child`, `:empty`, etc.
- Un meilleur affichage des polices (lissage) est observé, facilitant l'usage de polices exotiques et les format TTF (*TrueType Font*) et WOFF (*Web Open Font Format*) sont reconnus.
- Le standard SVG (*Scalable Vector Graphics*), graphisme vectoriel, est pris en charge.
- Quelques éléments HTML 5 tels que `<audio>`, `<video>` et `<canvas>` sont reconnus, du moins partiellement.

À l'instar d'IE7, Internet Explorer 9 semble être plus une esquisse qu'une version véritablement mature : un effort a été réalisé sur le traitement des sélecteurs CSS 3, mais il faudra sans doute patienter jusqu'à IE10 pour disposer enfin d'un outil reconnaissant toutes les différentes propriétés CSS 3 et HTML 5 tels que le font déjà très bien tous ses concurrents.

Et en attendant, IE6 vivote encore…

Prendre en compte les anciens navigateurs ?

En 2011, la cohabitation chez Microsoft de quatre générations de navigateurs va constituer un handicap majeur chez les concepteurs web car les différences de performance, de sécurité et de conformité seront ingérables : il ne sera plus possible de concevoir un site web identique à la fois pour la version actuelle du navigateur et pour son lointain ascendant, bien rétrograde…

L'expérience n'est bien entendu pas réservée à Microsoft : tous les constructeurs de navigateurs proposent des évolutions de leurs produits et les délais de propagation parmi les utilisateurs sont rarement immédiats. De tous temps, les diverses générations d'un même navigateur ont cohabité, pour le meilleur et pour le pire…

Deux notions accompagnent ce phénomène : celle de *dégradation gracieuse* et celle de la *prise en charge progressive* (ou graduelle) des différentes versions de navigateurs.

Dégradation gracieuse

La dégradation gracieuse est une – hasardeuse – traduction de l'expression anglophone *graceful degradation* qui signifie qu'un site web doit continuer à être « opérationnel » quelle que soit l'avancée technologique employée. Dans notre cas, cela se traduit par un usage de styles CSS de telle sorte que cela ne nuise pas à la consultation des pages sur un ancien navigateur (figures 1-8 et 1-9).

> **CHOISIR Dégradation gracieuse ou amélioration progressive ?**
>
> Notez qu'à l'inverse du concept de dégradation gracieuse, le principe d'amélioration progressive (*progressive enhancement* en anglais) est une stratégie de conception web en couches successives, qui permet à chacun d'accéder au contenu et aux fonctionnalités de base d'une page web en utilisant n'importe quel navigateur, tout en offrant une version améliorée aux utilisateurs disposant de navigateurs plus récents ou plus évolués.

Entre Internet Explorer 6 qui ne reconnaît qu'une faible partie de CSS 2 et Internet Explorer 9 qui incorpore CSS 3, il faudra définir votre curseur de dégradation gracieuse selon vos besoins, ceux de vos clients et votre cible. Ne nous leurrons pas : au vu des lacunes du dinosaure IE6, votre latitude est assez faible et ne concernera qu'un assortiment de diverses améliorations cosmétiques et certainement pas des positionnements complexes ou des comportements décisifs.

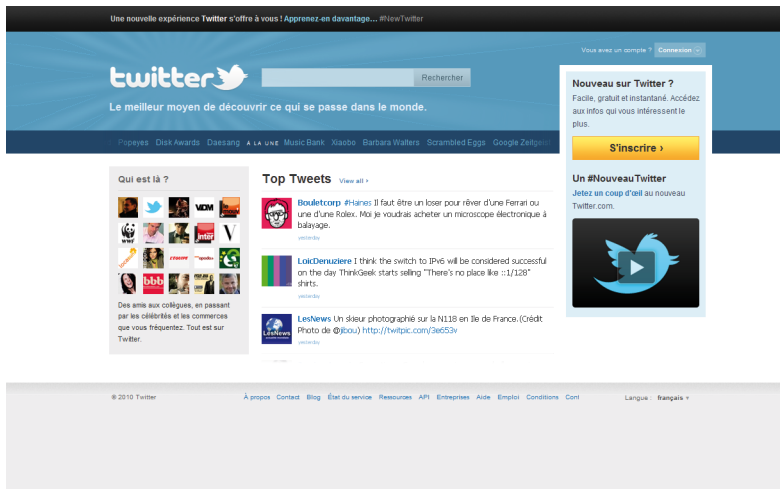
Cela permet toutefois d'opter occasionnellement pour une technologie récente telle que CSS 3 pour certaines décorations d'éléments de page (coins arrondis, ombrages, semi-transparence, transitions progressives…) qui pourraient consister en des « bonus » mérités pour les navigateurs récents. Ces ornements pourraient également être traités sans CSS 3 à l'aide de méthodes plus longues et coûteuses en performances (blocs `<div>` imbriqués, arrière-plans multiples, JavaScript…).

Figure 1-8

Twitter.com sur un navigateur conforme récent (Firefox 3.6)

Figure 1-9

Twitter.com sur Internet Explorer 8



Prise en charge progressive

L'acceptation de la notion de dégradation gracieuse favorise la planification d'une prise en charge progressive des différentes générations de navigateurs.

En amont de votre projet web, déterminez trois niveaux de prise en charge selon votre public et vos statistiques de visites :

- **Niveau 1** : prise en charge maximale. Les navigateurs dans cette catégorie doivent offrir au visiteur toutes les performances techniques, visuelles et fonctionnelles définies par le cahier des charges et la maquette graphique.

- **Niveau 2** : prise en charge dégradée. Les navigateurs de niveau 2 doivent permettre une expérience utilisateur équivalente au niveau précédent, mais qui pourra toutefois présenter des différences considérées comme négligeables (décalages minimes, arrondis, ombrages…).
- **Niveau 3** : prise en charge minimale. Le site doit être accessible, fonctionnel et agencé convenablement, mais aucun effort ne sera porté sur la compatibilité visuelle avec les niveaux précédents (allant jusqu'à l'affichage en texte brut).

Il ne vous reste plus qu'à affecter chaque version de navigateur à un niveau de prise en charge que vous aurez défini préalablement avec votre client (figure 1-10).

Figure 1-10

Tableau de prise en charge progressive

Navigateur	Internet Explorer	Mozilla (Firefox)	Apple Safari	Google Chrome	Opera
Moteur	Trident	Gecko	Webkit	Webkit	Presto
Niveau 1	8.0	3.6 3.5	5.0	7.0 6.0	10.5
Niveau 2	7.0	3.x	4.0	5.0	10.0
Niveau 3 sur demande spécifique	*6.0*	*2.x*	*3.0*	*4.0*	*9.x*

Voici l'exemple d'une telle liste de prise en charge graduelle :

- Navigateurs de niveau 1 : IE8, Firefox 3.6+, Opera 10.5+, Safari 5+, Chrome 6+ ;
- Navigateurs de niveau 2 : IE7, Firefox 3.0, Opera 9 et 10, Safari 4, Chrome 5 ;
- Navigateurs de niveau 3 : IE6, Firefox 2.0, Opera 8, Safari 3, Chrome 4.

Dans les faits, nous constaterons tout au long de ce livre qu'opter pour des « techniques avancées » en CSS nécessitera de se poser constamment la question de la dégradation gracieuse, de la prise en charge progressive et des alternatives pour le navigateur de Microsoft, actuellement unique frein vers des designs et fonctionnalités évolués.

Première partie

Tirer le meilleur de CSS

2

Exploiter les possibilités de CSS 2.1

Ce chapitre est consacré aux propriétés et techniques méconnues de la norme CSS 2.1 : la généalogie, la priorité, les pseudo-éléments, les sélecteurs avancés (attribut, adjacence, parenté), la création de contenu, les règles @ et les microformats.

Tandis que le mot-clé « CSS3 » compte parmi les plus recherchés sur les moteurs et les outils sociaux tels que Twitter, il est bon de rappeler que durant de très longues années, la norme en vigueur demeure CSS 2.0, qui date pourtant de 1998.

La variante CSS 2.1 vient à peine d'être officiellement validée, mais elle bénéficie du statut de « recommandation officielle » depuis la fin de l'année 2010 !

Fort heureusement, les constructeurs de navigateurs et les concepteurs web ne sont pas tenus d'attendre l'adoubement confirmé d'un standard pour le prendre en charge. Ainsi tous les navigateurs actuels (à partir d'Internet Explorer 8) prennent en charge les propriétés et sélecteurs CSS 2.1, voire diverses parties de niveau CSS 3, encore au statut de brouillon.

Terminologie et syntaxe de base

Chaque évolution des CSS repose sur les mêmes fondements et la même terminologie que la version antérieure, notamment pour des raisons de rétro-compatibilité et de meilleure intégration des nouveaux concepts.

Malgré cette constance dans les spécifications, je rencontre continuellement des termes non adéquats lorsque j'arpente notre forum de discussion sur Alsacréations (forum.alsacreations.com) : c'est ainsi que je côtoie des « attributs CSS », des « balises CSS » ou encore des « fonctions CSS ».

Ces minimes fautes de langage sont le signe que beaucoup apprennent la mise en page CSS sur le tas, au gré de recherches et flâneries sur le Web et à l'aide de sources d'information variées, tant du point de vue de leur complexité que de leur exactitude. On ne peut pas nous en vouloir : les normes HTML et CSS sont des documents abscons et ennuyeux pour le commun des mortels, auquel ils ne sont d'ailleurs pas destinés au départ.

Dans le cadre de cet ouvrage, j'ai par conséquent pris l'initiative de revenir rapidement sur la terminologie générale et la syntaxe des styles CSS, non pas pour vous harceler ou pour fanfaronner, mais surtout pour être bien certain que nous soyons parfaitement en adéquation lorsque je désignerai tel ou tel élément au cours des chapitres suivants.

Commentaire

Les commentaires en CSS sont des instructions facultatives commençant par les caractères `/*` et se terminant par `*/` (figure 2-1). On peut les placer partout entre les règles, voire au sein des règles (mais pas entre une propriété et sa valeur), leur contenu n'ayant aucune influence sur le rendu. On ne peut pas les imbriquer.

Leur usage est principalement informationnel, dans le cadre de projets en commun et dans l'optique de justifier tel ou tel choix de déclaration CSS, ou encore de délimiter les différentes parties de la feuille de styles.

Figure 2-1

Illustration d'un commentaire CSS

```
header nav a:hover, header nav a:focus {
  background: #E4E9ED;
  background: rgba(255,255,255,0.3);
  text-decoration: none;
  -moz-border-radius: 10px;
  -wekbit-border-radius: 10px;
  border-radius: 10px;
  -moz-outline-radius: 10px;  /* a11y : on arrondit le focus, si un focus carré pose problème */
  -webkit-outline-radius: 10px;
  outline-radius: 10px;
  outline: none;
}
```

Propriété, valeur et déclaration

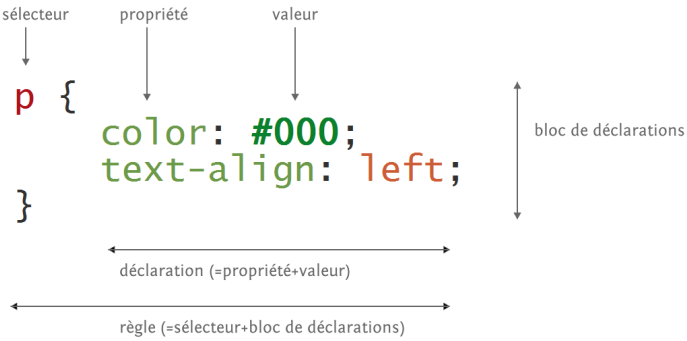
Une *propriété* est une syntaxe définie explicitement par les spécifications CSS et définissant la mise en forme de l'élément auquel elle s'applique (figure 2-2). On peut regrouper les propriétés par type (typographie, positionnement, modèle de boîte), mais aussi par type de média compatible (écran, imprimante, mobile, auditif, projection).

La version 1 de CSS comportait un total de 53 propriétés, tous types confondus. CSS 2.1 en propose 115 et CSS 3 en admet 246 à l'heure actuelle (source : http://meiert.com/en/indices/css-properties/).

Les normes CSS spécifient quelles sont les *valeurs* autorisées pour chaque propriété. Ces valeurs peuvent être construites à partir d'identificateurs, de chaînes de caractères, de nombres, de longueurs, de pourcentages, d'adresses URI, de couleurs, d'angles, de durées, etc.

Figure 2-2

Visuel de la syntaxe d'une règle CSS



Une propriété dont la syntaxe n'est pas homologuée ou dont la valeur n'est pas valide est a priori ignorée par les navigateurs. Cependant, certains d'entre eux offrent en réalité une tolérance qui peut être exploitée sous forme de *hack*, c'est-à-dire un détournement volontaire d'une propriété ou d'une valeur pour cibler une version de navigateur déficiente et lui appliquer des styles particuliers.

Le terme *déclaration* désigne un ensemble formé par la propriété et sa valeur associée, par exemple `font-style: italic`. On peut regrouper plusieurs déclarations en les séparant par des points-virgules, par exemple `font-style: italic; background-color: #ffc`.

> **BONNE PRATIQUE** **Dernière déclaration**
>
> La dernière déclaration d'un bloc ne nécessite pas de séparateur (`;`), mais pour des raisons de rigueur, je vous recommande de le signaler, ne serait-ce que pour éviter une erreur dans le cas où vous ajouteriez une déclaration par la suite.

Sélecteur

Le terme de sélecteur désigne la partie précédant le bloc de déclaration, c'est-à-dire à gauche de l'accolade ouvrante d'une règle CSS. Un sélecteur doit d'ailleurs toujours être accompagné d'un bloc de déclaration.

Le sélecteur indique l'élément sur lequel vont s'appliquer les déclarations CSS présentes dans le bloc qui lui est associé. Il peut revêtir plusieurs formes : sélecteur d'élément (exemple : `p`), sélecteur de classe (exemple : `.fruit`), sélecteur d'identifiant (exemple : `#kiwi`), sélecteur d'attribut (ex : `[href]`), etc.

Plusieurs sélecteurs peuvent précéder un bloc de déclaration. Il suffit de les séparer par une virgule. Le bloc sera alors appliqué à chacun des sélecteurs, comme l'illustre l'exemple suivant :

```
h1, h2, h3 { font-weight: bold;}.
```

BONNE PRATIQUE Attention à la syntaxe !

Un sélecteur ne peut être composé que de caractères alphanumériques, de tirets (-) et de caractères *souligné* (_). Les caractères spéciaux ne sont pas valides et – excepté en HTML 5 – il n'est pas autorisé de commencer un sélecteur par un chiffre ou un tiret.

Sélecteur de classe

Une *classe* est un nom que l'on choisit librement (en se limitant aux caractères alphanumériques classiques) et dont on baptise les éléments concernés via l'attribut HTML `class`. Un sélecteur de classe reprend ce nom en le préfixant d'un point (par exemple : `.error`, `.bloc`, etc.) :

Partie HTML

```
<span class="error">formulaire incomplet</span>
```

Partie CSS

```
.error {color: red;}
```

Sélecteur d'identifiant

Un *identifiant* (ou `id`) est lui aussi un nom choisi librement (en se limitant aux caractères alphanumériques classiques). Il se distingue de la classe en ce qu'il ne peut porter au plus que sur un objet du document.

Les sélecteurs CSS s'y réfèrent par l'emploi d'un caractère dièse (`#`) suivi de ce nom (exemples : `#header`, `#nav`, `#contact`, etc.).

Partie HTML

```
<ul id="nav">...</ul>
```

Partie CSS

```
#nav {margin: 0; padding: 0;}
```

L'identifiant désigne un élément unique dans le document (par exemple, `#kiwi`), contrairement aux classes pouvant symboliser plusieurs types d'éléments ou des groupes d'éléments (par exemple, `.fruit`) : l'élément dont l'`id` est `kiwi` sera unique, mais plusieurs éléments peuvent appartenir à la classe `fruit`.

Prenez donc le réflexe d'attribuer un identifiant aux objets uniques de votre code ou aux grands blocs qui structurent la page. On trouve souvent un bloc d'en-tête (`#header`), le pied de page (`#footer`), la partie latérale (`#aside`), la partie principale (`#main`), le menu de navigation (`#nav`), etc. Les éléments susceptibles d'apparaître plusieurs fois dans un document HTML seront quant à eux qualifiés à l'aide de classes.

Règle et bloc de déclaration

Une règle CSS se compose d'un sélecteur suivi de son bloc de déclaration, ce dernier débutant par une accolade gauche ({) et se terminant par l'accolade droite (}) correspondante et contenant une ou plusieurs déclarations.

Au final, une feuille de styles est un fichier de type textuel contenant l'ensemble des règles CSS.

Pseudo-classe et pseudo-élément

Dans l'optique de mettre en forme des éléments selon un état contextuel (survol, premier enfant, première ligne…) qui ne serait pas défini dans l'arbre du document, CSS introduit les concepts de pseudo-classe et de pseudo-élément.

La différence entre les pseudo-classes et les pseudo-éléments étant plutôt ténue, d'autant plus qu'un certain nombre d'exceptions en compliquent encore la compréhension, je vais par facilité regrouper ces deux termes sous la même dénomination de « pseudo-élément » tout au long de cet ouvrage, sauf dans les cas où la distinction sera pertinente.

EXEMPLES Pseudo-classes et pseudo-éléments

Parmi les pseudo-classes les plus couramment usitées, retenons les différents états des liens hypertextes (`:link`, `:visited`, `:hover`, `:focus` et `:active`) et le premier enfant d'un élément (`:first-child`). Du côté des pseudo-éléments, mentionnons plus particulièrement la première lettre d'un bloc (`:first-letter`), sa première ligne (`:first-line`) ou le contenu créé avant (`:before`) ou après (`:after`) un élément.

L'exception :visited

Les dernières générations de navigateurs (à partir de Firefox 3.6, Chrome 5 et Safari 4.0.5) viennent subitement de restreindre considérablement l'éventail des propriétés CSS applicables à la pseudo-classe `:visited`, vieille comme le Web et désignant un lien que l'on a déjà suivi. Les seules propriétés dorénavant tolérées sur cet élément se limitent à la définition des couleurs (`color`, `background-color`, `border-color`, `outline-color`, `column-rule-color`, `fill` et `stroke`).

Cette restriction imposée par les navigateurs est due à une vulnérabilité de `:visited` découverte très récemment et permettant d'exploiter l'historique de navigation d'un visiteur. Le détail de cette faille est expliqué sur le blog du développeur principal de Mozilla, David Baron, à l'adresse :

▸ http://dbaron.org/mozilla/visited-privacy

En traitant ces données à l'aide des technologies dynamiques actuelles, il devient possible de récupérer un grand nombre d'informations véhiculées via les URL visitées et de traiter un vaste champ de données personnelles telles que :

- les visites de liens dits « sensibles » (banques, sites politiques, religieux ou pour adultes) ;

- les recherches effectuées sur les moteurs (exemple : http://www.google.fr/search?q=ma+recherche) ;
- votre réseau social dans sa globalité (Qui sont vos contacts sur Facebook, LinkedIn ou Twitter ? Où habitent-ils ? Quel est leur profil ?) ;
- des informations relatives à votre vie privée, votre nom, vos coordonnées, éventuellement votre état de santé, votre orientation politique, etc.

Si vous n'avez pas la chance de disposer d'une très récente version de navigateur, sachez que vous êtes exposé à ces risques de confidentialité.

EN SAVOIR PLUS **Faille de confidentialité**

Un site Internet d'information anglophone sur ce sujet vous permettra d'en savoir plus à propos de cette faille et son exploitation en pratique :

▸ http://whattheinternetknowsaboutyou.com

Généalogie

Les différents éléments composant un fichier HTML (titres, paragraphes, listes, liens, images…) s'organisent sous la forme d'un « arbre généalogique », que l'on nomme également « arbre du document » (ou DOM pour *Document Object Model*). Cet arbre généalogique est composé de fratries et de degrés de parenté qui sont exploités par les sélecteurs CSS pour cibler les différents éléments du document.

Ancêtre, parent, frère

Choisissons arbitrairement un élément situé au sein de l'arborescence. Cet élément, que nous nommerons X pour préserver son anonymat, est susceptible d'être entouré d'éléments ancêtres, descendants, parents, enfants et frères.

Au sein de cette arborescence, le terme d'*ancêtre* désigne tout élément situé dans la même branche que l'élément X, mais à un niveau supérieur dans la hiérarchie. Le *parent* est un élément unique qui fait référence à l'élément placé directement au dessus de X.

À l'inverse, le concept de *descendant* fait référence à tout élément qui est situé dans la même branche que X, mais à un niveau plus bas dans la hiérarchie. Les *enfants* sont les éléments placés directement au dessous de X.

Tous les éléments situés au même niveau que X et partageant le même parent sont appelés *frères*.

Influence sur les sélecteurs

Pour des raisons de compatibilité avec les navigateurs, nous avions jusque-là coutume de n'employer que des sélecteurs très basiques se limitant généralement aux noms de classes, d'identifiants ou aux ancêtres et descendants.

Cependant, CSS 2.1 et CSS 3 proposent un vaste panel de sélecteurs avancés ciblant bien plus spécifiquement les éléments selon leur hiérarchie, ce qui évite d'attribuer à outrance des noms de classes sur de nombreux éléments. Grâce à ces « nouveaux » sélecteurs enfin utilisables depuis Internet Explorer 7, nous allons par exemple pouvoir atteindre les éléments frères ou les enfants directs.

Priorité des sélecteurs

Il est reconnu que lorsque deux règles CSS différentes ciblent le même élément, c'est la dernière déclarée (la plus basse dans le code source) qui s'applique et qui écrase la précédente. En pratique, cette loi ne vaut que dans le cas où les deux sélecteurs incriminés sont exactement du même type et déclarés de la même manière, car il existe une priorité entre les différents types de sélecteurs.

Mode de déclaration

Je ne vais pas vous apprendre qu'il existe plusieurs façons d'appliquer des styles CSS à un élément HTML. En revanche, vous ne saviez peut-être pas que chaque méthode a une priorité d'application différente :

1. Style en ligne : on applique le style directement au sein de la balise HTML, via l'attribut `style=`. Ce mode est prioritaire sur tous les autres.
2. Déclaration dans l'en-tête du document : on emploie la balise HTML `<style>` que l'on place au sein de l'élément `<head>` de la page HTML. Ce mode est prioritaire sur les feuilles de styles externes.
3. Feuille de styles auteur : la mise en forme est complètement externalisée dans un fichier de styles CSS lié à l'aide de `<link>` ou `@import`.

> **À RETENIR Feuille de styles utilisateur**
>
> Aux styles définis par l'auteur du site web s'ajoute la possibilité, pour chaque internaute, de créer des styles personnalisés dans la feuille de styles « utilisateur », qui toutefois demeurera en retrait par rapport à celle proposée par le concepteur du site.

Poids des sélecteurs

Au sein du même mode de déclaration, les spécifications CSS proposent une classification des sélecteurs selon un barème de poids sous forme de notation à quatre chiffres :

1. Poids « a » : règle CSS déclarée à l'aide de l'attribut HTML `style=` au sein de l'élément.
2. Poids « b » : sélecteur d'identifiant (`id`).
3. Poids « c » : sélecteur de classe, d'attribut (`[]`) ou de pseudo-classe (`:hover`, `:focus`, `:first-child`).

4. Poids « d » : sélecteur d'élément (`p`, `div`, `a`,…) ou de pseudo-élément (`:first-letter`, `:before`, `:after`, `:first-line`).
5. Poids nul : sélecteur joker (`*`), de parenté (`>`) ou d'adjacence (`+`).

L'explication de ce barème est simple : une déclaration qui s'applique à un sélecteur d'`id` (exemple : `#kiwi {color: green;}`) aura toujours un poids supérieur à la même déclaration appliquée à un sélecteur de classe (exemple : `.kiwi {color: lime;}`), qui elle-même sera prioritaire sur une déclaration visant un élément en particulier (ex : `h1 {color: red;}`).

En cas de conflit, il est alors parfois nécessaire « d'ajouter du poids » à votre sélecteur en y incluant un type de sélecteur supplémentaire.

Prenons un exemple concret en observant la priorité des styles dans le cas du lien hypertexte du code suivant.

Partie HTML

```
<div id="warning">
  <a class="error" href="url">source de l'erreur</a>
</div>
```

Supposons que les règles CSS appliquées soient les suivantes.

Partie CSS

```
a {color: green;} /* 1 sélecteur d'élément (poids « d ») */
#warning a {color: blue;} /* sélecteur d'identifiant (poids « b ») + 1 sélecteur de
➥ poids « d » */
a.error {color: red;} /* 1 sélecteur de classe (poids « c ») + 1 sélecteur de poids
➥ « d » */
```

La couleur du lien indiquant une erreur sera bleue car la règle appliquée est `#warning a {color: blue;}`. Bien que la règle `a.error {color: red;}` lui succède dans l'ordre du code CSS, le sélecteur d'identifiant demeure prioritaire.

Si je souhaite que la couleur du lien soit rouge, je dois ajouter un poids au moins équivalent à celui du sélecteur `a.error`. Concrètement, j'écrirai `#warning a.error`.

> **BONNE PRATIQUE** **Sélecteurs « à rallonge »**
>
> Attention à ne pas verser dans l'extrême dans l'apport de poids, afin de ne pas vous retrouver avec des accumulations de multiples types de sélecteurs. Identifiez au plus court les éléments à mettre en forme, sans sélecteurs intermédiaires inutiles, et ce, en amont de votre projet de conception CSS.

!important

La déclaration `!important` a été introduite par CSS dans le but d'outrepasser volontairement la priorité conférée par défaut aux modes de déclaration que sont les feuilles de styles auteur et utilisateur. Dans la pratique, une déclaration suivie du mot-clé `!important` devient préférentielle,

quel que soit le poids du sélecteur qui l'accompagne : les styles marqués ainsi écrasent d'office tous les styles similaires antérieurs.

Pour reprendre notre exemple précédent, nous pouvons forcer l'application de la couleur rouge à nos liens de classe `error` en agissant de la sorte :

```
a.error {color: red !important;}
```

Quelle que soit la place d'apparition de cette règle dans la feuille de styles et quel que soit le poids de son sélecteur, les styles seront affectés en priorité… sauf si une autre règle équivalente revendique elle aussi sa préséance à l'aide d'une déclaration `!important`, auquel cas, la règle habituelle du poids des sélecteurs s'applique.

APPLICATION PRATIQUE **!important**

L'usage du mot-clé `!important` est courant dans les scripts JavaScript, Ajax ou jQuery que vous pouvez rencontrer sur Internet, tout simplement parce que ces scripts doivent pouvoir s'appliquer sur n'importe quelle page web sans en connaître la teneur et les styles déjà présents. Cependant, je ne vous recommande guère de l'employer sans en maîtriser toutes les conséquences : sa présence est généralement un indicateur d'une feuille de styles brouillonne avec une gestion hasardeuse des poids des sélecteurs.

Sélecteurs et pseudo-éléments CSS 2.1

Les versions CSS 2 et CSS 2.1 élaborent de nouveaux types, mal connus, de sélecteurs et de pseudo-éléments. La grande majorité de ces concepts est acquise par tous les navigateurs actuels et, du côté de Microsoft, depuis la version Internet Explorer 7 ; autant dire que nous pouvons d'ores et déjà les manipuler, à condition de maîtriser un tant soit peu leur alternative sur le vénérable ancêtre IE6. Seules exceptions de taille dans ce tableau idyllique, les pseudo-éléments `:before` et `:after` autorisant la création de contenu via CSS ne sont pris en compte qu'à partir d'Internet Explorer 8.

Sélecteur d'enfant

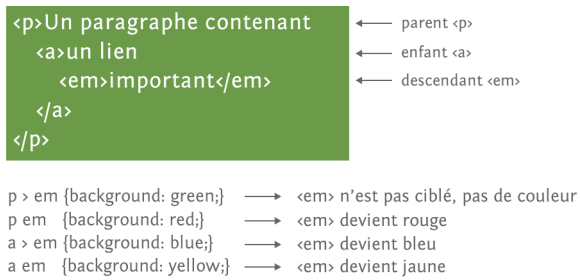
Le sélecteur d'enfant s'applique, comme son nom le laisse présager, à l'enfant ou aux enfants d'un élément désigné (figure 2-3). Sa syntaxe est la suivante :

```
E > F {propriété: valeur;}
```

À l'instar des autres sélecteurs, celui-ci se lit de la droite vers la gauche. Ainsi notre exemple va-t-il appliquer la déclaration à l'élément `F` s'il est enfant d'un élément `E`. À la différence du sélecteur classique de descendance, symbolisé par un espace entre les éléments (`E F`), le sélecteur d'enfant (`E > F`) ne s'applique qu'en cas de parenté directe et n'aura aucune prise si un élément se trouve hiérarchiquement entre `E` et `F`.

Figure 2-3

Exemple de sélecteur d'enfant



Au premier abord, l'intérêt de cette syntaxe peut paraître mineur. Il est pourtant très pratique dans le cas de documents emboîtant plusieurs données du même type, par exemple des listes imbriquées.

Partie HTML

```
<ul id="menu">
  <li><a href="url1">accueil</a></li>
  <li><a href="url2">société</a></li>
  <li><a href="url3">contact</a>
    <ul class="submenu">
      <li><a href="url3-1">plan d'accès</a></li>
      <li><a href="url3-2">formulaire de contact</a></li>
      <li><a href="url3-3">réseaux sociaux</a></li>
    </ul>
  </li>
</ul>
```

Partie CSS

```
#menu > li {list-style-type: square;}
```

Notre exemple illustre un menu de navigation à deux niveaux. Afin de ne cibler que la première génération de la liste (les enfants directs de l'élément identifié comme `menu`), j'ai choisi le sélecteur d'enfant. Ainsi, les éléments `<li>` contenus dans `submenu` ne seront pas affectés et conserveront leurs puces par défaut.

Il y a fort à parier que ce type de sélecteur prenne de l'importance avec la promotion de HTML 5, car ce dernier envisage de larges possibilités d'imbrications d'éléments, notamment des balises `<h1>` pour chaque section ou article, eux-mêmes pouvant être incorporés dans d'autres sections.

COMPATIBILITÉ Sélecteur d'enfant

Le sélecteur d'enfant est pris en charge par tous les navigateurs, à partir d'Internet Explorer 7.

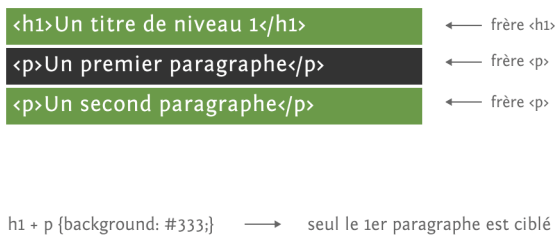
Sélecteur de frère adjacent

Le sélecteur de frère adjacent (ou sélecteur d'adjacence) affecte un élément à condition qu'il soit immédiatement précédé d'un autre élément spécifié (figure 2-4). Sa syntaxe est la suivante :

```
E + F {propriété: valeur;}
```

Figure 2-4

Illustration du sélecteur de frère adjacent



Cette règle s'applique à l'élément `F` s'il est frère de `E` et s'il lui succède dans l'ordre de déclaration HTML. Illustrons ce concept par un exemple simple constitué d'un titre de niveau 1 suivi par deux paragraphes.

Partie HTML

```
<h1>Le kiwi</h1>
<p>Le kiwi est un fruit, mais c'est aussi un animal</p>
<p>Le kiwi le plus répandu est le fruit de Actinidia deliciosa</p>
```

Partie CSS

```
h1 + p {font-style: italic;}
```

Dans notre exemple, seul le premier paragraphe `<p>`, directement adjacent au titre `<h1>`, se distinguera en italique. Les paragraphes suivants demeureront inchangés.

Visualiser le résultat en ligne

▸ http://www.ie7nomore.com/fun/columns/

Seul le paragraphe directement adjacent au titre principal diffère des suivants.

Ce type de sélecteur a pour avantage de cibler un groupe d'éléments identiques tout en excluant le premier. Par exemple, pour afficher une bordure à gauche de tous les éléments d'une liste sauf le premier, il suffit d'indiquer la règle suivante :

```
li+li {border-left: 1px solid black;}
```

Nous verrons un peu plus loin que CSS 3 étend la portée de ce sélecteur en l'appliquant à la fratrie indirecte.

COMPATIBILITÉ Sélecteur de frère adjacent

Le sélecteur de frère adjacent est compris par tous les navigateurs à partir d'Internet Explorer 7.

Sélecteur d'attribut

Le sélecteur d'attribut apparaît avec la spécification CSS 2.1. Il consiste en une extension du sélecteur d'élément, avec la possibilité de cibler un attribut HTML, la valeur d'un attribut, voire une partie de cette valeur.

Sa syntaxe est la suivante :

```
[attribut] {propriété: valeur;}
```

On peut éventuellement préciser l'élément concerné à l'aide de la syntaxe suivante :

```
E[attribut] {propriété: valeur;}
```

Il est ainsi possible de cibler, par exemple, uniquement les images disposant d'un attribut `alt`, à l'aide du sélecteur CSS `img[alt]`, et de les styler comme bon nous semble.

Le sélecteur d'attribut offre également l'opportunité de préciser la valeur attendue de l'attribut ciblé : le sélecteur `a[hreflang="fr"]` va s'appliquer à tous les liens dont l'attribut `hreflang` contient exactement la valeur `fr`. Pour ne cibler que les cellules de tableaux dont l'attribut `colspan` vaut `3`, il suffit d'écrire `td[colspan="3"]`.

Notez qu'il est parfaitement envisageable de cumuler deux sélecteurs d'attribut, comme le décrit l'exemple suivant :

```
img[class="vignette"][width="250"] {border: 1px solid green;}
```

Cette règle aura pour but d'ajouter une bordure verte autour des éléments d'image à condition que leur attribut `class` ait la valeur `vignette` et que leur attribut `width` vaille `250`.

Les guillemets entourant la valeur de l'attribut ne sont pas nécessaires, ni demandés par les spécifications, mais l'expérience a démontré une meilleure reconnaissance du sélecteur lorsqu'il en dispose.

Nous verrons dans le chapitre 8 que la version 3 de CSS ajoute encore de nouvelles options à ce sélecteur, permettant de cibler les valeurs débutant ou se terminant par une chaîne de caractères définie.

COMPATIBILITÉ Sélecteur d'attribut

Le sélecteur d'attribut est supporté par tous les navigateurs à partir d'Internet Explorer 7, mais avec des lacunes selon les syntaxes. En effet, dans certaines situations, IE7 ne reconnaît pas certains sélecteurs d'attributs si le nom de l'élément n'y est pas accolé : `input[type="text"]` sera globalement mieux interprété que `[type="text"]`.

:first-letter et :first-line

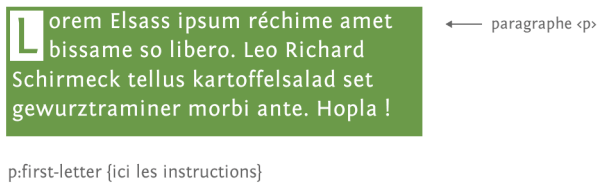
Les pseudo-éléments `:first-letter` et `:first-line` correspondent respectivement au premier caractère et à la première ligne de texte au sein d'un élément de type bloc ou équivalent.

Créés dès CSS 1, ces pseudo-éléments comptent encore aujourd'hui un certain nombre de différences d'appréciation minimes entre les navigateurs, bien qu'ils soient globalement bien reconnus.

`:first-letter` est traditionnellement utilisé pour mettre en exergue une lettrine, à l'instar de certains anciens ouvrages typographiques, comme le montre la figure 2-5.

Figure 2-5

Création d'une lettrine



Code CSS correspondant à l'élaboration d'une lettrine

```
p:first-letter {
  float: left; /* positionnement de la lettrine dans le conteneur */
  font: bold 3em Georgia, "Times New Roman", Times, serif;
  color: #900;
  padding: 3px 8px;
  margin: 5px 5px 0 0;
  border: 1px solid #900;
}
```

La spécification CSS 2 liste les propriétés qui peuvent être appliquées à `:first-letter`. Il s'agit des propriétés de police, marges externes (`margin`), marges internes (`padding`), bordures, couleur, arrière-plan, ainsi que les propriétés `text-decoration`, `text-transform`, `letter-spacing`, `word-spacing`, `line-height`, `float` et `vertical-align` (seulement si `float` a pour valeur `none`).

La norme indique que les agents utilisateurs peuvent prendre en charge d'autres propriétés, notamment de positionnement, s'ils le souhaitent.

COMPATIBILITÉ :first-letter et :first-line

Pris en charge depuis longtemps, ces deux pseudo-éléments présentent un certain nombre d'erreurs célèbres sur l'ensemble des navigateurs. La première est qu'Internet Explorer 6 ne va pas interpréter la règle lorsque le bloc de déclaration est collé au séparateur : `p:first-letter{propriété: valeur}` sera invisible pour IE6, contrairement à `p:first-letter {propriété: valeur}`.

Il est également connu, toujours au chapitre Internet Explorer, que ces pseudo-éléments ne vont pas s'appliquer à tous les éléments (voir partie consacrée à la résolution de bogues au chapitre 6).

:first-child

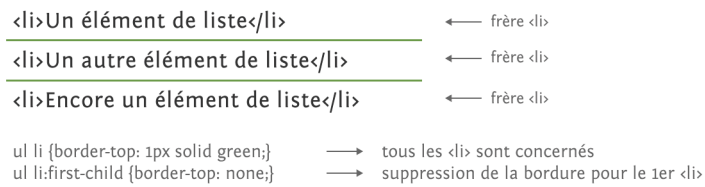
La pseudo-classe `:first-child`, définie depuis CSS 1, désigne le premier élément enfant au sein d'un élément (figure 2-6). Par exemple, `li:first-child` qualifie le premier élément `<li>` d'une liste `<ul>` ou `<ol>`, *et non pas le premier élément contenu dans* `<li>` *comme beaucoup sont tentés de le croire.*

Ce sélecteur rend très commode l'identification du premier élément parmi des frères semblables, comme le montre cet exemple qui a pour but d'appliquer une bordure sous les éléments d'une liste, sauf le premier :

```
ul li {border-top: 1px solid green;}
ul li:first-child {border-top: none;}
```

Figure 2-6

Illustration de first-child



COMPATIBILITÉ :first-child

`:first-child` n'est reconnue par Internet Explorer qu'à partir de la version 7. C'est pourquoi cette pseudo-classe est encore sous-exploitée malgré les nombreux bénéfices qu'elle apporte.

Par expérience personnelle, j'ai également parfois pu constater que certains navigateurs n'appliquent pas les styles lorsque le parent n'est pas spécifié : je recommande donc d'employer plutôt la syntaxe `ul li:first-child` que `li:first-child`.

:focus

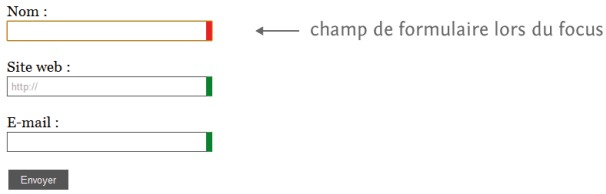
La pseudo-classe `:focus` se rapporte à un élément lorsqu'il est atteint via le clavier ou autre dispositif de pointage (figure 2-7). Selon la spécification, seuls les éléments acceptant l'attribut HTML `tabindex` (qui définit l'ordre de navigation avec la touche tabulation) sont concernés, c'est-à-dire `<a>`, `<area>`, `<button>`, `<input>`, `<object>`, `<select>` et `<textarea>`.

Pour rappel, l'ordre de déclaration des pseudo-classes est important dans le cas des liens hypertextes, pour faire en sorte que toutes s'appliquent :

1. `:link`
2. `:visited`
3. `:hover / :focus`
4. `:active`

Figure 2-7

Focus d'un champ de formulaire sur Chrome



COMPATIBILITÉ :focus

`:focus` est universellement reconnu par l'ensemble des navigateurs à l'exception d'Internet Explorer (seulement à partir de IE8, sauf pour les éléments de formulaires). Cependant, Internet Explorer 6 applique (incorrectement) la pseudo-classe `:active` en lieu et place de `:focus`. Je vous invite par conséquent à systématiquement doubler, voire tripler, vos états de survol en cumulant les sélecteurs `:hover`, `:focus` et `:active` pour le même bloc de règles, ceci pour une plus grande accessibilité de vos liens.

:before et :after

Les pseudo-éléments `:before` et `:after` ouvrent la voie à la création automatique de contenu via CSS, sans que celui-ci ne soit présent dans le document HTML (figure 2-8).

Attachés à un sélecteur, ces pseudo-éléments couplés à des propriétés spécifiques telles que `content` permettent d'afficher une chaîne de caractères, une image d'arrière-plan, des guillemets de citation ou encore des compteurs avant ou après l'élément. Pour être très précis, ce contenu apparaîtra dans la boîte de l'élément, entre les marges internes (`padding`) et le contenu de celui-ci.

La propriété `content` accepte différentes valeurs : toute chaîne de caractères placée entre guillemets simples ou doubles, une image de fond via `url(`*`chemin_de_l'image`*`)`, la valeur d'un attribut récupéré via `attr(`*`nom_de_l'attribut`*`)`, ou encore des mots-clés tels que `counter` (compteur), `open-quote` (guillemets de citation ouvrants), `close-quote` (guillemets de citation fermants), `no-open-quote` ou `no-close-quote` (pas de guillemets ouvrants ou fermants).

L'élément créé via `:before` ou `:after` n'existe pas dans l'arbre de document et a un mode de rendu par défaut de type `inline`, mais peut être mis en forme, dimensionné et positionné à l'aide des propriétés CSS classiques existantes.

ATTENTION Différence de syntaxe entre CSS 2.1 et CSS 3

Depuis CSS 3, la syntaxe de ces pseudo-éléments s'écrit à l'aide d'un double deux-points « : », c'est-à-dire `::before` et `::after`. Cependant, pour des raisons de compatibilité ascendante (l'écriture CSS 3 n'est pas reconnue par Internet Explorer 8 notamment) et parce que les deux syntaxes sont valides, j'ai choisi dans ce livre la syntaxe CSS 2.1 qui est `:before` et `:after`.

Dans les spécifications CSS 3, il est par ailleurs prévu que le contenu puisse être créé sans nécessiter la présence des pseudo-éléments `:before` ou `:after`.

Voici quelques exemples d'usage de ce couple de pseudo-éléments.

Libellé s'affichant automatiquement devant le fil d'Ariane d'un site web

```
#ariane:before {content: "Vous êtes ici : ";} /* à appliquer sur l'élément <ul
id="ariane"> */
```

Séparateurs « > » entre chaque élément du fil d'Ariane

```
#ariane li {display: inline}
#ariane li + li:before { content: "> "; }
```

Guillemets ouvrants au début d'un bloc de citation

```
blockquote:before {content: open-quote;}
```

Guillemets fermants à la fin d'un bloc de citation

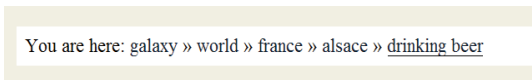
```
blockquote:after {content: close-quote;}
```

Images de fond affichées avant et après chaque paragraphe

```
p:before {
  content: url(images/arrondi_haut.png);
}
p:after {
  content: url(images/arrondi_bas.png);
}
```

Figure 2-8

Fil d'Ariane avec création de contenu



COMPATIBILITÉ :before et :after

Les pseudo-éléments `:before` et `:after` ne sont pas pris en charge par les versions d'Internet Explorer inférieures à IE8. Il est encore quelque peu prématuré de s'en servir, sauf à maîtriser parfaitement les conséquences d'une non-reconnaissance.

À ce propos, l'élément créé n'existant pas dans l'arbre du document, ni sur un agent utilisateur dont les CSS seraient désactivées, ce genre de techniques de création de contenu à la volée est vivement déconseillé pour des raisons évidentes d'accessibilité, à moins qu'il ne soit purement décoratif ou insignifiant.

Valeur de l'attribut attr()

`attr()` est une valeur particulière de la propriété `content` – et donc actuellement liée aux pseudo-éléments `:before` ou `:after` – qui renvoie la chaîne de contenu d'un attribut de l'élément ciblé. Cette valeur dynamique offre la possibilité d'afficher un contenu servant de méta-donnée normalement réservée aux agents utilisateurs.

Parmi les applications courantes, citons l'affichage des URL des liens hypertextes, très pratique au sein d'une feuille de styles d'impression :

```
a:after {
  content: " (" attr(href) ")";
}
```

Pour appliquer la règle précédente exclusivement à l'impression, il est possible de prévoir une feuille de styles dédiée et appelée via l'attribut `media`, ou simplement de placer une *règle* @ (`@media`) en début de feuille de styles classique.

```
@media print
{
  a:after {
    content: " (" attr(href) ")";
  }
}
```

Il est également possible d'employer cette fonction pour afficher la valeur d'un attribut `title` d'un lien, afin de signaler son ouverture dans une nouvelle fenêtre.

Partie HTML

```
<a href="http://www.alsacreations.com" title="s'ouvre dans une nouvelle fenêtre"
target="_blank">Le site d'Alsacréations</a>
```

Partie CSS

```
a[title]:after { /* on ne cible que les liens disposant d'un attribut title */
  content: " (" attr(title) ")"; /* on affiche la valeur de title à la suite du lien */
}
```

Une méthode plus simple consiste à considérer systématiquement les liens munis d'un attribut `target` comme hors de la page actuelle, ce qui nous dispense de l'usage de l'attribut `title`.

Partie HTML

```
<a href="http://www.alsacreations.com" target="_blank">Le site d'Alsacréations</a>
```

Partie CSS

```
a[target]:after { /* on ne cible que les liens disposant d'un attribut target */
  content: " (nouvelle fenêtre)";
}
```

DÉVELOPPEMENTS FUTURS **Possibilités étendues avec CSS 3**

Notez qu'en CSS 3, la valeur dynamique `attr()` a pour vocation de devenir une véritable fonction renvoyant d'autres types de valeurs et qui bénéficiera d'une autre syntaxe plus évoluée. Il sera par exemple possible d'utiliser `attr()` directement comme valeur d'une propriété CSS et pas forcément couplée à `::before`, `::after` ou `content`.

Exemple : `p {float: attr(class)}` sur un élément tel que `<p class="left">` ; dans ce cas, le paragraphe sera flottant à gauche.

Exercice pratique : langue de destination d'un lien

Notre premier cas d'étude concret consiste à agrémenter graphiquement les liens dont la destination vise des pages web dans une autre langue.

HTML propose un attribut destiné à indiquer la langue de destination d'un lien hypertexte : `hreflang`. La pseudo-classe `:after` permet d'afficher la valeur de cet attribut à la suite du lien.

Partie HTML

```
<a href="http://www.alsacreations.com" hreflang="fr">Le site d'Alsacréations</a>
```

Partie CSS

```
a[hreflang]:after { /* on ne cible que les liens disposant d'un attribut hreflang */
  content: " (" attr(hreflang) ")"; /* on affiche la valeur de hreflang à la suite
  ➥ du lien */
}
```

Cette solution demeure quelque peu austère, mais nous pouvons la remplacer avantageusement par une méthode plus graphique, en affichant un petit drapeau de langue :

```
a[hreflang="fr"]:after { /* on ne cible que les liens disposant d'un attribut hreflang
➥ dont la valeur est « fr » */
  content: url(img/flag_fr.png); /* on affiche une image de drapeau */
}
```

Règles @

Les règles @ (ou *règles-at*) sont des instructions pour la plupart définies dès CSS 1 et qui se présentent généralement sous forme de « méta-sélecteurs » permettant de regrouper des blocs de règles au sein d'un bloc de niveau supérieur.

De nombreuses règles @ sont proposées dans les spécifications, mais n'ont pour la plupart qu'un intérêt limité hors cas spécifiques. Je n'ai souhaité développer que les plus intéressantes, à savoir `@import`, `@media` et `@page`. La plus séduisante de toutes, `@font-face`, bénéficiera d'une section de chapitre entière dans la partie du livre dédiée à CSS 3 (chapitre 8).

@import

La règle `@import` inclut une feuille de styles CSS au sein d'une autre feuille de styles. Cette règle doit absolument être placée en tête du fichier, sous peine d'être invalide. Le chemin vers la feuille de styles est indiqué via la valeur `url(styles.css)`. Il est possible de préciser les types de médias concernés par cette feuille de styles au moyen d'un mot-clé placé après l'adresse du fichier externe.

Voici un exemple d'utilisation de `@import` pour l'appel d'une feuille de styles nommée `reset.css`. Notez qu'il n'y a pas de séparateur double point (`:`) après un sélecteur de type règle @ :

```
@import url(css/reset.css);
```

Il est aisément possible de préciser à quoi la feuille de styles doit s'appliquer :

```
@import url(css/print.css) print;
```

COMPATIBILITÉ @import

La règle `@import` est comprise par l'ensemble des navigateurs actuels. Toutefois, jusqu'à la version 8 (incluse), Internet Explorer présente une lacune dans l'implémentation de @import : si le type de média est précisé, la feuille de styles ne sera pas chargée. En outre, le navigateur de Microsoft révèle un comportement insolite, nommé FOUC (*Flash of Unstyled Content*), qui consiste en une fugitive apparition du contenu brut de la page, non stylée, avant que la navigateur n'en affiche la présentation CSS. Vous trouverez plus d''informations sur le lien suivant :

▸ http://www.bluerobot.com/web/css/fouc.asp/

@media

La règle `@media` indique qu'un bloc de règles ne concernera que des périphériques de sortie déterminés. Par exemple, certains styles peuvent être définis pour l'écran uniquement, d'autres pour l'impression :

```
@media screen
{
  body {background: url(img/wallpaper.jpg) center top;}
}
@media print
{
  body {background: #fff; color: #000;}
}
```

Les différents types de médias communément reconnus sont :

- `all` (l'ensemble des médias possibles) ;
- `aural` et `speech` (synthèses vocales) ;
- `handheld` (mobiles) ;
- `print` (imprimantes) ;
- `projection` (projecteurs) ;
- `screen` (écrans d'ordinateurs).

D'autres médias plus rares existent : `braille` (périphériques en ligne Braille), `embossed` (imprimantes Braille), `tty` (médias de sortie non graphiques : Télex, navigateurs orientés texte...) ou `tv` (télévision et apparentés...).

CSS 3 introduit la notion de *Media Queries* qui va étendre la portée de cette règle @ en donnant la possibilité de ne l'appliquer que sous certaines conditions environnementales (par exemple, uniquement aux écrans dont la définition maximale est de 480 px). Nous verrons cette instruction dans le détail au sein des parties consacrées à CSS 3 et aux mobiles (chapitre 9).

COMPATIBILITÉ @media

La règle `@media` est globalement assez bien traitée par les navigateurs actuels malgré une erreur présente à la fois chez Internet Explorer (jusqu'à la version 8 incluse) et Mozilla Firefox (jusqu'à la version 3.5 incluse) : si le type de média n'est pas renseigné, la règle @ ne sera pas prise en compte, alors qu'elle devrait s'appliquer par défaut à `all`.

@page

La règle `@page` ne s'applique qu'au sein d'un fichier de styles pour l'impression et a pour seule fonction de déterminer les marges des feuilles d'un support de sortie paginé.

Il est ainsi possible de fixer les marges appliquées au document lorsqu'il sera imprimé, par exemple 2 centimètres en haut et en bas et 1,5 centimètres à gauche et à droite :

```
@page {
  margin: 2cm 1.5cm;
}
```

À l'aide des pseudo-éléments `:left`, `:right` et `:first`, on peut également agir de façon plus précise sur les pages paires (ou gauche) et impaires (ou droite), ainsi que sur la page de garde :

```
@page {
  margin: 2.5cm; /* marge par défaut pour l'ensemble des pages */
}
@page :left {
  margin-left: 5cm; /* marge à gauche pour les pages de gauche uniquement */
}
@page :right {
  margin-right: 5cm; /* marge à droite pour les pages de droite uniquement */
}
@page :first {
  margin-top: 5cm; /* marge haute pour la page de garde uniquement */
}
```

COMPATIBILITÉ @page

La prise en charge de la règle `@page` est actuellement extrêmement médiocre : sur l'ensemble des navigateurs modernes, seul Opera reconnaît cette règle @. Il est donc malheureusement bien trop tôt pour l'employer.

Tableau récapitulatif

Vous trouverez ci-après un résumé des sélecteurs et règles évoqués au sein de ce chapitre ainsi que leurs compatibilités avec les navigateurs actuels.

Ce tableau ne tient compte que des différentes versions contemporaines des navigateurs : Internet Explorer à partir de IE6, Firefox 3 et plus, Chrome 5 et plus, Safari 4 et plus, Opera à partir de la version 9. Les versions très anciennes des navigateurs n'ont pas été prises en compte.

Tableau 2-1 Compatibilité des navigateurs avec CSS 2.1

	IE6	IE7	IE8	Firefox	Chrome	Safari	Opera
`!important`		OK	OK	OK	OK	OK	OK
`E > F`		OK	OK	OK	OK	OK	OK
`E + F`		OK	OK	OK	OK	OK	OK
`[attribut]`		OK	OK	OK	OK	OK	OK
`:first-letter`	OK	OK	OK	OK	OK	OK	OK
`:first-line`	OK	OK	OK	OK	OK	OK	OK
`:first-child`		OK	OK	OK	OK	OK	OK
`:focus`			OK	OK	OK	OK	OK
`:before`			OK	OK	OK	OK	OK
`:after`			OK	OK	OK	OK	OK
`@import`	OK	OK	OK	OK	OK	OK	OK
`@media`	OK	OK	OK	OK	OK	OK	OK
`@page`							OK

Microformats

Définition et usage

À l'ère du « Web 2.0 » – ouh ! que je déteste ce terme – des légions de technologies « sociales » se développent chaque jour en révolutionnant les standards de communication habituels du Web : XML, flux RSS, Ajax, réseaux sociaux, etc. Tant et si bien que ce flot d'informations continu et complexe bouscule parfois quelque peu les outils en mesure d'exploiter ces données.

Le concept récent des *microformats* vise à redéfinir une description sémantique du contenu afin de le rendre non seulement plus visible, mais aussi et surtout plus compréhensible aux différents agents utilisateurs et moteurs de recherche intelligents. Il ne s'agit pas d'un nouveau langage, mais d'un apport d'informations aux langages web courants.

L'une des définitions des microformats pourrait être : *tout format standard basé sur XML ou (X) HTML et destiné à fournir des métadonnées additionnelles aux contenus web qui peuvent être extraites et interprétées par des programmes.*

Plus concrètement, ces entités sémantiques facilitent la publication et la diffusion d'éléments tels que des actualités, des événements, des localisations, des cartes de visite virtuelles ou produits en ligne, notamment via les éléments HTML `<address>`, `<cite>` et `<blockquote>`, les attributs tels que `rel`, `rev` ou `title` ou encore de classes CSS définies comme `class="vcard"`, `class="postal-code"`, `class="hreview"` ou encore `class="rating"`.

ALLER PLUS LOIN Microformats

Le site web Microformats explique et détaille le principe des microformats, propose des exemples d'usages pratiques ainsi qu'une liste de sites web exploitant ces données.

▸ http://microformats.org

Types de microformats

Les microformats se présentent en réalité sous plusieurs entités différentes que l'on peut employer séparément ou de façon imbriquée :

- hCard : représentation de personnes, de sociétés ou d'organisations ainsi que leur localisation ;
- hCalendar : format destiné aux calendriers, aux événements ;
- hReview : format dédié à la popularité de produits, aux avis de consommateurs pour des catalogues, des services ou des événements ;
- hResume : pour la publication de CV ;
- attribut `rel` (valeurs `nofollow` ou `tag`) ;
- et bien d'autres encore (XFN, hAtom, Géo…).

Qui en tient compte ?

Encore peu de sites web tiennent compte des microformats à l'heure actuelle, mais les plus avant-gardistes (j'ai encore failli écrire « Web 2.0 » !) en intègrent déjà les bénéfices. On peut citer par exemple Twitter, MySpace, Kelkoo, Yahoo!, Technorati ou encore quelques extensions Firefox telles que Greasemonkey. Le site microformats.org tient à ce propos une liste assez impressionnante.

Toutefois, si je ne devais en mentionner qu'un seul pour faire pencher votre décision, il s'agirait sans aucun doute de Google. En effet, Google est un fervent adepte des microformats. Il a confirmé leur prise en charge dans son annuaire de recherche et prône leur usage au sein de son « Centre pour webmasters ».

Par exemple, si vous indiquez l'adresse exacte d'un lieu, ou ses latitude et longitude, vous permettez à Google Maps de le situer sur son plan interactif. De même, un événement défini à l'aide d'un balisage de dates microformaté sera interprété par les actualités de Google (News) ainsi que son calendrier (Calendar).

Vous devinez que dans le cas d'une entreprise ou d'une marque, l'ajout de microformats de type hCard peut être appréciable en termes de référencement, notamment pour faire ressortir des points de vente locaux.

Exercice pratique : contact d'entreprise

Pour mieux comprendre le concept des microformats, appliquons-le à un exemple courant sur un site web : la partie affichant les coordonnées de son entreprise. J'ai bien entendu choisi une société de façon totalement aléatoire, une certaine agence web nommée Alsacréations.

HTML d'origine, sans microformats

```
<div>
  <ul>
    <li>Alsacréations</li>
    <li>Agence web exotique</li>
    <li>5, rue des Couples</li>
    <li>67000 Strasbourg France</li>
  </ul>
  <p>contact@alsacreations.fr</p>
  <p>Tél. xx xx xx xx xx</p>
  <p>
    <a href="http://twitter.com/alsacreations">Twitter</a>
  </p>
</div>
```

Vous allez constater que le format hCard, prévu pour décrire les personnes ou les entreprises, ainsi que leur localisation, va nous fournir des valeurs prédéfinies que l'on va associer à un attribut HTML de classe.

HTML avec microformats

```
<div class="vcard"> <!-- appel au format hCard -->
  <ul class="adr">  <!-- propriété globale d'adresse hCard -->
    <li class="fn org name">Alsacréations</li>
    <li class="title">Agence web exotique</li>
    <li class="street-address">5, rue des Couples</li>
    <li><span class="postal-code">67000</span>  <span class="locality">Strasbourg
        </span> <span class="country-name">France</span></li>
  </ul>
  <p class="email">contact@alsacreations.fr</p>  <!-- classe hCard d'e-mail -->
  <p class="tel">Tél. xx xx xx xx xx</p>  <!-- classe hCard de téléphone -->
  <p class="social url">   <!-- classes hCard de type social et de lien -->
    <a href="http://twitter.com/alsacreations">Twitter</a>
  </p>
</div>
```

Aucun des noms de classes présents dans cet exemple n'est le fruit du hasard : ils sont tous répertoriés et codifiés dans le standard hCard 1.0, que l'on peut consulter sur le site microformats.org.

Chaque type de microformat définit son propre périmètre d'action et ses propriétés intrinsèques. Libre à vous de les employer dès aujourd'hui en production : ils représentent un greffon totalement inoffensif à vos documents web. Au pire, ils ne seront interprétés par aucun agent utilisateur et ne porteront pas atteinte à vos informations ; au mieux, ils faciliteront la description de vos données, voire leur référencement dans les moteurs de recherche.

Quiz de connaissances

Questions

Question 1

De quelle manière optimale puis-je cibler à la fois tous les titres de niveau 1 de ma page, tous les éléments de classe `fruit`, ainsi que l'élément dont l'identifiant est `header` ?

Question 2

Comment puis-je cibler uniquement le premier niveau d'éléments `<div>` contenus au sein de mon document (`<body>`) ?

Question 3

Je désire colorer le texte du troisième paragraphe ci-dessous en rouge. Comment puis-je procéder pour le cibler en CSS 2.1 sans modifier le code HTML ?

Partie HTML

```
<p>premier paragraphe</p>
<p>deuxième paragraphe</p>
<p>troisième paragraphe</p>
```

Question 4

De quelle couleur sera le titre de mon document issu du code HTML ci-dessous et des règles de styles CSS suivantes ?

Partie HTML

```
<div class="container"><h1 id="first" class="fruit">Potage de kiwi</h1></div>
```

Partie CSS

```
h1 {color: green;}
#first {color: blue;}
h1.fruit {color: yellow;}
h1.fruit:first-child {color: red;}
div.container > h1.fruit:first-child {color: gray;}
* h1.fruit {color: pink;}
```

Question 5

À quel endroit de la boîte d'un élément s'affiche le contenu créé avec `:before` ou `:after` ?

- à l'extérieur de ses marges externes ?
- entre ses marges externes et sa bordure ?
- entre sa bordure et ses marges internes ?
- entre ses marges internes et son contenu ?

Question 6

Construisez le code HTML correspondant à la règle CSS suivante :

```
p + h1.kiwi + p strong
```

Question 7

Construisez le code HTML correspondant à la règle CSS suivante :

```
p[title*="kiwi"] + p > strong.info
```

Question 8

Construisez le code HTML correspondant à la règle CSS suivante :

```
form#contact[action="destination.php"] > p > label[for="kiwi"] + #kiwi[type="text"]
```

Réponses

Réponse 1

La meilleure façon de cibler rapidement tous les différents éléments mentionnés est d'employer le sélecteur de groupe :

```
h1, .fruit, #header { ici les déclarations }
```

Réponse 2

Pour cibler uniquement le premier niveau de `<div>` du document, nous allons nous servir du sélecteur d'enfant :

```
body > div  { ici les déclarations }
```

Réponse 3

Dans notre cas de figure limité à trois paragraphes, il est aisé de sélectionner le dernier élément à l'aide du sélecteur d'adjacence :

```
p + p + p  { ici les déclarations appliquées au 3è paragraphe }
```

Réponse 4

La règle de priorité des sélecteurs s'applique aux différentes règles : le poids de l'identifiant (`#first`) l'emporte sur tous les autres types de sélecteurs proposés (classe, attribut, pseudo-classe, élément et pseudo-élément). Au final, le titre du document sera de couleur bleue.

Réponse 5

Le contenu créé via les pseudo-éléments `:before` et `:after` s'affiche entre ses marges internes et son contenu.

Réponse 6

Voici un exemple de code HTML correspondant à la règle CSS proposée :

```
<p>un paragraphe</p>
<h1 class="kiwi">un paragraphe</h1>
<p>un <strong>dernier</strong> paragraphe</p>
```

Réponse 7

Voici un exemple de code HTML correspondant à la règle CSS proposée :

```
<p title="nicekiwi">un paragraphe</p>
<p>un <strong class="info">dernier</strong>paragraphe</p>
```

Réponse 8

Voici un exemple de code HTML correspondant à la règle CSS proposée :

```
<form id="contact" action="destination.php">
  <p>
    <label for="kiwi">Nom :</label>
    <input type="text" id="kiwi" />
  </p>
</form>
```

3

Gestion de projet et performance

Ce chapitre apporte les clés et les solutions pour perfectionner votre méthode de travail, à travers trois axes : la gestion de projet (conventions, nommage, ordonnancement et méthodologie), l'optimisation des performances (mises en forme par défaut, performance des sélecteurs, minification et compression des styles) et l'usage des outils en ligne et des extensions de navigateurs.

Dans le domaine de la conception web, le seul bagage technique ne suffit pas à mener à bien un projet dans son ensemble. Quand bien même vous connaîtriez toutes les propriétés CSS et les schémas de positionnement sur le bout des doigts, vous ne pourrez vous dispenser d'une méthodologie convenable et d'un minimum d'optimisation de vos ressources. Or très peu d'ouvrages s'attachent au contexte dans lequel sont produites les pages CSS et aux astuces du quotidien pour gagner en productivité.

Que votre projet soit collaboratif ou personnel, voici un certain nombre de bonnes pratiques issues de ma propre expérience en agence de conception web.

Bien gérer un projet CSS

Vous êtes-vous déjà posé la question suivante : « Pourrai-je réutiliser cette feuille de styles CSS dans le cadre d'un autre projet ? »

Irrémédiablement, vous y répondriez par la négative, pas seulement parce que chaque projet est différent ni parce que vous aurez peut-être changé de collègues la prochaine fois, mais

principalement parce que vous avez employé des règles de nommage CSS trop spécifiques et non réutilisables, que vos fichiers manquent de documentation et de commentaires explicatifs… ou tout simplement de cohérence dans l'ordonnancement des déclarations.

Ce chapitre passe en revue les différentes méthodologies à adopter pour mieux organiser ses projets d'intégration HTML et CSS.

Un code pertinent et réutilisable

Je ne vous apprendrai pas que le choix d'un nom de classe ou d'identifiant n'est pas une décision à prendre à la légère et je suis persuadé que vous évitez déjà autant que possible les appellations se rapportant à la présentation. En effet, si votre menu de classe `gauche` devait finalement se retrouver à droite de votre page, vous risqueriez d'être passablement perturbé...

Convention de nommage

C'est pourquoi vous privilégiez dans la mesure du possible des noms de classes ou d'identifiants riches de sens et exprimant une fonction précise, tels que : `information`, `entête`, `recherche`…

Toutefois, dans l'optique de vous préparer à la nomenclature prévue par HTML 5, que nous verrons en détail un peu plus loin (chapitre 7), je vous invite dès à présent à opter pour une dénomination que l'on pourrait qualifier de *HTML 5 friendly*, c'est-à-dire optimisée pour la prochaine mouture de HTML, car cette bonne pratique vous facilitera le passage vers les nouveaux éléments à venir.

Pour nous aider dans notre tâche quotidienne, nous avons défini chez Alsacréations une convention de nommage interne pour les éléments les plus couramment stylés en CSS. Précisons que par souci d'internationalisation de nos codes, nous avons opté pour l'anglais.

Cette liste est conçue pour être réutilisable dans toutes nos missions d'intégration :

```
#header {…} ← bloc d'en-tête, en prévision de HTML 5 <header>
#footer {…} ← bloc de pied de page, en attendant <footer>
#nav {…} ← liste de navigation, un jour remplacée par <nav>
#sidebar {…} ← partie latérale
#contact {…} ← formulaire de contact
.clear {…} ← élément en clear: both
.info {…} ← élément ou bloc d'information
.warning {…} ← élément ou bloc d'attention
.error {…} ← signalement d'une erreur (formulaire)
.success {…} ← signalement d'un succès (formulaire)
.button {…} ← élément sous forme de bouton
.more {…} ← lien générique « en savoir plus »
.block {…} ← mise en forme générique d'un bloc (coins arrondis, ombrages,
             couleur de fond,…)
etc.
```

Retenez que l'important n'est pas tant la « sémantique » choisie que la cohésion au sein de votre équipe : optez pour une convention de nommage interne qui vous assurera que vous serez à même de comprendre les codes de votre collègue dans six mois et que vous pourrez réutiliser

votre feuille de styles de base pour de nouveaux projets sans avoir à réinventer la roue. Bref, entraînez-vous à produire du code recyclable.

Un nommage polyvalent

Sans doute saviez-vous qu'il est parfaitement autorisé de cumuler un attribut d'identifiant et un attribut de classe pour le même élément, par exemple `<p class="fruit" id="kiwi">` (je conviens que le choix de nommage n'est pas forcément le plus pertinent ici).

Dans cet exemple, notre paragraphe appartient à la classe `fruit` et peut être stylé comme tel, comme d'autres éléments, tout en se distinguant par son identifiant `kiwi`.

Un exemple de mise en forme pourrait être celui-ci :

```
.fruit { /* on applique les styles communs */
  float: left;
  width: 200px;
  border: 1px solid #555;
}
#kiwi { /* on cible l'exception */
  background-image: url(img/kiwi.png);
}
```

Cette méthode a pour avantage de regrouper des propriétés de mise en page identiques à un groupe d'éléments.

Dans cette lignée, il est moins connu qu'un élément peut disposer de noms de classes multiples en les séparant par un espace, par exemple `<p class="fruit exotic">`. Dans ce cas, le paragraphe en question sera affecté à la fois par les styles appliqués sur `.fruit` et sur `.exotic` :

```
.fruit {
  ici des déclarations ciblant les éléments de classe « fruit »
}
.exotic {
  ici des déclarations ciblant les éléments de classe « exotic »
}
.fruit.exotic {
  ici des déclarations ciblant les éléments de classe « fruit » et « exotic »
  à la fois. N'est pas reconnu par IE6
}
```

Ce genre de regroupement de classes a l'avantage d'affecter plusieurs types de propriétés aux mêmes éléments : inutile d'en créer de supplémentaires.

Ordre des déclarations

S'y retrouver au sein d'un fichier CSS de plusieurs centaines de lignes de déclarations est souvent fastidieux, notamment dans le cadre d'un projet web collaboratif et quand vos collègues ont d'autres habitudes d'écriture que les vôtres.

Par expérience, j'ai pu constater que l'un des moyens pour améliorer rapidement la lisibilité du code CSS est de systématiquement faire apparaître les déclarations dans un ordre identique au sein des blocs de règles. La compréhension et la relecture de la feuille de styles en sont grandement facilitées (figure 3-1).

À ma connaissance, aucun organisme n'établit ce que l'on pourrait appeler un « agencement officiel ». Je vous suggère cependant de respecter la logique suivante, même si celle-ci – vous allez vous en rendre compte – n'est pas forcément adaptée à tous les cas pratiques :

1. **Contenu créé :** les propriétés afférentes au contenu créé via `:after` et `:before` (`content`, `counter`, `quotes`).
2. **Propriété `display` :** tout ce qui affecte le rendu par défaut de l'élément (`none`, `block`, `inline`, `inline-block`, `table`, `table-cell`, `table-row`, `box`, `list-item`).
3. **Positionnement :** tout ce qui détermine la position de l'élément (`position`, `float`, `top`, `right`, `bottom`, `left`, `vertical-align`, `z-index`, `clear`).
4. **Modèle de boîte :** tout ce qui influe sur les dimensions de l'élément (`width`, `height`, `min-width`, `min-height`, `max-width`, `max-height`, `padding`, `margin`, `border`, `overflow`).
5. **Transformations et transitions :** propriétés et valeurs CSS 3 (`transform`, `rotate`, `scale`, `skew`, transition).
6. **Typographie :** tout ce qui détermine les caractéristiques de la police de caractères (`font`, `text-align`, `text-decoration`, `letter-spacing`, `text-indent`, `line-height`, `text-transform`, `white-space`, `word-wrap`).
7. **Décoration :** les propriétés purement ornementales (`background`, `color`, `list-style`, `outline`).

Ce genre de compartimentage demeure imparfait dans la mesure où certaines propriétés sont polymorphes, à l'image de `border`, composante intrinsèque du modèle de boîte mais qui sert généralement à la décoration, ou encore de `color`, propriété commune à la fois en typographie et en décoration.

Figure 3-1

Un bloc de règles bien ordonné

```
h1 {
  display: inline-block;
  float: left;
  width: 15em;
  margin-top: 0;
  font-family: 80px/0.8 Tahoma, sans-serif;
  text-align: right;
  color: white;
  background-color: #fae;
  text-shadow: 0px 0px 5px #fff;
}
```

Commentaires « utiles »

À l'instar du langage HTML, CSS prévoit la possibilité d'insérer des commentaires au sein des feuilles de styles. Ces commentaires ne sont pas interprétés par les agents utilisateurs mais uniquement destinés aux éventuels relecteurs du fichier. Il est possible d'en placer à n'importe

quel endroit de la page CSS, sauf au cœur d'une déclaration (entre le couple « propriété » et « valeur »).

En termes de syntaxe, vous écrirez un commentaire CSS en débutant par les caractères `/*` et en concluant par `*/`, par exemple `/* ceci est un commentaire CSS */`.

Dans le cadre d'un projet collaboratif, l'impact des commentaires CSS est loin d'être négligeable. Il peut devenir un extraordinaire outil de relecture et de débogage.

Segmenter le document

En pratique, il est généralement indiqué d'employer les commentaires CSS afin de distinguer nettement chaque bloc de structure principal (`header`, `menu`, `content`, `aside`, `footer`…) comme l'illustre l'exemple suivant.

Je vous invite aussi à placer un signe égal (`=`) dans le nom de ces blocs de structure, pour faciliter les recherches dans votre document. Ainsi, vous pourrez parcourir rapidement tous vos blocs en recherchant simplement ce symbole `=`.

```
/* ----------------------------- */
/*  =     Header                 */
/* ----------------------------- */

header {
  display: block;
}
…
```

Dans une moindre mesure, il peut être recommandé de désigner des sous-parties à l'aide de commentaires plus discrets mais néanmoins pertinents :

```
/*    global links        */

a {
  text-decoration: none;
  color: #000;
}
a:hover, a:focus, a:active {
  text-decoration: underline;
}
…
```

Employer des mots-clés définis

Toujours dans l'optique de mieux appréhender un projet en commun, CSSdoc (www.cssdoc.net) propose une convention à appliquer au sein des commentaires d'un document CSS. Toute personne partageant des fichiers CSS, qu'elle soit intégrateur, développeur ou créatif, peut tirer bénéfice de cette recommandation née en 2008.

Le principe de CSSdoc est de se référer à une liste de mots-clés prédéfinis qui permettent, non seulement de retrouver rapidement des parties spécifiques du document (zones laissées en

suspens ou non résolues), mais également d'adopter une nomenclature facilitant la relecture du fichier. Ces différents mots-clés sont immédiatement perceptibles, puisqu'ils commencent tous par un signe `@`.

Dans sa version complète, cette convention de nommage peut s'avérer quelque peu « usine à gaz » tant les fonctionnalités et mots-clés sont nombreux.

Dans la pratique, je vous conseille de vous limiter à l'usage de quelques mots-clés préalablement identifiés et débattus avec votre équipe de projet, dont voici une courte liste :

- mots-clés relatifs au document : `@autor` (nom de l'auteur), `@version` (numéro de version), `@date` (date de la version), `@licence` (droits sur le document), `@note` (note pour soi même) ;
- mots-clés concernant les erreurs : `@bugfix` (détail sur la technique de correction d'erreur utilisée), `@affected` (navigateurs affectés par cette erreur), `@tested` (navigateurs testés), `@see` (lien vers une ressource en ligne) ;
- mots-clés désignant les tâches restantes : `@todo`.

Quelques exemples au quotidien :

```
/* @autor : Raphael Goetter, Alsacreations */
/* @date : Mai 2010 */
/* @bugfix : bug IE6/IE7 de haslayout, résolu avec zoom: 1 */
/* @bugfix : ici les inline-block transformés en inline + zoom: 1 pour IE6/IE7 */
/* @todo : ici penser à trouver une alternative aux coins arrondis pour IE */
/* @todo : corriger le problème d'espace entre les listes */
/* @note : on pourrait trouver une technique plus propre pour cette partie */
```

Débuter par un sommaire

Si la feuille de styles se révèle longue et complexe, il est parfois extrêmement judicieux de proposer dès le début du fichier un résumé rassemblant l'ensemble des grandes sections développées dans le document.

Voici un exemple de sommaire placé en tête du fichier CSS :

```
/* ----------------------------- */
/*  =     Summary                */
/* ----------------------------- */

/* 1- Mentions */
/* 2- Base styles */
/* 3- Blocs and Misc */
/* 4- Header */
/* 5- Main */
/* 6- Sidebar */
/* 7- Contact */
/* 8- Footer */
…
```

Afficher ses codes couleur

La notion de constante et de variable n'existe pas encore en CSS à l'heure actuelle. Il est pourtant fréquent de réutiliser des valeurs identiques tout au long de son fichier de styles.

Dans le cadre d'un projet conséquent et de feuilles de styles complexes, il devient pertinent de débuter votre document par la liste de vos « codes couleurs » en commentaires. Cela va faciliter non seulement la lecture et la recherche de styles, mais aussi l'ajout de nouvelles règles à partir d'une base définie dans votre charte.

Voici un exemple de codes couleurs initiant un fichier CSS :

```
/* ----------------------------- */
/*  =     Color codes             */
/* ----------------------------- */
/*
Black (text):            #000
Dark gray (links, h1)    #008dde
Mid gray (visited links) #0066a1
Light gray (h2)          #747474
Mid grey (strong, code)  #333
*/
```

Gérer les versions

Afin d'éviter les conflits et l'écrasement de styles (ou de pages entières) par inadvertance, des solutions logicielles existent. Il s'agit des logiciels de gestion de versions, CVS (ou VCS en anglais, pour *Version Control System*). Les plus célèbres de ces outils, sous licence libre, se nomment Subversion (SVN) et GIT (figure 3-2).

Figure 3-2

Le site web de GIT

« Le système travaille par fusion de copies locale et distante, et non par écrasement de la version distante par la version locale. Ainsi, deux développeurs travaillant de concert sur une même source, les changements du premier à soumettre son travail ne seront pas perdus lorsque le second, qui a donc travaillé sur une version non encore modifiée par le premier, renvoie ses modifications. » (Source : Wikipédia.)

Les opérations permises par Subversion sont nombreuses :

- récupérer toutes les versions intermédiaires des fichiers ;
- afficher les différences entre les versions ;
- verrouiller des fichiers ;
- renommer et déplacer des fichiers sans en perdre l'historique ;
- commenter les modifications effectuées ;
- attacher à un fichier des propriétés comme les permissions.

Optimiser les performances

Au-delà de la gestion globale d'un projet CSS, un second aspect est malheureusement trop souvent négligé : celui de la performance des outils, des techniques et des fichiers employés. Même si les connexions Internet se font de plus en plus rapides, la démocratisation de supports mobiles tels que les ordinateurs portables, les smartphones, iPad et autres netbooks font que les réseaux « lents » Wi-Fi et 3G ne se sont jamais aussi bien portés.

Par ailleurs, Google a annoncé récemment que son nouvel algorithme de référencement tenait compte de la rapidité d'affichage des pages, ce qui a fait l'effet d'une bombe dans le monde de la conception web et des performances.

ALLER PLUS LOIN Performance des sites web

Dans le vaste domaine de la performance de sites web œuvre un expert français en la matière, Éric Daspet, auteur d'un best-seller sur PHP 5 avancé. Il a créé un groupe de discussion sur le sujet, où vous découvrirez de nombreuses pistes et recommandations extrêmement riches :

▸ https://sites.google.com/a/survol.fr/webperf-user-group/

Appliquer un Reset CSS

Préambule : le contenu de cette partie provient d'une astuce libre (licence Creative Commons) publiée par Corinne Schillinger sur Alsacréations.

▸ http://www.alsacreations.com/astuce/lire/36-reset-css.html

Le *Reset CSS* est une technique qui consiste à réinitialiser à 0 la valeur de certains éléments HTML, afin d'éviter une partie des différences d'affichage sur les divers navigateurs.

Le Reset universel : méfiance !

Les navigateurs n'utilisent pas toujours les mêmes marges et padding par défaut pour les différents éléments HTML. Cela peut représenter un problème, par exemple lorsqu'on veut supprimer le retrait à gauche d'une liste avec un `margin-left: 0;` et que certains navigateurs conservent ce retrait, car ils utilisent une marge interne (`padding`) plutôt qu'une marge externe. Ajoutez quelques différences de ce type entre les styles par défaut des navigateurs et la solution qui semble s'imposer est la suivante : supprimer toutes les marges et retraits internes des éléments à l'aide du sélecteur universel étoile.

```
* {
  margin: 0;
  padding: 0;
}
```

Le problème de cette technique est qu'elle annule aussi certains styles par défaut des navigateurs qui sont réellement utiles. Le cas le plus flagrant est celui des éléments de formulaire.

Par défaut, et avec la plupart des navigateurs, ceux-ci prennent l'apparence des éléments de contrôle (champs de texte, listes déroulantes, boutons…) du système d'exploitation. Puisque ces styles sont connus et facilement identifiés des utilisateurs et que, par ailleurs, il est très difficile, voire impossible, de styler certains éléments de formulaire à sa convenance, on conseille donc de garder cette apparence par défaut.

Seulement voilà, le sélecteur universel étoile (`*`) a pour avantage, tout comme inconvénient, de sélectionner sans distinction tous les éléments HTML. Et si on utilise un `border: 0;`, on va perdre la plupart des styles « système » des éléments de formulaire.

De plus, utiliser le sélecteur consomme des ressources au niveau du navigateur, qui doit parcourir tous les éléments du document pour leur appliquer ce style.

Un Reset intelligent ?

Pour éviter d'employer la grosse artillerie qu'est le sélecteur universel, je vous invite plutôt à utiliser un Reset CSS « intelligent », qui vise plus spécifiquement les différents éléments HTML et leur applique les styles nécessaires. On peut citer les deux références suivantes :

- le Reset CSS d'Eric Meyer (troisième version, janvier 2008) : http://meyerweb.com/eric/tools/css/reset/ ;
- le Reset CSS du projet Yahoo! UI Library : http://developer.yahoo.com/yui/reset/.

Limites des Reset CSS

Ces deux techniques de Reset CSS ont le mérite d'être suffisamment détaillées pour s'affranchir des points les plus problématiques. Cependant, c'est le principe même du Reset CSS qu'il convient de juger : est-ce une bonne chose d'annuler une grande partie des styles par défaut ?

Les styles par défaut des navigateurs fournissent un document HTML lisible même en l'absence de styles CSS spécifiques créés par l'auteur du site. Ils permettent d'identifier les différents éléments : paragraphes, titres, blocs de citation, etc. Supprimer ces styles fait donc courir un

risque : celui de se retrouver avec des contenus peu ou pas stylés, parce que l'on aura oublié de redéfinir certains styles. Et cela arrive plus souvent qu'on ne le pense !

De plus, il ne faut pas oublier que la personne qui code le CSS a rarement accès à tout le contenu HTML qui sera utilisé sur le site, soit parce que les contenus ne sont pas définitifs et peuvent être modifiés, soit parce que les contenus sont variables : système de publication d'articles, commentaires des utilisateurs, mise en place d'un CMS qui sera utilisé par des personnes pas formées au HTML, etc.

Une feuille de styles réutilisable

Pour éviter ce type de problème, la solution peut être d'utiliser non pas un Reset CSS, mais une feuille de styles de base. Cette dernière devra non pas annuler les styles par défaut des différents navigateurs, mais les harmoniser.

Voici un exemple de feuille de styles par défaut qu'il m'arrive d'employer en production. Celle-ci est inspirée d'un projet plus global, HTML 5 Boilerplate (http://html5boilerplate.com) :

```
/* ------------------------------ */
/*   Stylesbase                   */
/* ------------------------------ */

/*
* @autor : Alsacreations
* @date : 2010, october
*/

/* soft reset */

html, body, form, fieldset, legend, ul, ol, dl, blockquote, pre, h1, h2, h3, h4, h5,
➥ h6, code, kbd, q {
  margin: 0; padding: 0;
}

/* structure */

html {
  font-size: 100%;
}
body {
  font-family: "Lucida Grande", Tahoma, Helvetica, Sans-Serif;
  font-size: 62.5%;
  color: #000;
  background-color: #fff;
  line-height: 1.3;
}

/* introducing new HTML5 elements */

header, footer, section, hgroup, aside, nav, article, figure, figcaption, time, dialog {
  display: block;
```

```
}

/* forms */

label, input[type="button"], input[type="submit"], button { cursor: pointer; }
input, button, select {
  font-size: 100%;
  vertical-align: middle; /* @bugfix : align solution */
}

/* overflows */

pre, code, kbd, samp { font-family: monospace, sans-serif; }
img, table, td, blockquote, code, pre, textarea, input, object, embed, video {
  max-width: 100%;
}
code, pre, samp, textarea, table, td  {
  word-wrap:break-word;
  white-space: pre-wrap;
}

/* misc  */

a img {border: 0;} /* @note : no borders on image-links */
abbr[title]{border-bottom: 1px dotted #555; cursor:help; }
table {table-layout: fixed; border-collapse: collapse; border-spacing: 0;}
th, caption { text-align: left;}

.clear {clear: both;}
.info {}
.warning {}
.error {}
.success {}
```

Selon le type de projet, je choisis soit de copier ces styles en tête de la feuille de styles générale du site web, soit de les inclure au préalable sous forme de fichier distinct :

```
<link rel="stylesheet" media="all" href="stylesbase.css" />
```

Performances des sélecteurs

Des études d'optimisation ont permis de constater un rapport entre, d'un côté, la syntaxe et la complexité des sélecteurs CSS et, de l'autre, la rapidité d'affichage de la page créée… dans le cas de documents extrêmement volumineux.

Contrairement à une idée reçue, nous ne facilitons pas la tâche du navigateur en lui fournissant le plus de détails possibles dans la syntaxe d'un sélecteur. Le moteur doit s'arrêter à chaque nœud du sélecteur ; donc plus celui-ci est multiple, plus cela est coûteux en termes de ressources.

Dans la pratique, cela signifie qu'il est toujours préférable de cibler un élément au plus simple, au plus court, sans sélecteur intermédiaire inutile. Par exemple : le sélecteur `#header a` est plus efficace que le sélecteur `div#header ul#nav li a`, même si les deux désignent généralement le même élément.

EN SAVOIR PLUS **Performances des sélecteurs**

À titre informatif, plusieurs ressources traitent de ce sujet de performance des CSS, notamment la rubrique *Use efficient CSS selectors* de Google :

▸ http://code.google.com/intl/fr-FR/speed/page-speed/docs/rendering.html

Quelques règles utiles

Voici une liste de quelques règles à suivre dans la mesure du possible :

- N'utilisez pas de sélecteur universel (caractère étoile). Exemple à éviter : `* {margin: 0;}`.
- Limitez le nombre de sélecteurs de descendance (caractère espace). Privilégiez `#header a` à `#header ul li a`.
- Choisissez si possible le sélecteur d'enfant (`>`) plutôt que le sélecteur de descendance (caractère espace). Préférez `#header > img` à `#header img`, tout en tenant compte que ce sélecteur n'est pas reconnu par IE6.
- Autant que possible, utilisez un identifiant, une classe ou un sélecteur d'élément, mais pas une combinaison de deux ou trois. Préférez `#header a` à `div#header menu a.lien`.
- Évitez de faire précéder un identifiant par le nom de l'élément ciblé (idem pour les classes). Préférez `#nav` à `ul#nav`.

Méthode pratique

Pour commencer… ne vous affolez pas ! Ces règles ne doivent pas devenir une contrainte au quotidien, ni même être respectées à la lettre. Je le rappelle : l'impact en termes de performance est quasi négligeable sur des documents de taille raisonnable et il y a bien d'autres aspects que celui-ci à traiter en priorité dans un projet web. En ce qui me concerne, j'essaye de m'en approcher, mais je considère qu'il est bien plus précieux de conserver la lisibilité et la compréhension de mes règles CSS.

Lorsque je dois cibler un élément de ma page, je commence par repérer son ancêtre principal de structure à l'aide de son identifiant (par exemple : `#wrapper`, `#header`, `#nav`, `#content`, `#footer`), puis j'ajoute directement l'élément à styler (par exemple : `a`, `p`, `strong`, `li`, `.info`). Au final, j'essaye d'obtenir un sélecteur de ce type : `#header > a` ou `#header a`.

Cette règle générale et simple ne convient bien évidemment pas à tous les cas de figure, notamment en situation de mise en page complexe. Par exemple, si plusieurs types de liens cohabitent dans l'ancêtre de structure `#header`, la règle `#header a` ne sera pas suffisante. Il convient dans ce cas de rechercher le second ancêtre de structure des liens à styler, par exemple `#header #nav a`, toujours en évitant les lourdeurs telles que `div#header ul#nav li a`.

C'est tout, rien de très compliqué finalement !

L'intérêt de cette méthode, outre la lisibilité des règles CSS et leur performance, est surtout d'appliquer au départ le moins de poids inutile au niveau des sélecteurs. Car lorsqu'il faut contrer l'une des règles en ajoutant du poids spécifique, on obtient vite des sélecteurs à rallonge dans certaines feuilles de styles complexes.

Utiliser les sprites CSS

La technique dite des *sprites CSS* consiste à regrouper un ensemble de petits éléments visuels dans un même fichier d'image afin d'y faire référence de manière uniforme et d'épargner des requêtes multiples au serveur (figure 3-3). Il s'agit d'une méthode (également appelée « portes coulissantes » dans certains cas) couramment employée pour les éléments décoratifs d'un menu graphique lors du survol des éléments, entre autres.

Figure 3-3

Les sprites CSS de quelques sites célèbres

EN SAVOIR PLUS **Sprites CSS**

L'excellente ressource Pompage.net explique en détail l'intérêt et la mise en application des sprites CSS dans un article traduit de Dave Shea :

- http://www.pompage.net/pompe/sprites/

Cette méthode a ses avantages et ses inconvénients. En termes de performances et d'optimisation de bande passante, elle a fait ses preuves sur des sites web d'envergure tels que Yahoo!, SFR, SNCF, La Poste ou Renault. D'un autre côté, elle n'est pas toujours des plus commodes à mettre en place. En effet, il est nécessaire de calculer au préalable les dimensions des zones à afficher et leurs coordonnées, puis d'exploiter la propriété `background-position` en CSS au pixel près pour « décaler » la vue au bon emplacement.

Dans une astuce publiée sur Alsacreations.com, nous avons déniché deux outils particulièrement utiles dans la mise en œuvre de sprites :

- **CSS Sprite Generator** (figure 3-4) est destiné à vous faciliter la tâche en combinant différentes images fournies dans une archive ZIP en un seul assemblage.

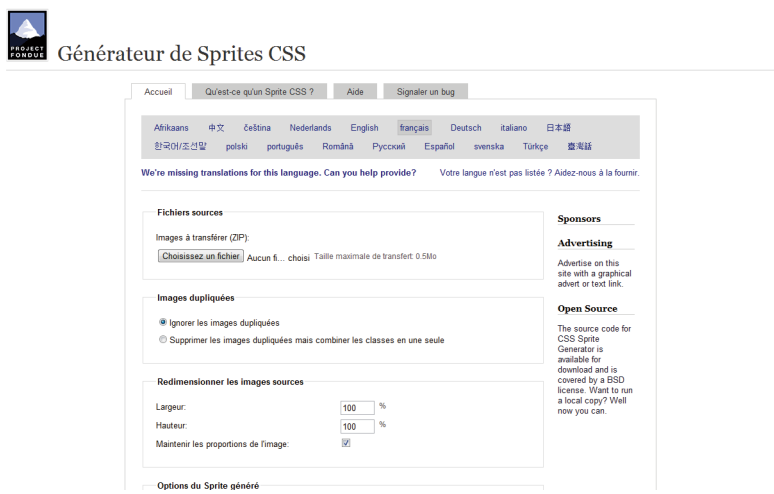
> http://spritegen.website-performance.org

- À l'inverse, **Sprite Creator** simplifie la découpe virtuelle car il suffit de tracer une zone de sélection autour du sprite que l'on souhaite utiliser dans la feuille de styles pour obtenir les instructions CSS.

> http://www.floweringmind.com/sprite-creator

Figure 3-4

CSS Sprite Generator



Optimiser les feuilles de styles

Même en prenant les précautions d'usage et en exploitant au mieux le potentiel des CSS (syntaxes et propriétés raccourcies, conventions communes d'écriture), une feuille de styles peut rapidement se transformer en un fichier long et complexe. Il n'est pas rare de rencontrer des documents de plusieurs dizaines, voire centaines de kilo-octets, ce qui représente une ressource non négligeable à charger pour un agent utilisateur tel qu'un navigateur web.

Si vous œuvrez à un projet d'envergure où l'optimisation de chaque ressource revêt une importance appréciable, il pourra être avantageux de réduire de façon importante la taille de vos fichiers CSS, soit en les réduisant, soit en les compressant.

Réduire la taille des fichiers en éliminant le superflu

Cette opération consiste à réduire autant que possible le fichier en éliminant toutes les portions superflues, telles que les espaces ou caractères inutiles ou les commentaires. On rencontre parfois l'anglicisme « minifier » pour désigner cette opération « d'élagage ».

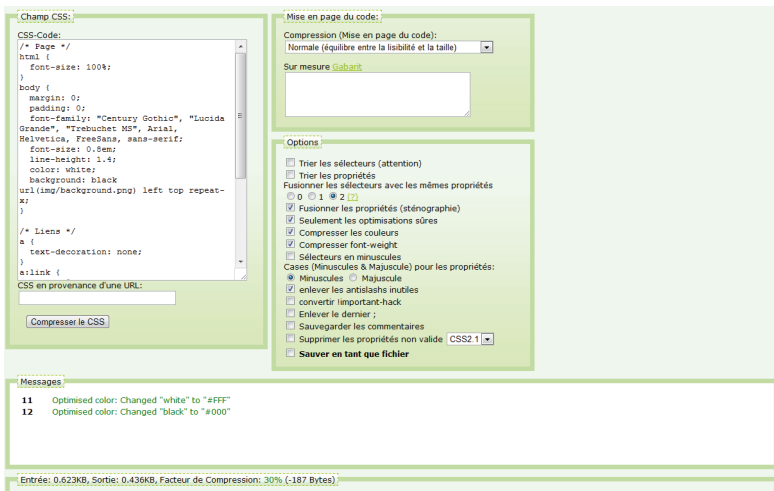
Vous aurez bien compris que ce procédé ne va guère faciliter la relecture des codes dans le contexte d'un projet collaboratif ; cela va même à l'encontre des préceptes de lisibilité exposés précédemment. Selon vos impératifs, vous devrez jongler entre l'optimisation de vos ressources et leur compréhension. Quel que soit votre choix, ne déployez cependant jamais ce genre de techniques avant que votre projet ne soit fini !

L'outil francophone en ligne CleanCSS (http://www.cleancss.com), basé sur CSSTidy, définit très finement les interventions à mener sur le document : fusionner les propriétés, les sélecteurs, enlever les séparateurs ou barres obliques inverses inutiles, compresser les valeurs, supprimer les portions non valides (figure 3-5).

L'ayant testé sur un certain nombre de fichiers CSS, j'ai pu constater un résultat assez probant : au moins 20 % de gain dans le mode de compression classique « privilégier la lisibilité », et parfois près de 50 % dans le mode le plus compact, avec un document complètement illisible à la clé !

Figure 3-5

L'outil en ligne CleanCSS



Plus imposante et polyvalente, l'application Minify (http://code.google.com/p/minify/), hébergée chez Google, est une ressource en PHP 5 permettant de combiner plusieurs fichiers CSS, voire JavaScript, de les élaguer et de les placer en cache dans l'ordinateur pour un chargement plus rapide. Sur la page de présentation de l'application, des taux de gains de bande passante de l'ordre de 70 % sont annoncés.

Si chacun de ces deux outils présente un intérêt évident en termes d'optimisation des ressources, sachez qu'ils ne sont pas toujours fiables à cent pour cent. Il existe un risque non nul pour que votre feuille de styles réduite ne soit plus intégralement compatible avec l'originale et que votre conception s'en trouve altérée ! Dans tous les cas, n'oubliez pas de procéder à une sauvegarde

de sécurité avant de compter tirer parti des programmes de réduction en ligne, surtout si votre feuille de styles comporte des propriétés non valides (`zoom`) ou CSS 3 (`opacity`, `border-radius`, `media queries`, `@media`, etc.), mais également pour retrouver ultérieurement les commentaires utiles laissés sur le fichier original.

Comptant parmi les fichiers les plus imposants de votre dossier web, la taille des images d'illustration et de contenu peut également être réduite. Dans ce domaine, les outils en ligne SmushIt (http://www.smushit.com) de Yahoo! ou PngOptimizer (http://psydk.org/PngOptimizer.php) sont réputés pour leur efficacité. Après avoir transféré un lot d'images, ces dispositifs analysent les données composant les fichiers et suppriment les métadonnées et informations inutiles sans toucher à la qualité de rendu final. Cette étape est sans risque et je vous la conseille vivement avant la mise en ligne de votre projet !

Compresser les fichiers

Vous êtes sans doute familier avec le format de compression ZIP fréquemment employé au quotidien pour les échanges de fichiers.

Le protocole HTTP prévoit lui aussi, depuis 1999, des techniques de compression permettant de compacter les contenus, de les placer en cache et de les délivrer directement à la volée lorsque l'agent utilisateur les demande. Ces méthodes de compression, Gzip ou Deflate, compatible avec tous les navigateurs, peuvent parfaitement être mises en œuvre pour tous les documents web : fichiers HTML, mais aussi CSS, XML ou JavaScript.

Apache est équipé depuis sa version 2.0 du module officiel `mod_deflate`, qui utilise `zlib`, et pour sa version 1.3, de `mod_gzip` ou `mod_deflate`. Selon l'hébergeur, ces modules peuvent être désactivés par défaut, mais il est possible de les actionner dans la configuration générale du serveur si vous y avez accès. Par défaut, `mod_deflate` permet de spécifier les types de fichiers à compresser à la volée grâce à la directive `AddOutputFilterByType DEFLATE`. Une fois ces modules disponibles, vous pouvez également exploiter les fichiers `.htaccess` dans chaque répertoire pour plus de souplesse.

Voici un exemple de contenu du fichier `.htaccess`, dans le répertoire contenant les fichiers CSS et JavaScript (si le serveur est sous Apache 2) :

```
# Apache 2.0
SetOutputFilter DEFLATE
AddOutputFilterByType DEFLATE text/html text/css text/plain text/xml
➥ application/x-javascript
```

Outils en ligne et logiciels

Extensions pour navigateurs

Presque tous les navigateurs contemporains acceptent des petits programmes d'extensions plus connus sous le petit nom de *plug-ins*. Ces additifs optionnels, téléchargeables librement sur les catalogues en ligne des différents navigateurs, se greffent sur le moteur principal et peuvent rendre de fiers services dans le cadre d'un projet de conception de site web.

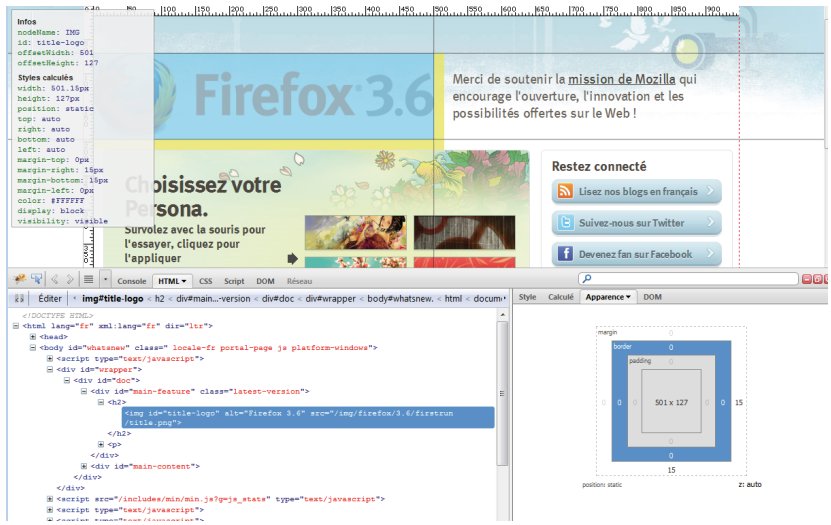
Je vous présente ceux que je considère comme véritablement indispensables dans mon métier. Je n'ai pu les tester que sur mes navigateurs de prédilection (Firefox et Chrome), mais sachez que chacun de ces plug-ins est disponible sur d'autres plateformes telles que Safari, Opera, et même Internet Explorer.

Firebug

Sans conteste l'un des plug-ins les plus appréciés des intégrateurs web, Firebug est un allié de choix dans votre traque des erreurs ou des différences d'affichage entre les navigateurs (figure 3-6). Il permet d'inspecter en détail les codes HTML ou JavaScript de la page en cours, voire de modifier en temps réel les propriétés CSS et l'agencement des différents éléments de la page. Firebug est disponible sur Firefox, mais aussi en version simplifiée (Firebug Lite) sur Chrome, Safari et Opera.

Figure 3-6

Firebug en action



Web Developer

Incontournable extension pour manipuler à loisir une page web en ligne, Web Developer pour Firefox (ou Web Developer Tools pour Chrome) offre une panoplie d'outils pratiques : masquer certains types d'éléments, désactiver les styles CSS, JavaScript, les images ou les couleurs, entourer des éléments HTML définis (tableaux, `div`, titres…), tirer parti de fonctionnalités persistantes, redimensionner en taille personnalisée (1 024 × 600 pixels pour netbooks, par exemple), valider le code HTML ou CSS, etc.

Fireshot

Fireshot est l'une des nombreuses applications de capture d'écran. Selon la configuration de l'outil, il est possible d'enregistrer la partie visible de l'écran, mais aussi de définir une zone

de capture ou toute la page au-delà de la partie visible. Fireshot a été créé exclusivement pour Firefox, mais des équivalents existent sur les autres navigateurs, comme Capture de Page Web pour Chrome.

MeasureIt

Dans un projet de conception web, il est fréquent de devoir mesurer avec précision les différents éléments qui composent une page. MeasureIt est une extension simple et idéale pour visualiser avec précision ces dimensions au pixel près, disponible sur Firefox et sur Chrome (figure 3-7).

Figure 3-7
MeasureIt en action

ColorZilla

Complémentaire de MeasureIt, l'outil ColorZilla permet d'afficher et de conserver dans le presse-papier la couleur de la zone survolée par la souris, ce qui est très pratique pour piocher les multiples teintes composant une page en ligne. ColorZilla n'existe que sur Firefox, mais des équivalents sont disponibles sur Chrome (notamment Eye Dropper).

Dust-Me Selectors

Ce plug-in de Firefox analyse l'ensemble des sélecteurs de styles CSS du document (internes ou externes) afin d'en extraire ceux qui ne sont pas utilisés.

CSS Usage

CSS Usage est une extension de Firebug qui permet de visualiser le nombre de fois que chacune des règles CSS est appliquée, ciblant ainsi les déclarations inutilisées.

CSS Validator

Comme son nom l'indique, cette extension fait contrôler la page en cours par le validateur CSS du W3C.

Yslow

Yslow (prononcez « WaïSlow ») est un outil développé par Yahoo! pour Firefox, destiné à identifier et corriger les problèmes de performance des pages web. Il affiche sous forme graphique les temps de chargement et d'exécution des différents composants du document et pointe du doigt les applications trop coûteuses en ressources ou mal implémentées. Sur Chrome, vous apprécierez sans doute une extension similaire : Speed Tracer.

Outils en ligne

ViewLike.Us

ViewLike.Us n'est pas une extension, mais un service en ligne. Ce site web propose de tester vos pages en les redimensionnant dans les différentes résolutions courantes (de l'iPhone à 1 920 px de large, en passant par des valeurs plus courantes telles que 1 024 × 768).

▸ http://viewlike.us

Le principe est intéressant puisqu'il permet de se projeter sur des supports très diversifiés et de vérifier que l'aspect ne s'en trouve pas détérioré. En revanche, les vues « iPhone » et « WII browser » ne sont que des simulations imparfaites qui se limitent à redimensionner fidèlement l'affichage aux résolutions de ces terminaux. Or, nous le savons, cela ne correspond pas à l'affichage réel obtenu sur ces supports.

WooRank

WooRank est un outil francophone d'analyse en ligne du référencement d'un site web. Il prend en compte un grand nombre de critères tels que la popularité, l'indexation, le ratio entre le contenu et le code, la conformité aux standards, les liens provenant des réseaux sociaux, etc. Le rapport délivré est plutôt complet et permet de mettre le doigt sur un bon nombre de points pouvant être bloquants pour la visibilité de son site web.

▸ http://www.woorank.com

GTmetrix

GTmetrix est un site web anglophone dédié à la vitesse et à la performance des pages, délivrant des rapports établis à partir des références YSlow de Yahoo! et PageSpeed de Google. Comme WooRank, Gtmetrix attribue au final une note globale de performance pour un site testé, mais propose également des solutions concrètes pour optimiser les pages dans de multiples domaines. Je conseille vivement d'y faire un tour lorsque votre projet est finalisé et avant de le présenter à votre client.

▸ http://gtmetrix.com

IETester

Le quotidien de notre métier de concepteur web est de traquer et d'éradiquer les différences d'affichage entre les navigateurs. Dans la majorité des cas, ces écarts sont dus aux anciennes versions du navigateur de Microsoft, à savoir IE6 et IE7. Or, par défaut, il n'est pas possible d'installer plusieurs versions d'Internet Explorer sur un même poste.

C'est là qu'intervient IETester, que l'on peut considérer comme un navigateur web ayant la capacité dans le même processus d'afficher les pages sous la forme d'onglets représentant IE5.5, IE6, IE7, IE8 et même IE9 !

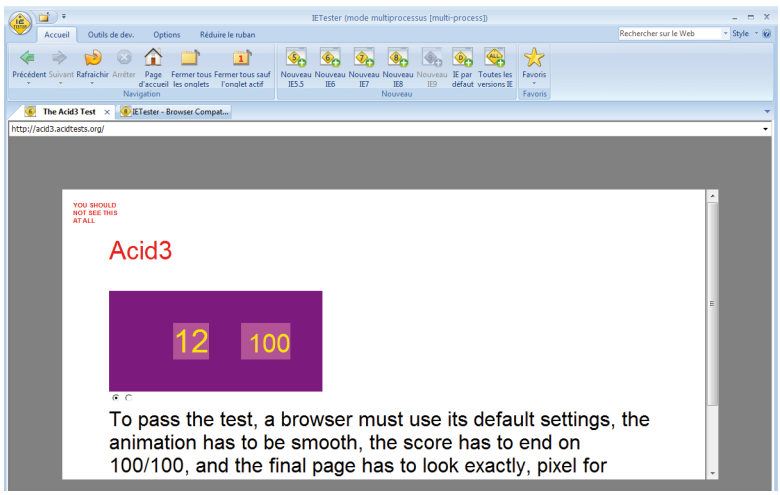
IETester est un logiciel compatible avec Windows XP, Vista et Windows 7 et reconnaît, contrairement à d'autres, les commentaires conditionnels.

Signalons toutefois quelques petites erreurs à l'usage : l'écran clignote au redimensionnement, certaines fonctionnalités sont bridées si vous n'êtes pas administrateur de votre poste et, plus gênant, certains onglets font « planter » le logiciel au bout de quelques minutes d'utilisation.

Dans l'ensemble, IETester est un très bon outil, indispensable pour vos tests qualité sur de multiples navigateurs (figure 3-8).

Figure 3-8

Illustration de IETester



Grilles de mise en page

Concevoir sur la base de grilles est le lot quotidien des agences de communication dédiées au support d'impression papier. Les logiciels dédiés à l'infographie comportent tous des règles graduées et des grilles magnétiques sur lesquelles les éléments des pages peuvent – ou doivent – s'aligner tels des aimants.

Parfois perçues comme une contrainte, les grilles de mise en page peuvent au contraire servir de patron ou de guide pour agencer harmonieusement les différents éléments de la page, tout en étant conscient que le Web est fluide et que les contenus et la taille de la police de texte servis à l'utilisateur final pourront varier.

Par commodité et pour offrir la plus grande variété possible de colonnes, de nombreux systèmes de grilles actuels pour le Web sont basés sur une largeur de 960 px : non seulement cette dimension est idéale pour la plupart des écrans (dont la résolution minimale est 1 024 px), mais en outre, cette valeur est susceptible de contenir uniformément un grand nombre de colonnes. Pour

être très précis, 960 est divisible par 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 32, 40, 48, 60, 64, 80, 96, 120, 160, 192, 240, 320 et 480 !

Observez de plus près vos sites Internet préférés, ceux dont l'aspect vous semble harmonieux : aucun des éléments n'est disposé au hasard, ni ne flotte sans influence avec son entourage. Dans leur très grande majorité, les sites web professionnels réussis se réfèrent à une grille de mise en page… et je ne vais pas leur donner tort car nous sommes de fervents adeptes de cette technique (figure 3-9).

Figure 3-9

Une grille de 960 px

De nombreux outils en ligne, la plupart en anglais, facilitent l'élaboration de telles grilles. En voici quelques uns :

- http://www.1kbgrid.com
- http://grid.mindplay.dk
- http://netprotozo.com/grid/

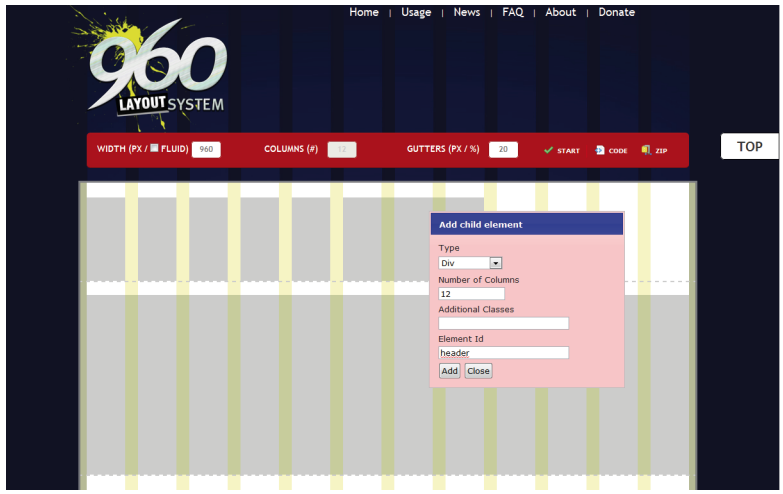
- http://www.designbygrid.com/tools
- http://spry-soft.com/grids
- http://www.puidokas.com/portfolio/gridfox/

Frameworks CSS

Si vous travaillez en équipe, les frameworks CSS – les plus célèbres d'entre eux étant 960.gs et Blueprint – sont des outils fournissant des briques de base pour l'intégration HTML et CSS (figure 3-10).

Figure 3-10

Le constructeur de gabarit de 960.gs



Plutôt que de tout reprendre à zéro à chaque projet, ces environnements définissent un code CSS standardisé, normalisé et surtout réutilisable. L'efficacité de ces outils est loin d'être négligeable, car convertir les tâches répétitives en blocs génériques réemployables est un réel gain de temps et réduit les coûts de production.

Le principe global est toujours de se référer à une grille divisée en colonnes (habituellement 12, 16 ou 24), puis de nommer les différents composants de cette grille à l'aide d'une valeur de classe correspondant à leur dimension.

Nommage HTML avec le framework 960.gs

```
<div class="container_16"> <!-- bloc conteneur principal pour une grille
en 16 colonnes -->
  <p class="grid_4">Mon texte de remplissage</p> <!-- bloc interne occupant 4 colonnes
  -->
  <p class="grid_4">Mon texte de remplissage</p>
  <p class="grid_4">Mon texte de remplissage</p>
```

```
    <p class="grid_4">Mon texte de remplissage</p>
    <!-- la somme des "grid" doit correspondre au total du "container" -->
</div>  <!-- on ferme "container" -->
```

La partie CSS indiquera que les quatre paragraphes devront se positionner les uns à côté des autres à l'aide de la propriété `float`.

Si l'on veut placer d'autres blocs en dessous, on ajoutera un saut de ligne matérialisé par :

```
<div class="clear"> </div>
```

Comme le dit Bruno Bichet, auteur du blog Css4design : « Il existe de nombreuses raisons pour ne pas utiliser de frameworks CSS lors de la phase de conception ou d'intégration de votre site web : le poids, l'intrusion des classes CSS dans le code HTML qui devient la règle pour placer le moindre bloc de contenu, le fait que de nombreuses règles ne sont pas utilisées, etc. » *(Source :* http://www.css4design.com*).*

Pour ma part, je ne peux que confirmer ces réticences envers ces outils un peu trop génériques. Dans notre quête de la conception web « propre », un framework pourrait s'apparenter à un logiciel Wysiwyg (*What You See Is What You Get*) tel que Dreamweaver : employé de façon consciencieuse, il peut être tout à fait performant, mais dans bien des cas il demeure une « usine à gaz » lourde supposée couvrir des besoins multiples, donc pas forcément adaptée à chacun de vos projets et toujours nantie de fonctionnalités et codes inutiles. Il ne faut pas non plus oublier que l'on a affaire à une couche supplémentaire qui nécessite un apprentissage et une appropriation préalables.

OUTILS Frameworks CSS spécialisés

Parallèlement aux usines à gaz, d'autres outils plus légers, car plus spécialisés, gravitent dans le monde des frameworks CSS. Selon les besoins et les projets, il est parfois plus avantageux de tirer parti de ces solutions :

- Formee : exclusivement destiné à la conception et à la mise en forme de formulaires. Personnalisable et flexible.

▸ http://formee.org

- Tripoli : dédié à la typographie, cet outil redéfinit toutes les valeurs des textes par défaut, incluant des douzaines d'éléments (tables, listes, typographie, mais aussi les en-têtes (h1 - h6), les abréviations, les citations et les formulaires.

▸ http://devkick.com/lab/tripoli/

Zen Coding

Zen Coding est une méthode d'écriture favorisant la composition rapide d'un code HTML à l'aide d'une syntaxe extrêmement raccourcie.

Elle se présente sous la forme d'une extension destinée aux logiciels éditeurs tels que Notepad++, Textmate, Coda ou Aptana, que l'on peut télécharger chez Google (http://code.google.

com/p/zen-coding/). Une démonstration interactive est consultable sur la page suivante : http://zen-coding.ru/demo/.

La notation Zen Coding, bien que rédigée dans un document de type HTML (ou XHTML ou XML), est relativement intuitive puisque basée sur les sélecteurs CSS. Il conviendra donc de bien connaître les sélecteurs classiques indispensables de classe, d'identifiant, de parenté, d'adjacence et d'attribut avant de se lancer dans la découverte de cette approche.

Principe d'expansion

Concrètement, c'est assez simple : il suffit de « penser » CSS pour écrire son code HTML. Par exemple, la syntaxe `div#sidebar` deviendra automatiquement `<div id="sidebar"></div>` après expansion via Zen Coding grâce à des raccourcis clavier tels que `Ctrl + E` pour Notepad++ par exemple.

Outre l'accélération des processus de production, la syntaxe de Zen Coding permet de créer et de conserver facilement des morceaux de code (*snippets*) réutilisables et injectables dans plusieurs types de documents HTML ou à transmettre à des collègues de travail.

Voici quelques fonctionnalités et règles spécifiques à cette méthode :

- `balise#name` et `balise.name` construisent des éléments dotés d'un `id` ou d'une `class` de valeur `name` ;
- `balise1>balise2` construit une imbrication (l'élément `balise2` sera enfant de `balise1`) ;
- `balise1+balise2` construit une fratrie (l'élément `balise2` sera frère adjacent de `balise1`) ;
- `balise[attr="valeur"]` construit un élément doté d'un attribut et d'une valeur définis ;
- `balise*3` construit trois éléments frères identiques ;
- `balise.name$*3` construit trois éléments frères identiques dont les classes seront respectivement `name1`, `name2` et `name3` ;
- `html:xs`, `html:xt` et `html:5` construisent l'ensemble de la structure d'un document HTML incluant le Doctype (XHTML strict, XHTML transitionnel ou HTML 5), les éléments `<html>`, `<title>` et `<body>`. Par exemple `html:xs>div#main` va créer un document XHTML strict complet contenant un élément `<div>` avec l'`id main`.

Je vous propose d'illustrer le fonctionnement pratique de ce procédé au travers des trois exemples suivants : la construction d'un tableau de données, l'élaboration d'une structure HTML 5 complète et d'un formulaire de contact.

Construction de tableau de données

```
table[summary="résumé du tableau"]>(caption+(tr>th*3)+(tr*2>td*3))
```

Résultat après expansion

```
<table summary="résumé du tableau">
  <caption></caption>
  <tr>
    <th></th>
    <th></th>
```

```
    <th></th>
  </tr>
  <tr>
    <td></td>
    <td></td>
    <td></td>
  </tr>
  <tr>
    <td></td>
    <td></td>
    <td></td>
  </tr>
</table>
```

Une structure complète en HTML 5

```
html:5>(header+(#sidebar>nav>ul>li*5>a)+#main+footer)
```

Résultat après expansion :

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <header></header><div id="sidebar">
    <nav>
      <ul>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
      </ul>
    </nav>
  </div><div id="main"></div><footer></footer>
</body>
</html>
```

Un petit formulaire pour finir

```
form#contact[action="destination.php"]>((p*2>(label[for]+input[type="text"][id]))
+(p>(label[for]+textarea))+p>input[type="submit"][value="send"])
```

Résultat après expansion

```
<form action="destination.php" id="contact">
  <p><label for=""></label><input type="text" id="" /></p>
  <p><label for=""></label><input type="text" id="" /></p>
```

```
  <p><label for=""></label><textarea name="" id="" cols="30" rows="10"></textarea></p>
  <p><input type="submit" value="send" /></p>
</form>
```

Appliquer un masque

Une autre fonctionnalité pratique de Zen Coding consiste en l'habillage de contenu selon un « masque » défini (menu Wrap with abbreviation ou Ctrl+Shift+A sur Notepad++).

Il est ainsi possible d'appliquer le masque `ul#nav>li*>a` sur la partie de contenu suivante :

```
Accueil
Société
Produits
Contact
```

… pour obtenir le code HTML suivant :

```
<ul id="nav">
  <li><a href="">Accueil</a></li>
  <li><a href="">Société</a></li>
  <li><a href="">Produits</a></li>
  <li><a href="">Contact</a></li>
</ul>
```

Zen Coding et CSS

La méthodologie d'écriture de Zen Coding ne se limite pas au langage HTML, mais prévoit une large batterie de syntaxes raccourcies pour les styles CSS, toutes versions confondues.

L'ensemble des propriétés apparaît dans la liste de l'outil, mais aussi les règles @ ou la fonction `!important`. Voici quelques exemples courants :

- La syntaxe « ! » devient `!important`.
- `@i` se transforme en `@import`.
- `m` et `p` deviennent respectivement `margin` et `padding`.
- `fl` est changé en `float`.
- `w` et `h` deviennent `width` et `height`, etc.

La liste complète est proposée à cette adresse :

> http://code.google.com/p/zen-coding/wiki/ZenCSSPropertiesEn

Étendre le langage CSS : Less

La principale lacune de CSS est… sa simplicité : à trop vouloir en faire un langage intuitif et facile à interpréter, il en devient répétitif, fastidieux et parfois réducteur.

Less est un projet basé sur le langage Ruby et peut s'apparenter à une extension du langage CSS, dans le but de lui apporter les fonctionnalités qui lui manquent parfois cruellement, telles que la notion de variables, les opérations de calculs dynamiques, la factorisation de parties de code ou encore les imbrications de sélecteurs.

▸ vhttp://lesscss.org

Dans sa version de base, le code Less crée des fichiers CSS après avoir été interprété et compilé via un serveur en Ruby, mais il existe une version exploitable via PHP. Il s'agit d'un projet jeune issu de groupes de travail de développeurs, c'est pourquoi il demeure encore quelque peu ardu à mettre en œuvre et nécessite un minimum de connaissances en langages et administration de serveur.

▸ http://leafo.net/lessphp/

Toutefois, avec un peu de pratique, on en vient à se demander pourquoi un langage tel que CSS ne propose pas encore un mode d'emploi aussi logique et productif que celui de la surcouche Less.

Concrètement, voici comment se présente ce métalangage Less :

```
@nice-blue: #5B83AD;
@light-blue: @nice-blue + #111;
.rounded {
  border-radius: 10px;
  padding: 5px;
}
h1 {
  color: @nice-blue;
  background: @light-blue;
}
#menu a {
  color: @nice-blue;
  .rounded;
  :hover { text-decoration: none; }
}
```

Exploiter son éditeur HTML

Pour finir sur cette section dédiée aux gains de productivité, rappelons que la première bonne pratique à observer reste de s'approprier parfaitement son outil de travail au quotidien, à commencer par l'éditeur de texte.

Quel que soit le logiciel de développement choisi, chacun de nous vit sa propre expérience en tant qu'utilisateur. Certains vont se contenter des fonctionnalités de rédaction de base, d'autres vont explorer les entrailles de l'éditeur afin de l'adapter à des besoins spécifiques. Certains

réglages favorisant la tâche, disponibles sur une majorité d'outils, me paraissent suffisamment pertinents pour être mentionnés dans cet ouvrage.

OUTILS Quelques éditeurs efficaces

Il n'est pas toujours évident de faire son choix parmi le large panel d'éditeurs HTML, gratuits ou non, sur le marché. Au cours de mon expérience d'intégrateur, j'ai eu l'occasion d'en tester un bon nombre, parmi lesquels certains ont retenu mon attention : Notepad++, HTMLkit, PSPad, Aptana Studio, Eclipse, jEdit, et encore E-Texteditor (celui que j'affectionne tout particulièrement).

Pour les *aficionados* des outils Wysiwyg, ma liste de choix comporte le célébrissime Dreamweaver, très efficace si maîtrisé, mais aussi BlueGriffon, développé par Daniel Glazman, l'un des membres influents du groupe de travail CSS du W3C. BlueGriffon est à la fois léger et performant, ses fonctionnalités n'ayant rien à envier aux ténors : éditeur HTML et CSS, transfert FTP, colorisation, nettoyeur de code, onglets, etc.

▸ http://www.bluegriffon.org

Réservés à Mac OS, Smultron (gratuit) et Coda sont actuellement plébiscités pour leurs nombreuses fonctionnalités et leur ergonomie. Entièrement configurable, Textmate conviendra aux amateurs de raccourcis clavier.

Sur un environnement GNU/Linux, les outils d'édition les plus connus sont Quanta Plus, Bluefish, eMacs et VIM.

Macros et automatisations

L'un des principaux avantages des syntaxes telles que Zen Coding est de pouvoir être réutilisées. Dans cette optique, il devient particulièrement intéressant de conserver certains morceaux de codes types dans une sorte de presse-papier et de pouvoir les piocher selon ses besoins.

De nombreux éditeurs HTML autorisent l'usage de macros, voire de plug-ins tels que Quicktext ou AutoHotkey. L'un et l'autre permettent d'enregistrer des séquences préétablies que l'on peut rejouer à l'aide d'une combinaison de touches du clavier.

Pour ma part, j'ai enregistré de courtes macros sur Notepad++ afin d'affecter certains raccourcis clavier à la création automatique de bouts de code : une structure globale d'un document XHTML Strict ou HTML 5, un menu de navigation à cinq éléments, ou encore un formulaire de contact.

Une alternative très intéressante est proposée par les éditeurs Textmate (Mac OS) et son clone E-Texteditor (Windows) : entièrement configurables, ces logiciels sont fondés sur le principe de *bundles*, qui sont des raccourcis clavier évolués et intelligents. Il est ainsi possible d'effectuer un grand nombre de tâches répétitives à l'aide de mots-clés prédéfinis et d'activer les bundles à l'aide de la simple touche Tab.

Réglages personnalisés

Chaque éditeur de texte se distingue de ses concurrents par son ergonomie propre et ses fonctionnalités spécifiques. Pour en avoir testé un certain nombre, je peux en conclure qu'aucun n'est vraiment parfait, mais que chacun a ses avantages et ses inconvénients. L'important est que vous soyez parfaitement à l'aise avec votre outil.

Quel que soit votre éditeur de prédilection, apprenez-en toutes ses facettes cachées, car ils proposent le plus souvent des fonctionnalités bien pratiques au quotidien et il serait dommage de s'en priver.

La *coloration syntaxique* est un grand classique que l'on trouve sur quasi tous les éditeurs. Bénéficier d'une couleur distincte pour les éléments, les attributs, les valeurs et les propriétés CSS facilite grandement la lecture de vos fichiers. Sachez qu'il est souvent possible de redéfinir la coloration par défaut et d'enregistrer vos propres thèmes graphiques.

Autre fonction traditionnelle, l'*autocomplétion* est la faculté de proposer automatiquement une liste de propriétés ou valeurs pertinentes à partir de ce que l'utilisateur a commencé à taper au clavier. C'est un mécanisme très pratique pour des intégrateurs néophytes, mais qui peut être rapidement gênant si vous êtes un briscard de la conception web. Il est alors indiqué de purement et simplement désactiver l'outil.

Enfin, au cas par cas, d'autres fonctionnalités agrémentent les différents éditeurs de texte et peuvent faire la différence selon les projets ou – plus simplement – vos goûts : vues multiples de fichiers (afficher le code HTML à gauche et la feuille CSS à droite), duplication de lignes de code à l'aide d'un simple raccourci clavier, ou encore outil de transfert FTP intégré ou de gestion de fichiers par projets.

Checklist générale

Ce chapitre dédié aux bonnes pratiques, à la bonne gestion d'un projet HTML/CSS et à l'optimisation des ressources a pour but de fournir des pistes ou des méthodes pour vous faciliter la tâche au quotidien, gagner en performance et en temps de production lors de vos projets collaboratifs. Pour le conclure, je vous propose de synthétiser les différents conseils prodigués tout au long de cette section dans un tableau récapitulatif, loin d'être exhaustif.

Tableau 3-1 Vérification des étapes de votre projet

Liste des tâches à vérifier	Validation
Identifier les éléments HTML selon leur fonction *(titraille, paragraphes, listes, formulaires, citations, images/arrière-plans, tableaux de données, code informatique).*	
Regrouper les éléments par blocs de structure (`<div id="header">`, `<div id="main">`, `<div id="footer">`, `<ul id="nav">`, `<form id="contact">`).	
Appliquer un rendu de type bloc aux nouveaux éléments HTML 5 (`<header>`, `<footer>`, `<nav>`, `<aside>`, `<section>`...).	
Appliquer une feuille de styles de base (*Reset*) maîtrisée *(suppression des marges sur* `html`, `body`, `form`, `ul`, `ol` *et* `blockquote`, *styler les liens de la page, la typographie ; regrouper les styles identiques).*	
Commenter les documents HTML et CSS en adoptant une nomenclature constante.	
Optimiser les fichiers HTML et CSS (compresser, réduire), ainsi que les images.	

Liste des tâches à vérifier	Validation
Renseigner les attributs HTML `width` et `height` des images pour éviter des calculs inutiles au navigateur.	
Employer les propriétés CSS raccourcies (`font`, `border`, `background`, `margin`…) par souci d'optimisation de la taille du fichier de styles.	
Se servir d'une grille de mise en page.	
Utiliser des sélecteurs performants en termes de ressources et réutilisables (pas de spécificité forte).	
Vérifier que tous les liens externes et internes sont valides.	
Définir une feuille de styles alternative spéciale Internet Explorer pour pallier à ses déficiences (`min-width` → `width` *sur IE6,* `min-height` → `height` *sur IE6,* `inline-block` → `inline` + `zoom 1` *sur IE6/7…).*	
Tester les pages web sur différents types de navigateurs et de systèmes d'exploitation.	
Tester les pages web en désactivant les CSS, les images, JavaScript, Flash ou autre plug-in.	
Tester les pages en agrandissant la taille du texte de contenu.	
Valider les fichiers HTML.	
Valider les fichiers CSS.	
Ne pas oublier la page 404, le plan du site, la favicon.	

4

Le positionnement en CSS

Ce chapitre a pour but de consolider et d'élargir vos connaissances des différents schémas de positionnement classiques actuellement utilisés : le positionnement en flux, le positionnement absolu, fixé ou relatif et le positionnement flottant.

Histoire du positionnement en CSS

Lorsque naît le Web en 1994, sa portée est tout d'abord extrêmement limitée et essentiellement destinée à un monde d'étudiants ou de scientifiques. Adaptées à des documents de type principalement textuel, où les éléments sont disposés de façon linéaire, empilés les uns sur les autres (titres, paragraphes, listes), les CSS sont dépassées dès leur apparition par un Web en pleine ébullition : gigantesque marché naissant, Internet ne pouvait se réduire à de simples pages textuelles et frustrer ainsi tous les graphistes en quête de l'aspect le plus vendeur pour leur vitrine.

Entre tableaux, cadres et calques

Blocs contigus, positionnements originaux, empilements, grilles et architectures complexes, les besoins des concepteurs web ne peuvent être assouvis par une technologie née obsolète. Ils se rabattent sur des mécanismes qui apportent des réponses concrètes à leurs fantaisies graphiques :

- les tableaux de mise en page (balise HTML `<table>`) ;
- les cadres (*frames*) ;
- les calques (positionnement absolu).

Quand le W3C introduit l'élément `<table>` dans le standard HTML 3.2, celui-ci est déjà largement utilisé par les navigateurs Netscape et Explorer, qui n'ont pas attendu un standard pour répondre aux besoins des concepteurs (figure 4-1).

Figure 4-1

Mise en page à l'aide de tableaux (chaque encadré de couleur correspond à une cellule)

Il en est de même pour les cadres et les calques, tout d'abord conçus par les constructeurs de navigateurs, puis introduits officiellement dans les spécifications dans l'optique de répondre au besoin des concepteurs web employant les tableaux de mise en page à mauvais escient. Cependant, les problèmes rencontrés deviennent assez rapidement gênants : mauvais référencement, entrave aux retours en page précédente, mise en favoris impossible, accessibilité réduite, etc.

Les logiciels Wysiwyg, comme Dreamweaver, Frontpage ou Golive, créent leurs propres calques, d'abord avec l'élément `<layer>`, puis avec l'élément `<div>` auquel ils ajoutent automatiquement une multitude de propriétés contraignantes et souvent inutiles (`position: absolute`, `z-index`, `name`, `top`, `left`…). Dreamweaver emploie même durant quelques versions l'élément `<span>` comme un calque, alors qu'il s'agit d'un élément de type `inline` pas du tout destiné à structurer des blocs !

Le Web « visuel » est né, enterrant le Web « universel » et accessible à tous.

Flottement et retour à la « sémantique »

Bien que les tableaux, les cadres et le positionnement absolu offrent des solutions fonctionnelles aux problèmes concrets de mise en page du quotidien, une scission commence à s'opérer entre les graphistes « purs » et les défenseurs de l'universalité du Web après les années 2000.

Le navigateur dominant de l'époque ne supportant guère les techniques contemporaines de positionnement, une syntaxe issue de CSS 1 sort peu à peu du lot et va devenir – à contre-emploi – massivement utilisée pour la mise en page CSS globale : la propriété `float`.

Malgré les nombreux problèmes d'affichage des navigateurs et ses diverses contraintes, ce que l'on nomme le *positionnement flottant* devient le nouvel Eldorado des évangélistes férus de « sémantique », car il permet de s'affranchir de la mise en page par tableaux, de conserver la fonction des éléments HTML et de séparer complètement le contenu de la mise en forme.

À l'heure actuelle, le schéma de positionnement basé sur la propriété `float` demeure le plus conseillé et le plus usité par les prêcheurs de la bonne parole d'un Web accessible à tous. Nous allons toutefois revoir en détail les autres méthodes de placement des éléments, tels que le positionnement absolu, relatif et fixé. Très souvent mal comprises et mal employées, ces techniques offrent certains avantages au cas par cas.

Modèle de boîte

Anatomie d'une boîte

Selon les spécifications W3C, tout élément structuré par une balise HTML (ou plusieurs) se représente sous forme d'une « boîte » rectangulaire (figure 4-2) définie par diverses composantes que sont :

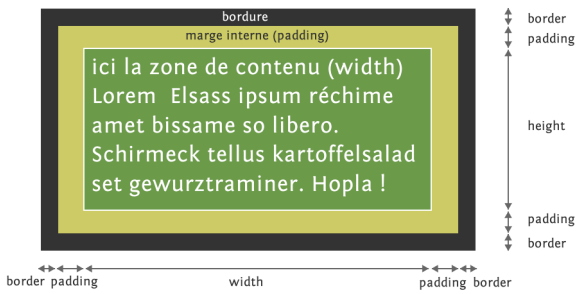
- la zone de son contenu, représentée par une largeur et une hauteur : en CSS, il s'agit des propriétés `width` et `height` ;
- un espace réservé à la bordure de la boîte (propriété `border`) ;
- une marge interne à la boîte (`padding`), se situant entre la zone de contenu et la bordure.

Une dernière composante représente la marge externe (`margin`) et se situe hors de la boîte, au-delà de l'espace alloué à la bordure. Elle affecte le positionnement de l'élément puisqu'elle pousse sa boîte ou ses sœurs environnantes.

Toutes ces composantes doivent être prises en considération : contrairement à ce que beaucoup pensent, un élément n'occupe pas uniquement l'espace déterminé par sa valeur `width`, mais aussi celui de ses marges internes (`padding`) et ses bordures.

Figure 4-2

Modèle de boîte



Dimensions des éléments

Le calcul de la largeur « réelle » d'un élément est souvent mal appréhendé par les débutants, car il peut sembler illogique au premier abord. En réalité, c'est le terme de `width` qui est source de confusion, car il ne désigne pas la largeur de la boîte entière, mais uniquement celle de la partie de contenu. Le terme de *content-width* ou *inner-width* eût été moins trompeur.

Avec un peu d'expérience, jongler avec ces composantes devient très vite intuitif et les erreurs de dimensions se dissipent.

Prenons par exemple un paragraphe (élément `<p>`) disposant des déclarations CSS suivantes :

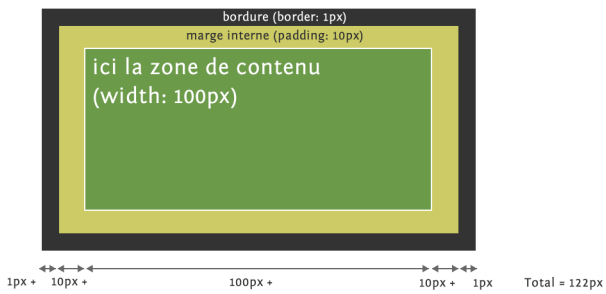
```
p {
  width: 100px;
  padding: 10px;
  margin: 5px;
  border: 1px solid #000;
  background-color: yellow;
}
```

La largeur de la boîte de paragraphe (la surface occupée par l'arrière-plan jaune) est dans cet exemple de… 122 pixels. L'aviez-vous trouvé ?

En effet, les composantes à prendre en compte sont `width`, `padding` et `border`. La marge externe (`margin`) n'entre pas dans le calcul de la largeur de l'élément. La propriété `padding` exprime les marges tout autour de l'élément, il faut donc la compter de chaque côté. Il en est de même pour la propriété `border`. Au final, nous additionnons 100 px (`width`) + 20 px (`padding`) + 2 px (`border`) pour parvenir à une largeur réelle de 122 pixels (figure 4-3).

Figure 4-3

Une boîte de 122 pixels de large



BONNE PRATIQUE **Cumulez avec discernement**

Puisque plusieurs facteurs entrent en jeu dans le calcul des dimensions des éléments HTML, une sage précaution est de s'abstenir de les appliquer sans discernement sur un même bloc. Ainsi, je préconise souvent d'éviter de cumuler les propriétés `width` et `padding` (ou `width` et `border`) pour éviter de mauvaises surprises.

Dans ce registre, je croise parfois des déclarations telles que `width: 100%`, inutiles dans la majorité des cas, puisqu'un bloc occupe toujours par défaut toute la largeur de son parent. Le problème devient réellement gênant lorsque l'on souhaite appliquer un `padding` à cet élément, dont la dimension sera alors supérieure à 100 % de son conteneur !

Minima et maxima

Une bonne pratique de la conception web accessible à tous est de favoriser autant que possible la fluidité des éléments. Souvent, on leur affecte une hauteur (propriété `height`) en espérant « figer » leur aspect comme on le ferait sur le support d'impression. C'est un leurre puisque le contenu de l'élément demeure libre alors de dépasser de sa boîte. Or, nous ne pouvons pas dompter la taille des textes que le visiteur va définir selon ses besoins et envies. Évitez par conséquent de fixer une hauteur, sauf s'il s'agit d'un élément dont vous maîtrisez parfaitement les dimensions et le contenu.

Dans ce domaine, quelques propriétés CSS 2 méconnues sont souvent particulièrement appropriées. Il s'agit de :

- `min-width` : largeur minimale (en pixels, pourcentage, `em`...) ;
- `max-width` : largeur maximale ;
- `min-height` : hauteur minimale ;
- `max-height` : hauteur maximale.

Reconnues depuis Internet Explorer 7, les propriétés sus-citées ne figent pas les dimensions de l'élément, à la différence des classiques `height` et `width`, mais se contentent de définir une valeur minimale ou maximale, quel que soit le contenu présent.

Il est d'ailleurs possible de cumuler plusieurs de ces propriétés comme le montre cette règle :

```
body {
  width: 80%;
  min-width: 800px;
  max-width: 1200px;
}
```

Cet exemple regroupe un ensemble de propriétés affectant l'élément `<body>` (et par extension, la page web) : sa largeur est fluide et occupe toujours 80 % de la taille de l'écran, tout en fixant un minimum de 800 pixels et un maximum de 1 200 pixels. Ces jalons évitent que l'affichage soit dégradé sur les petits écrans ou que le contenu, trop large, soit pénible à lire sur les très grandes surfaces.

BONNE PRATIQUE **Max-width à la rescousse des gestionnaires de contenus**

La propriété `max-width` est particulièrement utile pour la gestion des contenus récalcitrants : plutôt que de limiter la largeur du bloc parent, nous pouvons nous en servir pour restreindre les dimensions des éléments internes de contenus. L'avantage est que cette propriété peut s'appliquer sur des éléments tels que des images, des champs de formulaires (`<input>`, `<select>`, `<textarea>`) ou des tableaux de données (`<table>`, `<td>`, `<th>`).

Si une zone de contenu occupe 600 pixels de large, il est possible de restreindre l'affichage d'une grande image à une largeur maximale de 600 px :

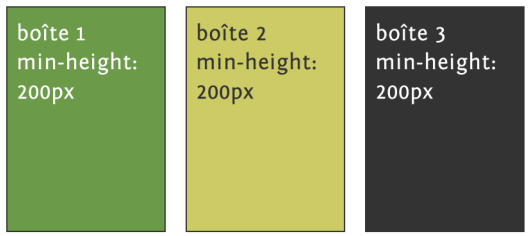
```
p img { max-width: 600px; /* les images des paragraphes seront limitées à 600px
de large */ }
```

La propriété `max-width` est une excellente rustine pour de nombreux éléments dont le contenu n'est pas forcément maîtrisé dans toutes les situations. Ainsi, la déclaration `max-width: 100%` restreint l'élément désigné à ne pas déborder de son parent.

Une autre application judicieuse de ces propriétés est de se servir de la hauteur minimale (`min-height`) d'un élément pour simuler les blocs contigus ayant la même hauteur, tout en demeurant extensibles en hauteur selon leur contenu (figure 4-4).

Figure 4-4

Blocs de même hauteur minimale, mais demeurant étirables en hauteur



Le mode Quirks de Microsoft

Le modèle de boîte proposé par le W3C est adopté par l'ensemble des navigateurs du marché, pour le plus grand bonheur des concepteurs web que nous sommes. Sachez cependant que ce ne fut pas toujours le cas, puisque du temps de l'apogée d'Internet Explorer 5.0 et 5.5, Microsoft avait décidé d'adopter un modèle de boîte différent de celui proposé par le Consortium. Ce modèle de boîte assimilait les composantes de marges internes (`padding`) et les bordures (`border`) à la zone de contenu (`width`).

Reprenons notre paragraphe précédent et les styles associés :

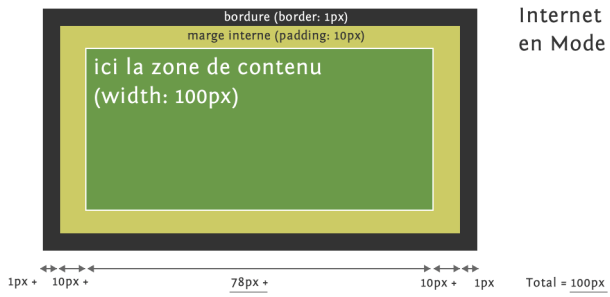
```
p {
  width: 100px;
  padding: 10px;
  margin: 5px;
  border: 1px solid #000;
```

```
    background-color: yellow;
}
```

La largeur de la boîte de paragraphe dans ce modèle Microsoft sera tout simplement de 100 pixels. Cela signifie que la zone de contenu sera réduite à 78 pixels (figure 4-5).

Figure 4-5

Mêmes composantes, largeur finale différente



Ce mode de calcul, où les propriétés visibles (espaces et bords) sont incluses dans le calcul de la boîte, est appelé mode *Quirks*. C'est un modèle de boîte ancien et spécifique à Microsoft. Fort heureusement, depuis la version IE6, ce mécanisme propriétaire disparaît au profit du modèle de boîte dit *Standard*, où l'interprétation des dimensions des boîtes est la même que pour les autres navigateurs.

Il demeure cependant certains cas de figure particuliers qui font passer IE6 et IE7 en mode Quirks. Ces cas concernent le Doctype de la page : un document antérieur à HTML 4, ou sans Doctype, ou qui contient n'importe quel caractère avant le Doctype, fera basculer Internet Explorer en mode Quirks avec son lot d'interprétations erronées des dimensions des boîtes, ce qui pose souvent de lourds problèmes de compatibilité (figure 4-6).

Figure 4-6

Avec ou sans Doctype ?

Retenez simplement que s'il n'y avait qu'une seule bonne raison d'ajouter un Doctype à vos pages HTML et de les rendre valides, ce serait sans aucun doute celle de se soustraire au mode Quirks de Microsoft.

Valeurs calculées et box-sizing en CSS 3

CSS 3 introduit la notion de calcul dans les valeurs via la fonction `calc()` adoptée dans Firefox 4 (sous réserve de préfixe `-moz-`) et Internet Explorer 9, qui accepte des opérateurs tels que l'addition, la soustraction, la multiplication, la division et le modulo, ainsi que le panachage des unités. Il devient possible d'indiquer une valeur de la sorte : `width: calc(100% - 20px)`.

Longtemps rêvée par les intégrateurs, cette fonction avancée est toutefois à prendre avec des pincettes car dans la majorité des cas, elle devrait être avantageusement remplacée par un schéma de positionnement moderne, fluide et bien pensé. En clair, il s'agit plutôt d'une « béquille » à employer au cas par cas.

Une autre propriété CSS 3, `box-sizing`, inclut les composantes de `padding` et `border` à l'intérieur de la boîte. Elles ne s'ajoutent plus à la largeur générale de l'élément ; la partie de contenu (`width`) en est par conséquent réduite.

Reconnue depuis Internet Explorer 8 ainsi que sur Opera 9, Mozilla et WebKit (moyennant les habituels préfixes vendeur `-moz-` et `-webkit-` pour ces derniers), la propriété `box-sizing` offre ni plus ni moins la possibilité à tous les navigateurs d'afficher les dimensions des éléments selon le modèle de boîte Quirks. Il s'agit d'une technique puissante et risquée modifiant tout le comportement des éléments ; je ne la recommande que dans des cas très particuliers où d'autres méthodes échoueraient.

Exercice pratique : centrer horizontalement en CSS

Dès les premières moutures du langage CSS, les alignements horizontaux sont pris en compte au sein des spécifications.

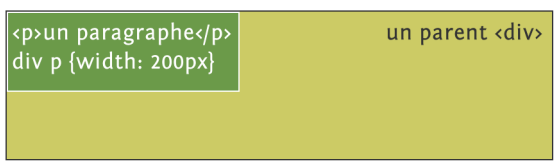
Si une déclaration `text-align: center` appliquée à un bloc suffit à centrer horizontalement le contenu textuel ou les images au sein de celui-ci, l'alignement des éléments de type bloc est généralement plus problématique. Pourtant, l'une des particularités du modèle de boîte, et plus précisément de ses marges externes, permet de centrer simplement les blocs.

Prenons dans cet exercice l'exemple d'un paragraphe que nous souhaiterions centrer horizontalement au sein d'un élément `<div>`.

Puisqu'un paragraphe est de type bloc par nature, il occupe forcément toute la largeur de son parent. Il est par conséquent inutile de tenter de le centrer sans lui attribuer de largeur. Notre première mission consiste donc à intervenir sur cette composante de boîte. Choisissons une largeur de 200 pixels (figure 4-7) :

Figure 4-7

Paragraphe de 200 pixels prêt à être centré



```
div p {
  width: 200px;
}
```

L'alignement au centre du parent est définie à l'aide de marges externes latérales automatiques, c'est-à-dire en appliquant la valeur **`auto`** :

```
div p {
  width: 200px;
  margin-right: auto;
  margin-left: auto;
}
```

À ce stade, le paragraphe large de 200 pixels est centré horizontalement au sein de son conteneur. C'est aussi simple que cela et cette méthode est reconnue depuis Internet Explorer 6.

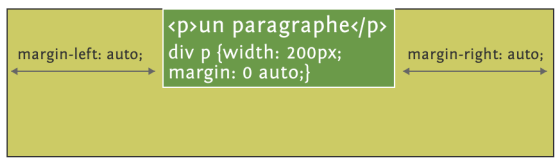
Nous pouvons optimiser le code en employant la syntaxe raccourcie de la propriété `margin`, par exemple :

```
div p {
  width: 200px;
  margin: 0 auto;
}
```

Dans cet extrait, nous affectons les marges externes ainsi : `0` en haut et en bas, `auto` à droite et à gauche (figure 4-8).

Figure 4-8

Des marges automatiques et c'est centré !



Notez pour finir que les marges automatiques appliquées en haut et en bas n'ont aucun effet sur l'alignement *vertical* d'un élément. Le centrage vertical est une mission plus délicate, mais fort heureusement, Internet Explorer 8 a rattrapé son retard dans ce domaine et reconnaît désormais les techniques CSS 2.1 consacrées : `inline-block` et `table-cell` que je détaillerai au sein du chapitre 5.

Fusion de marges

La fusion de marges est une particularité du modèle de boîte qui concerne les marges externes verticales (`margin-top` et `margin-bottom`) des éléments de type bloc positionnés dans le flux. Lorsque deux ou plusieurs éléments sont adjacents, qu'ils soient frères ou imbriqués (parent-enfant), leurs marges verticales se combinent pour n'en former qu'une seule : la plus grande des deux.

Le concept n'étant pas simple à imaginer, voyons immédiatement un exemple.

Partie HTML

```
<p>La marge inférieure de ce paragraphe fusionne...</p>
<p>... avec la marge supérieure de celui-ci</p>
```

Partie CSS

```
p {
  background-color: green;
  margin-bottom: 20px;
}
p + p {
  margin-top: 30px; /* une marge haute sur le second paragraphe */
}
```

Dans notre exemple, nous définissons une marge basse de 20 pixels au premier paragraphe et une marge haute de 30 pixels au second. Nous nous attendons par conséquent à un écart de 50 pixels entre ces deux éléments. Or il n'en est rien : en vertu de la fusion de marges, seule la marge la plus élevée, c'est-à-dire 30 pixels, sera retenue (figure 4-9).

Figure 4-9

Fusion de marges entre frères

Plus déroutant encore, ce mécanisme intervient également dans le cas de boîtes imbriquées, c'est-à-dire entre un parent et son premier ou son dernier enfant, par exemple un paragraphe `<p>` au sein d'un `<div>`.

Le principe demeure identique, à l'exception notable que les marges concernées sont :

- la marge supérieure du parent et la marge supérieure du premier enfant ;
- la marge inférieure du parent et la marge inférieure du dernier enfant.

Parmi ces combinaisons, la marge externe la plus grande est retenue et s'applique *au-dessus (ou en dessous) du parent.*

Observez l'exemple qui suit.

Partie HTML

```
<div>
  <p>la marge haute va se répercuter sur le parent !</p>
</div>
```

Partie CSS

```
div {
  background-color: orange;
```

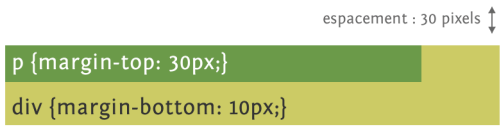
```
}
p {
  background-color: green;
  margin-top: 30px;
}
```

Cette situation illustre un comportement courant, dû à la fusion de marges entre un parent et son enfant : la marge supérieure de `<div>` (non renseignée donc nulle) fusionne avec la marge supérieure de `<p>` (30 pixels). Au final, la marge externe supérieure du bloc `<div>` prend pour valeur… 30 px et tout le bloc est décalé vers le bas (figure 4-10).

Vous seriez alors sans doute tentés d'appliquer une règle `div {margin-top: 0}`, mais celle-ci n'aura aucun effet tant que la valeur sera inférieure à celle du paragraphe.

Figure 4-10

Fusion de marges entre un parent et son enfant



Le mécanisme de fusion de marges a été instauré pour harmoniser le rythme vertical, par exemple pour mieux répartir une suite de paragraphes ou de liens, de façon à ce que l'espace entre chaque élément demeure identique. Dans certains cas, il est souhaitable de s'affranchir de ce phénomène, qui peut perturber l'affichage d'un site web.

Dans les différents cas suivants, la fusion des marges ne s'applique pas entre un parent et son enfant :

- Lorsqu'une bordure est appliquée sur le parent. Une simple bordure haute ou basse suffit et il est possible de rendre cette bordure invisible avec une déclaration de type `border-top: 1px solid transparent`, qui est reconnue depuis IE7. Attention, toutefois, à ne pas oublier que la largeur de la bordure compte dans le calcul des dimensions de la boîte.
- Lorsqu'une marge interne (`padding`) haute ou basse est appliquée au parent. Là aussi, un simple `padding-top: 1px` fonctionne, mais doit être pris en compte dans la hauteur du parent. Cette astuce fonctionne sur tous les navigateurs.
- Lorsqu'un contenu orphelin (non balisé) est introduit avant le premier enfant : n'importe quel texte brut, ou caractère, empêche alors la fusion de marges.
- Lorsque la déclaration `overflow: hidden` ou `overflow: auto` est appliquée sur le parent. Cette solution est intéressante, mais a d'autres conséquences (empêche le dépassement des flottants, peut cacher les contenus qui dépassent ou faire apparaître des barres de défilement). À utiliser avec prudence.
- Lorsque le parent est positionné hors du flux (position absolue, fixée ou flottement). Il s'agit là aussi d'une manœuvre séduisante, mais qui a des répercussions sur l'ensemble du design de votre page web.

BONNE PRATIQUE **Tenez compte des marges par défaut !**

Vous le savez, les éléments de type bloc (quasi tous, sauf `<div>` et les nouveaux éléments HTML 5) possèdent des marges internes ou externes par défaut, même si aucun style CSS ne leur est appliqué. Sachez que la fusion de marges opère également sur ces marges par défaut. Cela explique les cas où des titres (`<h1>`, `<h2>`), du fait de leur marge haute par défaut, « poussent » vers le bas l'ensemble de leur bloc parent !

Rendu par défaut et flux courant

Le rendu des éléments

Tous les éléments d'une page web sont définis par la norme HTML en termes d'imbrications tolérées, mais il le sont également dans leurs spécifications CSS au travers de la propriété intrinsèque `display`, qui leur confère un certain nombre de caractéristiques :

- un rendu général sous forme de « boîte » ;
- une disposition par défaut (les uns sous les autres, à côté, en haut à gauche du parent ou sous le frère précédent) ;
- d'éventuelles marges internes ou externes ;
- la possibilité d'être dimensionnés, etc.

La propriété `display` accepte 17 valeurs depuis CSS 2.1 et quelques valeurs supplémentaires en CSS 3. Les plus couramment employées sont `block`, `inline` et `none`, mais le tableau ci-après vous dévoile quelques autres valeurs intéressantes (figure 4-11).

Tableau 4-1 Le rendu par défaut des éléments

Display (défaut)	Exemples	Spécificités
`block`	`<div>`, `<p>`, `<ul>`, `<h1>`	Les éléments de rendu CSS `display: block` se placent toujours l'un en dessous de l'autre par défaut (comme un retour chariot). Par exemple : une suite de paragraphes (`<p>`). Par ailleurs, un élément bloc occupe automatiquement, par défaut, toute la largeur disponible dans son conteneur et peut être dimensionné à l'aide des propriétés telles que `width`, `height`, `min-width`, ou `min-height`…
`inline`	`<a>`, `<em>`, `<span>`, `<strong>`	Les éléments de rendu `display: inline` se placent toujours l'un à côté de l'autre afin de rester dans le texte. Par défaut, il n'est pas prévu qu'ils puissent se positionner sur la page (même si cela est possible), ni de leur donner des dimensions (hauteur, largeur, profondeur) : leur taille va être déterminée par le texte ou les éléments qu'ils contiennent. Certaines propriétés de marges peuvent être appliquées aux éléments, comme les marges latérales (`margin-left` et `margin-right`).
`none`	`<head>`	L'élément est complètement retiré du flux, il n'est pas restitué par les agents utilisateurs.

Display (défaut)	Exemples	Spécificités
`list-item`	`<li>`	Les éléments de rendu `display: list-item` ont un rendu de type `block` mais peuvent bénéficier des propriétés CSS liées aux puces (`list-style`…), par exemple `<li>`.
`inline-block`	`<input>`, `<select>`	Les éléments de rendu `display: inline-block` conservent les mêmes caractéristiques que les `inline`, mais peuvent être dimensionnés, par exemple l'élément `<input>`.
`table`	`<table>`	Les éléments de rendu `display: table` sont similaires aux éléments de type `block` mais n'occupent par défaut que la largeur de leur contenu, par exemple `<table>`.
`table-cell`	`<td>`, `<th>`	Les éléments de rendu `display: table-cell` ont un rendu similaire aux éléments `inline-block` dans la mesure où ils s'affichent les uns à côté des autres et peuvent être dimensionnés. Leurs particularités supplémentaires est d'accepter la propriété `vertical-align` pour aligner leur contenu verticalement, d'avoir des hauteurs identiques entre éléments frères et de répartir automatiquement leur largeur au sein de leur parent.
`table-row`	`<tr>`	Les éléments de rendu `display: table-row` ont un rendu similaire aux éléments `block` dans la mesure où ils s'affichent les uns sous les autres. Leur particularité supplémentaire est de répartir automatiquement leur hauteur au sein de leur parent.

COMPATIBILITÉ Valeurs de display

Attention : les valeurs « exotiques » de la propriété `display` ne sont pour la plupart reconnues qu'à partir d'Internet Explorer 8. Nous reviendrons en détail sur cette propriété au sein du chapitre 5 concernant le positionnement avancé.

La spécification CSS propose une feuille de styles par défaut qui indique quel devrait être le rendu par défaut de chacun des éléments HTML (*Appendix D* de CSS 2.1), mais cette feuille n'est pas normative et chaque navigateur est libre de décider du rendu des éléments et de la valeur de leur propriété `display`.

Feuille de styles par défaut fournie par la spécification CSS 2.1

```
html, address,
blockquote,
body, dd, div,
dl, dt, fieldset, form,
frame, frameset,
h1, h2, h3, h4,
h5, h6, noframes,
ol, p, ul, center,
dir, hr, menu, pre   { display: block }
li                   { display: list-item }
head                 { display: none }
table                { display: table }
```

```
tr              { display: table-row }
thead           { display: table-header-group }
tbody           { display: table-row-group }
tfoot           { display: table-footer-group }
col             { display: table-column }
colgroup        { display: table-column-group }
td, th          { display: table-cell }
caption         { display: table-caption }
input, select   { display: inline-block }
```

On remarquera que cette feuille de styles considère que le rendu par défaut est `inline` (lorsque non précisé, ou lorsque l'élément n'est pas reconnu, tel que HTML 5 pour les anciens navigateurs).

Le flux

L'ordre dans lequel apparaissent les balises dans le code HTML est celui dans lequel les boîtes sont affichées et s'empilent dans le document. Ce schéma de positionnement par défaut se nomme le *flux courant*. La mise en place des différents éléments de la page s'organise par défaut selon le flux courant.

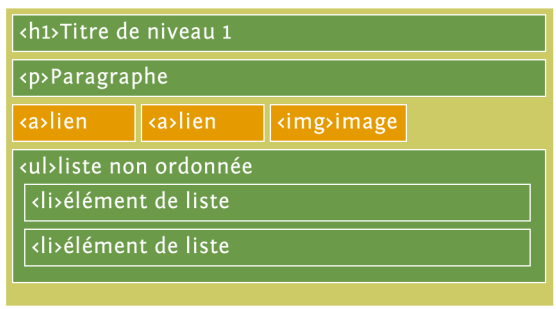
Les règles de positionnement dans le flux courant sont relativement simples et intuitives. Chaque élément :

- est situé sur le même plan que les autres éléments dans le flux ;
- se place le plus haut possible et le plus à gauche possible au sein de son parent ;
- est dépendant de l'élément frère précédent : deux éléments de type `block` s'empilent verticalement l'un sous l'autre, deux éléments de type `inline` se suivent sur la même ligne s'ils en ont la place.

Par défaut, chaque bloc est donc dépendant de ses frères immédiats (figure 4-11).

Figure 4-11

Organisation des éléments dans le flux



Contrairement à ce que l'on peut penser, un système aussi rudimentaire que le positionnement en flux permet d'aller très loin dans la mise en page d'un document (figure 4-12).

- Il offre une souplesse et une fluidité inégalées : puisque chaque élément est directement dépendant de ses voisins, l'ensemble de la page se réorganise automatiquement lorsqu'un

nouvel élément s'ajoute, lorsqu'un élément est retiré, ou lorsque la longueur (ou la taille) des contenus des blocs varie.

- Grâce à une bonne connaissance des composantes du modèle de boîte, on peut faire interagir chaque élément avec son entourage ou son contenu en jouant sur ses valeurs de marges internes, de marges externes, de largeur de contenu, ou de taille de bordures. On peut ainsi dimensionner un bloc et le « positionner » à 100 px en haut et à 300 px à gauche de son parent, simplement en lui affectant des marges externes (ou des marges internes au parent).
- En modifiant la valeur de la propriété `display` d'un élément (voir partie suivante), on lui permet de s'afficher à côté de son voisin ou en-dessous, sans perturber l'ordre naturel du flux.

Figure 4-12

Et voici un bloc positionné en flux grâce à ses composantes de boîte

Tout au long de cet ouvrage, nous allons nous intéresser à de nombreux schémas de positionnement. Certains s'écarteront volontairement du modèle de flux courant, il sont appelés « positionnement hors flux ». Retenez que dans la plupart des cas, il demeurera judicieux de rester dans le flux pour éviter tout écueil.

Positionnement absolu

Souvent employé pour l'agencement des blocs, mais rarement à bon escient ou de façon maîtrisée, le positionnement absolu représente à la fois un extraordinaire moyen d'étendre les limites imposées par le flux courant et une hantise constante du concepteur web ne sachant pas toujours où vont se placer les éléments. Et pourtant, le mécanisme décrivant ce schéma de positionnement est loin d'être une loterie aléatoire ; il est au contraire parfaitement défini et, comble de chance, entièrement compatible avec l'ensemble des navigateurs toutes générations confondues.

Un élément est positionné en absolu, vous l'auriez sans doute deviné, lorsqu'il est affublé de la déclaration `position: absolute`.

```
#info {
  position: absolute;
}
```

Sortir du flux

Cette évidence étant énoncée, ajoutons aux particularités de ce positionnement qu'il retire l'élément concerné du flux selon ce schéma.

- L'élément se retrouve sur un autre plan, placé au-dessus du niveau du flux, tel un calque du célèbre logiciel de dessin Photoshop.

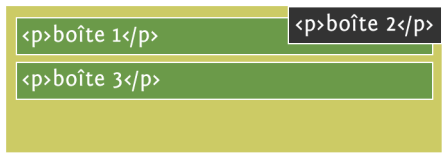
- Les éléments restants se réorganisent entre eux, dans le flux, sans tenir compte de l'élément positionné en absolu hors de leur plan d'affichage (figures 4-13 et 4-14).

Figure 4-13

Deux éléments en flux

Figure 4-14

L'élément B est retiré du flux (position absolue).



On dit des éléments héritant de la propriété `position` qu'ils sont « positionnés ». Aux éléments positionnés peuvent s'appliquer les quatre propriétés `top`, `right`, `bottom` et `left`. Il est inutile d'essayer de les affecter à un élément en flux, elles n'auront aucune conséquence.

En revanche, elles sont idéales pour placer l'élément (ou plus précisément son coin supérieur gauche) par rapport aux bords de son bloc référent. Voici quelques exemples :

```
#info {
  position: absolute;
  top: 0; left: 0; /* le coin supérieur gauche du bloc #info se place dans le coin
  ➥ supérieur gauche de son référent */
  top: 0; right: 0; /* le coin supérieur droit du bloc #info se place dans le coin
  ➥ supérieur droit du référent */
  left: 0; bottom: 0; /* dans le coin inférieur gauche du référent */
  top: 20px; left: 20px; /* à 20 pixels du coin supérieur gauche du référent */
  top: 10%; right: 10%; /* à 20 % du coin supérieur droit du référent */
  top: 50%; left: 50%; /* le coin supérieur gauche du bloc #info se place au centre
  ➥ de son référent */
}
```

À quel saint se vouer ?

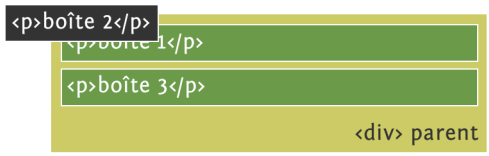
Nous avons à présent bien avancé sur le concept du positionnement absolu, mais il nous manque une donnée essentielle : qui est donc ce fameux « référent » mentionné partout ? Plus prosaïquement, par rapport à quoi se positionne un élément en absolu ?

C'est là que beaucoup de concepteurs web se fourvoient : souvent, la réponse qui m'est apportée est « par rapport aux coins de l'écran » ou « selon le document web ». Même si ce cas de figure peut survenir, il ne s'agit pourtant que d'une exception dans la règle.

L'étiquette « élément positionné » prend toute sa signification lorsqu'on énonce la formule magique décrivant ce mécanisme de positionnement : *un élément positionné en absolu se place par rapport à son premier ancêtre* ***positionné*** (figure 4-15).

Figure 4-15

Positionnement absolu sans référence

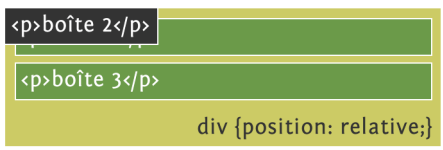


Le principe est le suivant : le bloc positionné en absolu remonte toute sa branche au sein de la hiérarchie dans le code HTML. Il vérifie si son parent est « positionné », c'est-à-dire s'il est muni de la propriété `position` avec une valeur autre que `static`. Si tel n'est pas le cas, il remonte d'une génération et ainsi de suite jusqu'à trouver un ancêtre positionné. En dernier recours, si l'élément positionné absolument ne rencontre aucun ancêtre positionné, son référent est l'élément racine `<html>`.

Nous comprenons, grâce à cette règle, qu'il devient aisé d'indiquer une référence à un bloc absolu : il suffit d'appliquer une déclaration `position: absolute`, `position: fixed` ou `position: relative` à cet élément de référence (figure 4-16).

Figure 4-16

Positionnement absolu avec une référence



Un mode de rendu particulier

Le positionnement absolu ne répond pas seulement à un mode de placement caractéristique, mais se double également de certaines spécificités de rendu :

- Sa boîte devient dimensionnable quel que soit son type de rendu initial. Cela signifie qu'un élément HTML de rendu `inline` par défaut peut bénéficier des propriétés `width`, `height`, `min-width`, `max-width`, etc. qui auraient été inopérantes dans le flux.
- L'élément occupe par défaut exactement la largeur de son contenu, c'est-à-dire qu'un bloc qui ne contient qu'un seul mot n'occupera que la surface de ce mot, tandis qu'un élément contenant de longs paragraphes s'étendra sur toute la largeur du référent.
- Les marges externes des éléments positionnés en absolu ou de leurs enfants ne sont pas sujettes au phénomène de fusion de marges évoqué précédemment.

La profondeur : z-index

Tous les éléments de la page se placent selon une composante horizontale (axe *x*), verticale (axe *y*) et de profondeur (axe *z*).

La propriété `z-index` a pour valeur un nombre entier qui représente l'ordre d'empilement de l'élément. Plus cet entier est grand, plus la couche sera haute dans la pile (figure 4-17). La valeur 0 de `z-index` représente la couche de base des éléments dans le flux. Une valeur négative est permise afin de passer un élément derrière tous les plans visibles, mais est encore relativement mal interprétée par les navigateurs.

> **IMPORTANT Éléments concernés par z-index**
> Notez que seuls les éléments « positionnés » peuvent être affectés par cette propriété d'empilement `z-index`.

Figure 4-17

z-index appliqué à des blocs positionnés

Dans la plupart des cas de figure classiques, la compréhension de l'empilement est intuitive, mais elle l'est moins dès que le contexte est plus complexe. Voici les règles à retenir :

- Sans `z-index`, les éléments s'empilent suivant leur ordre d'apparition dans le code HTML. Le premier se place sous les suivants.
- Lorsqu'une propriété `z-index` est appliquée à plusieurs éléments positionnés au sein d'un même référent, celui bénéficiant de la valeur la plus forte apparaît par dessus les autres.
- Lorsque les éléments positionnés dépendent de plusieurs référents qui eux-mêmes disposent d'une valeur de `z-index`, c'est le poids du `z-index` référent qui devient prioritaire.

Étirer un élément

L'une des possibilités originales offertes par les propriétés `top`, `right`, `bottom` et `left` est de pouvoir cumuler les paires contraires (`left` et `right` ou `top` et `bottom`) pour littéralement étirer l'élément positionné en absolu au sein de son référent.

Ainsi, un paragraphe positionné doté de la double déclaration `left: 0` et `right: 0` va s'étendre sur toute la largeur de son référent au lieu de n'occuper que la largeur de son contenu.

L'intérêt peut ne pas sembler évident au premier abord, puisqu'une largeur de type `width: 100%` suffit à déployer le bloc sur toute la surface libre. Cependant, rappelez-vous que la propriété `width` ne représente que la composante de contenu d'une boîte et que les autres unités (bordures, marges internes et externes) sont parfois nécessaires et vont s'ajouter à cette valeur de 100 %,

provoquant un conflit : le bloc peut dépasser de son référent, ou pire, interagir avec d'autres éléments avoisinants.

Dans le cas où l'élément absolu est « dimensionné » à l'aide de ses paires contraires, il devient possible d'occuper 100 % de la largeur tout en bénéficiant de `padding` et de `border` :

```
#header {
  position: absolute;
  left: 0; right: 0; /* s'étend sur toute la largeur  */
  padding: 20px; /* le padding est inclus à l'intérieur  */
  background: #abc;
}
```

Cette astuce fonctionne également pour la paire `top` et `bottom` dans le cas d'une expansion verticale et elle permet certaines fantaisies, telles que centrer un élément dans la page sans lui imposer de dimensions (figure 4-18) :

```
#popup {
  position: absolute;
  top: 25%; bottom: 25%; /* top et bottom  */
  left: 25%; right: 25%; /* left et right  */
  padding: 10px;
  background: #abc;
}
```

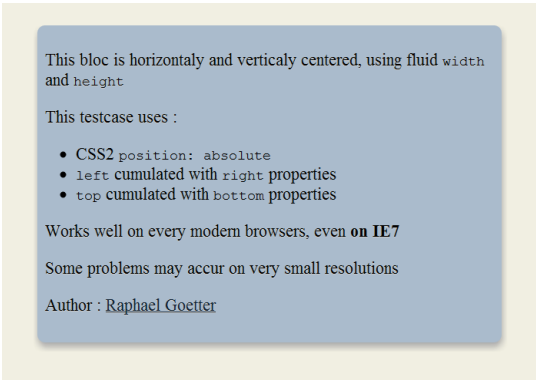
Visualiser le résultat en ligne

- http://www.ie7nomore.com/fun/stretching/

Un dernier détail a son importance : cette technique fonctionnelle sur tous les navigateurs récents n'est comprise qu'à partir d'Internet Explorer 7.

Figure 4-18

Centrer en absolu



Positionnement fixé

Autant le positionnement absolu est célèbre, autant il est bien plus rare d'entendre parler du positionnement fixé (figure 4-19). Pourtant les similitudes entre les deux schémas sont nombreuses :

- Un élément est dit *fixé* lorsqu'il bénéficie de la déclaration `position: fixed`.
- Puisqu'on lui applique une propriété `position`, l'élément est dit « positionné » (il sert donc de référent aux éléments positionnés en absolu).
- Un bloc fixé sort du flux et se place dans un autre plan, laissant les éléments en flux se réorganiser entre eux.
- La boîte d'un élément fixé devient dimensionnable, à l'instar des éléments absolus.

Le positionnement fixé se distingue principalement de son cousin absolu de par sa particularité de rendu : *un élément fixé demeure ancré à l'écran même lorsque l'utilisateur fait défiler le contenu à l'aide des ascenseurs* (*scrollbars*).

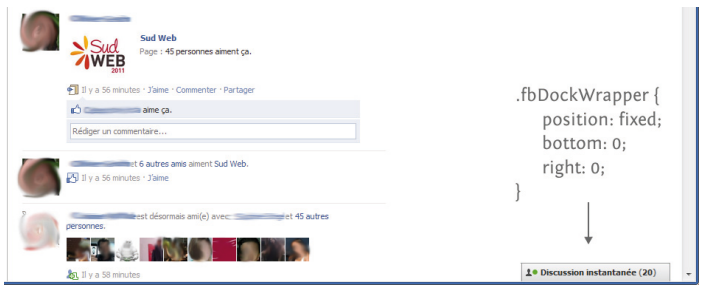
En termes de localisation, et lorsque les propriétés `top`, `right`, `bottom` et `left` sont précisées, *l'élément fixé est toujours positionné par rapport à la partie visible de la fenêtre du navigateur*, quel que soit son ancêtre, fût-il positionné.

Ce type de positionnement se révèle particulièrement séduisant dans la conception d'un menu de navigation qui demeurera figé à l'écran même si le contenu de la page défile :

```
#nav {
  position: fixed;
  top: 0;
  right: 0;
  width: 200px;
  background: #abc;
}
```

Figure 4-19

Principe du positionnement fixé : le tchat de Facebook



EXEMPLE Positionnement fixé dans Facebook

Le célèbre réseau social en ligne affiche une barre d'information (les contacts actuellement en ligne) constamment en bas de page à droite de la fenêtre (figure 4-19). Ce comportement est tout simplement fondé sur le mécanisme de positionnement fixé.

Quelques réserves sont à prendre en compte avant d'employer ce schéma de positionnement : tout d'abord, il n'est reconnu qu'à partir d'Internet Explorer 7. Par ailleurs, il peut dans certains cas poser de lourds problèmes d'ergonomie : pensez qu'un élément positionné à 800 pixels du haut de l'écran devient totalement invisible et inaccessible sur de petits écrans de netbooks par exemple, dont la résolution est de 600 pixels de hauteur. Ce type de positionnement est d'ailleurs inactivé sur certains smartphones, notamment ceux sous environnement Safari Mobile (iPhone), Opera Mini ou Android.

Positionnement relatif

Troisième et petit frère de la famille, le positionnement relatif porte à mon sens très mal son nom, tout simplement parce qu'il ne représente en aucune manière un positionnement tel qu'il a été défini jusque là, mais un simple décalage.

Un élément positionné relativement se place par rapport à sa position classique dans le flux, puis est éventuellement décalé si au moins une des propriétés `top`, `right`, `bottom` *ou* `left` *est renseignée*. La notion de « relatif » s'applique par conséquent à son emplacement initial dans le flux (figure 4-20).

Contrairement aux schémas absolus et fixés, les propriétés `top`, `right`, `bottom` et `left` ne servent plus à indiquer un emplacement, mais un *décalage* par rapport à la position initiale.

```
strong {
  position: relative;
  top: 10px;
  left: 10px;
  background: #abc;
}
```

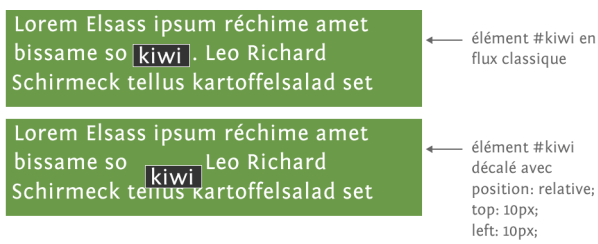


Figure 4-20

Principe du positionnement relatif

Un élément relatif demeure dans le flux et continue à interagir avec les autres éléments voisins. D'ailleurs, l'espace laissé par un élément décalé en relatif ne peut pas être occupé par d'autres éléments, car il est toujours censé l'occuper (figure 4-21).

```
strong {
  position: relative;
  top: 100px;
  left: 100px;
```

```
    background: #abc;
}
```

Figure 4-21

Un espace laissé libre

Un élément positionné relativement conserve une particularité due à sa propriété CSS : il est dit « positionné ». Le seul fait d'être positionné en relatif, sans adjonction de valeurs pour `top`, `right`, `bottom` et `left` en fait un référent dans le flux, parfait pour les contenus positionnés en absolu. C'est d'ailleurs son usage principal.

Positionnement flottant

La propriété `float` apparaît dès CSS 1, classée dans les « modèles de mise en forme », tels que `inline` et `block` (un peu comme si l'idée avait été de créer une règle de type `display: float`).

Un usage détourné de son objectif initial

Les spécifications initiales ne semblent pas avoir prévu l'usage de cette propriété pour positionner les éléments, comme que nous le faisons actuellement (même si rien n'indique que cela soit interdit non plus) : tous les exemples illustrant le flottement désignent des images ou des portions de texte à pousser à droite ou à gauche dans un élément.

Employant `float` comme un schéma de positionnement hors flux hybride, les concepteurs web détournent finalement cette propriété de son usage premier, en vue de disposer des éléments les uns par rapport aux autres, voire de les imbriquer.

Cet usage étendu de `float` a rendu son implémentation complexe dans les navigateurs : c'est le type de positionnement qui rencontre le plus d'erreurs d'affichage connues et recensées. Il nécessite de jouer avec des contextes de formatage obscurs pour des débutants. Le modèle global de flottement est interprété différemment selon chaque navigateur, c'est le moins qu'on puisse dire. Bref, il en résulte un joli mélange indigeste pour les concepteurs web néophytes à qui l'on a rabâché que la mise en page via tableaux était à proscrire.

En 2011, à l'heure où CSS 3 gagne du terrain, force est de constater que le mode de positionnement le plus employé actuellement date de CSS 1 (1996). Pire, il n'a pas été conçu pour cela au départ.

Un positionnement à part

Contrairement à la croyance générale véhiculée par les concepteurs novices qui l'emploient à tour de bras, le flottement est le positionnement le plus biscornu de tous et comptant le plus de

cas particuliers. Il détient le record inégalé de claviers jetés et de cheveux arrachés au sein de notre communauté.

Le principe de base est pourtant élémentaire : *un élément flottant est ôté du flux et placé à l'extrême gauche (`float: left`) ou droite (`float: right`) de son conteneur, tout en demeurant sur sa hauteur de ligne initiale dans le flux.*

Cela se corse légèrement par la suite, car si l'élément est extrait du flux courant, il ne l'est que *partiellement* : les éléments précédant le bloc flottant ne sont pas affectés ; cependant, tous les éléments suivants se réorganisent dans le flux comme dans le cas du positionnement absolu, sauf... leur contenu qui, lui, s'écoule autour du flottant en épousant sa forme.

Il est par conséquent nécessaire d'être très attentif au modèle de boîte des éléments suivant le flottant : seule la composante de contenu de la boîte s'écoule autour de l'élément flottant qui la précède. En revanche, la boîte elle-même se repositionne dans le flux à la suite de la boîte en flux précédente.

Vous avez compris ? Vous avez raison : je crois que quelques illustrations ne sont pas superflues (figures 4-22 et 4-23).

Figure 4-22

Avant flottement (deux blocs en flux)

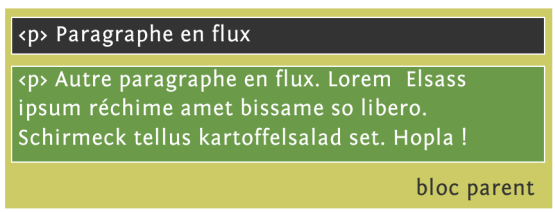
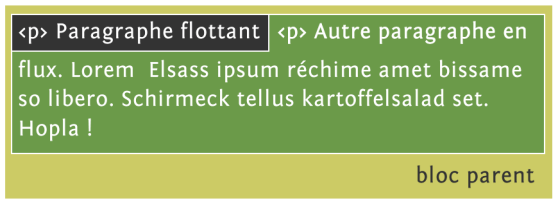


Figure 4-23

Après flottement (premier bloc flottant à gauche)



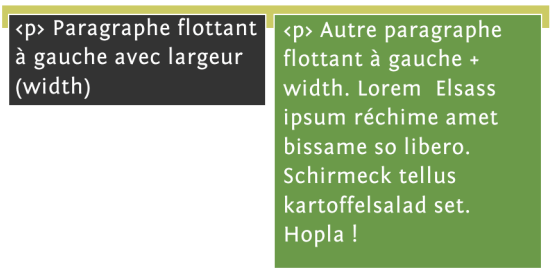
Si ce n'est toujours pas clair pour vous, replongez-vous dans la section traitant du modèle de boîte d'un élément (en début de chapitre), composé d'une zone de contenu, mais aussi de marges internes, de bordures et de marges externes.

Des blocs côte à côte

À l'instar du positionnement absolu, un élément flottant adopte par défaut la largeur qu'occupe son contenu (figure 4-24). Si celui-ci est dense, elle est susceptible d'occuper toute la largeur du parent, c'est pourquoi il est souvent nécessaire de fixer une largeur au flottant via la propriété `width` ou `max-width`.

Figure 4-24

Deux blocs côte à côte (float: left)



Lorsqu'un élément flottant est poussé dans la même direction qu'un autre élément flottant, il demeure sur le même plan et se « cale » à ses côtés. Il s'agit d'une méthode courante de mise en forme de blocs côte à côte tel qu'un menu de navigation et la zone de contenu.

Attention, toutefois, au caractère hâtif de cette disposition adjacente « magique » qui s'écarte de l'application originale de la propriété `float` : gardez à l'esprit que le flottement est un positionnement hors flux et qu'il n'a plus de prise sur les blocs alentours en flux, à l'exception des contenus qui vont « épouser » les flottants.

Cela signifie qu'un parent ne contenant que des flottants n'occupera physiquement aucune surface à l'écran, ou encore que les objets flottants vont « dépasser » de leur conteneur, puisque seuls des contenus en flux sont susceptibles de meubler l'espace dans le plan occupé par le flux courant. Cela est simple à démontrer en appliquant une couleur de fond au conteneur (figures 4-25, 4-26 et 4-27).

Partie HTML

```
<div>
  <p>Lorem Elsass ipsum réchime amet sed bissame so libero</p>
  <p>Hopla kuglopf flammekueche jetz gehts los</p>
</div>
```

Partie CSS

```
div {
  background-color: beige;
}
p {
  float: left;
  width: 150px;
  background-color: black;
}
```

Figure 4-25

Parent vide

Figure 4-26

Ajout de hauteur pour « voir » le parent

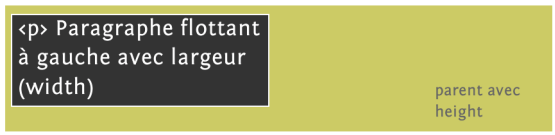
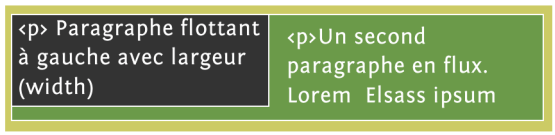


Figure 4-27

Ajout d'un paragraphe non flottant : le parent s'adapte.



La propriété clear

La propriété `clear` est une sorte de passerelle entre deux plans du document : celui du flux courant et celui des flottants (figures 4-28 et 4-29). Elle interdit à un élément de se placer sur la même ligne qu'un bloc flottant et le force par conséquent à se caler directement en dessous de celui-ci. Elle autorise par ailleurs un nettoyage des flottants exclusivement à gauche (`clear: left`), à droite (`clear: right`) ou les deux simultanément (`clear: both`).

Figure 4-28

Deux éléments : l'un flottant à gauche, l'autre en flux

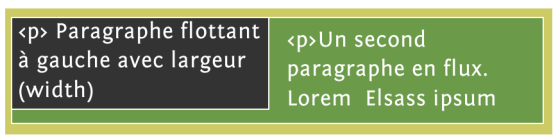


Figure 4-29

Clear left sur le second élément

Cette fonctionnalité offre un certain nombre d'avantages, puisqu'elle autorise un objet en flux à interagir avec les flottants précédents. Cela permet, par exemple, d'empêcher des « dépassements » de flottants de leur parent, ou de positionner un élément toujours en bas du bloc flottant le plus long comme le montre l'exemple suivant (figure 4-30).

Partie HTML

```
<div id="bloc1">Lorem Elsass ipsum</div>
<div id="bloc2">réchime amet sed bissame so libero knackes choucroute...</div>
```

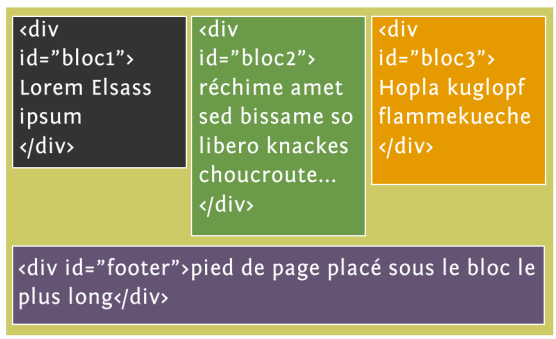
```
<div id="bloc3">Hopla kuglopf flammekueche</div>
<div id="footer">pied de page placé sous le bloc le plus long</div>
```

Partie CSS

```
#bloc1, #bloc2, #bloc3 {
  float: left;
  width: 150px;
  margin: 0 10px 0 0;
  background-color: yellow;
}
#footer {
  clear: both;
  background-color: green;
}
```

Figure 4-30

Une conception en trois colonnes grâce à clear



Quiz sur le positionnement flottant

Questions

Observez bien chacune des figures 4-31 à 4-36 et répondez à la question suivante : « Quelles sont les propriétés qui ont été appliquées aux différents éléments de l'image pour obtenir le rendu final ? »

Figure 4-31

Schéma 1

Figure 4-32

Schéma 2

Figure 4-33

Schéma 3

Figure 4-34

Schéma 4

Figure 4-35

Schéma 5

Figure 4-36

Schéma 6

Réponses

Réponse 1

Aucune propriété n'a été appliquée à aucun des éléments : il s'agit simplement du positionnement par défaut en flux de deux blocs.

Réponse 2

Le premier bloc a bénéficié de la déclaration `float: left`. Il est poussé à gauche sur sa ligne, sort du flux et n'occupe plus que la largeur de son contenu. Automatiquement, le second bloc, dans le flux, remonte et se positionne en haut à gauche de son parent, mais son contenu s'écoule autour du flottant.

Réponse 3

Les deux blocs sont en flottant à gauche. Ils sortent du flux, n'occupent que la largeur de leur contenu et sont poussés à gauche sur leur ligne. Leur parent ne contient plus d'éléments en flux et n'occupe plus de hauteur visible.

Réponse 4

Les deux blocs sont flottants à droite. L'ordre est inversé par rapport à l'apparition dans le code HTML puisque le navigateur traite un élément après l'autre : le bloc 1 est d'abord poussé à droite sur sa ligne; puis c'est au tour du bloc 2, après être remonté dans le flux un court instant.

Réponse 5

Seul le bloc 2 s'est vu attribuer une déclaration `float: right` et est poussé à droite sur sa ligne. Le premier élément demeure tout simplement dans le flux.

Réponse 6

La situation est identique au cas de figure précédent, mais un troisième élément a été ajouté. Il est simplement dans le flux mais bénéficie d'une déclaration `clear: right` ou `clear: both`. Notez qu'un tel rendu aurait été réalisable sans sortir du flux : il suffisait d'appliquer une marge à gauche du deuxième élément.

Exercice pratique : dépassement de flottants

L'exercice que nous allons décrire à présent a pour objectif d'empêcher les dépassements de flottants, en ayant recours à la propriété `clear` que nous allons confier à un élément créé automatiquement en CSS grâce au pseudo-élément `:after`.

Dans certaines conditions particulières (dites « contexte de formatage bloc »), un conteneur est susceptible d'englober ses descendants flottants : plutôt que de déborder, ceux-ci « poussent » le conteneur. Ces conditions sont définies par l'un des comportements suivants :

- un conteneur lui-même flottant ;
- en positionnement absolu ;
- en `display: inline-block ;`
- en `display: table-cell ;`
- ou avec une propriété `overflow` dont la valeur est autre que `visible`.

Toutes ces propriétés ont une incidence sur le rendu de l'élément et ce n'est généralement pas ce que l'on souhaite. C'est pourquoi je vous propose, dans cet exercice pratique, d'employer une astuce qui commence à faire son chemin chez les concepteurs web : créer un élément affublé d'une propriété `clear` afin d'empêcher tout dépassement de flottants.

Prenons pour exemple un bloc `<div>` contenant des éléments flottants qui débordent. Commençons par assigner la classe `.clear` à ce parent.

Partie HTML

```
<div class="clear"> ici des éléments dont des flottants </div>
```

À l'aide du pseudo-élément `:after`, créons un élément dont le contenu se résume à un caractère « espace » (caractère ASCII hexadécimal `\0020`).

Partie CSS

```
.clear:after {
  content: "\0020";
}
```

Pour rendre ce caractère invisible, nous devrons passer par plusieurs subterfuges : jouer sur la propriété `visibility`, mais également annuler sa taille de police et sa hauteur.

```
.clear:after {
  content: "\0020";
  visibility: hidden;
```

```
  height: 0;
  font-size: 0;
}
```

Enfin, la magie opère lorsque l'on applique la propriété `clear` sur cet élément créé, sans avoir oublié de le convertir en bloc, sans quoi l'action demeurera inopérante.

```
.clear:after {
  content: "\0020";
  clear: both;
  display: block;
  visibility: hidden;
  height: 0;
  font-size: 0;
}
```

Cette méthode pratique permet de se soustraire à l'ajout de balises HTML superflues et est prise en charge par tous les navigateurs depuis un bon moment. Seul Internet Explorer risque de nous poser des soucis pour ses versions antérieures à IE8.

Pour Internet Explorer 6 et 7, qui ne reconnaissent pas le pseudo-élément `:after`, attribuer le *Layout* à l'élément suffit à créer un contexte de formatage suffisant (voir le chapitre 6 concernant la résolution des erreurs) :

```
.clear {zoom: 1;}
```

Grâce à ce correctif, cette technique devient universelle.

Cumuler les schémas de positionnement

Les spécifications W3C admettent l'éventualité de cumuler différents schémas de positionnement. Un élément peut ainsi être affublé des déclarations `float: right` et `position: relative` dans le but de le décaler.

Il advient toutefois que certaines règles soient contradictoires : difficile de placer un élément à la fois en `float: left` et en `position: absolute` à la coordonnée `right: 0` !

C'est pour administrer ces cas de figure que CSS 2 a défini des conventions de priorité entre les propriétés `display`, `float` et `position` :

1. Si `display: none` est appliqué, alors l'élément ne crée pas de boîte et les propriétés `position` et `float` ne sont pas employables.
2. Sinon, si une propriété `position` est appliquée et a pour valeur `absolute` ou `fixed`, ce schéma devient prioritaire, la valeur de la propriété `float` est forcée à `none` et l'élément est placé selon les éventuelles valeurs de `top`, `right`, `bottom` et `left`.
3. Sinon, si une propriété `float` est appliquée avec la valeur `left` ou `right`, l'élément devient flottant.
4. Sinon, l'élément est disposé selon son mode de rendu par défaut inhérent à la propriété `display` (`inline`, `block`, `list-item`, etc.).

Quiz de connaissances

Questions

Question 1

Quelle propriété appliquer à un élément pour annuler la fusion de marge avec son frère ?

- `display: block`
- `border`
- `zoom`
- `margin`

Question 2

Quelle est la différence entre `height` et `min-height` en pratique, lorsque le contenu est plus haut que son parent ?

Question 3

Où se place un élément doté des déclarations suivantes ?

```
position: absolute;
right: 0; top: 0;
float: left;
```

Question 4

Comment se place un élément doté des déclarations suivantes ?

```
position: fixed;
bottom: 0; left: 0; right: 0;
```

Question 5

Quelle est la valeur par défaut de la propriété `display` associée aux éléments HTML `<input>` ?

Question 6

Je veux disposer deux blocs dimensionnés côte à côte. Je ne peux employer ni le positionnement absolu ni le flottement. De quelles autres solutions disposé-je en CSS 2 ?

Réponses

Réponse 1

Parmi toutes les propriétés proposées, seule `border` permet d'annuler la fusion de marges.

Réponse 2

Selon l'importance du contenu au sein de la boîte, celui-ci dépasse (`height`) ou pousse l'ensemble de la boîte (`min-height`).

Sur IE6, le comportement est faussé en raison d'une erreur d'interprétation : le contenu va pousser la boîte avec `height`… et la propriété `min-height` n'est pas reconnue.

Réponse 3

Lorsque deux types de positionnement entrent en conflit, certaines priorités opèrent. Dans le cas proposé, le positionnement absolu prend le pas sur le flottement. L'élément se place donc en haut à droite de son ancêtre positionné.

Réponse 4

L'élément se place tout en bas de l'écran et occupe toute la largeur.

Réponse 5

La valeur par défaut de la propriété `display` associée aux éléments HTML `<input>` est `inline-block`.

Réponse 6

Parmi les schémas de positionnement offrant la possibilité de placer deux éléments dimensionnés côte à côte, citons `display: inline-block` et `display: table-cell`. Ces deux schémas sont traités dans le chapitre suivant.

5

Positionnement avancé

Ce chapitre vous dévoile un panel de techniques de positionnement dites « avancées », assez peu connues et développées jusqu'à présent en raison du barrage imposé par le vénérable ancêtre Internet Explorer 6. On y traitera des schémas de rendu `inline` et `block` à la fois, des modèles d'affichages tabulaires et des nouveautés annoncées par CSS 3, dont le très attendu modèle de boîte flexible.

La version 2 des CSS apparaît en 1998. Les solutions apportées par CSS 2, aussi adéquates qu'elles puissent être, émergent dans un monde où d'autres techniques ont déjà été adoptées, maîtrisées et prônées depuis longtemps par les acteurs du marché. Les navigateurs acceptent ces normes au fur et à mesure, et avec précaution, afin de pouvoir conserver tout ce qu'ils ont déjà bâti jusque là et qui fonctionne de façon robuste.

La démocratisation du positionnement flottant, malgré ses contraintes, permet de se reconcentrer sur l'universalité du Web à travers la séparation entre le fond et la forme.

L'intention est louable et commence à porter ses fruits, mais de nombreuses autres ressources plus appropriées existent depuis de longues années en CSS 2 et demeurent inexploitées en raison du mutisme d'un célèbre navigateur jusqu'à maintenant. La sortie d'Internet Explorer 8, reconnaissant enfin toute la spécification CSS 2.1, change la donne de façon radicale et ouvre la voie à des mises en page propres, logiques et robustes… sans avoir nécessairement recours à des artifices.

Combiner block et inline

L'une des situations les plus couramment rencontrées au sein des forums de discussion concerne la mise en page de deux (ou plusieurs) blocs côte à côte tout en étant dimensionnés. Les schémas évoqués précédemment, le positionnement absolu et le flottement, répondent à cette problématique, mais dans un contexte difficile à maîtriser, dû à leur emplacement hors du plan des autres éléments en flux.

Cette double prédisposition, pourtant familière, est par conséquent plus délicate à obtenir qu'on ne le pense si l'on souhaite demeurer dans le flux. En effet, nous avions longtemps tendance à croire que :

- seuls les éléments de type `block` peuvent bénéficier de largeur (`width`) ou de hauteur (`height`) ;
- et seuls les éléments de type `inline` se placent les uns à côté des autres.

Pour cumuler les deux avantages, la coutume exige – comme nous l'avons vu – de recourir aux positionnements hors flux tels que les flottants ou le positionnement absolu.

display: inline-block

La solution à ce problème de disposition existe pourtant. Elle est même là devant nos yeux depuis bien longtemps. Née avec CSS 2.1, l'une des valeurs de la propriété `display` se pose en véritable pierre philosophale depuis plus de dix ans, mais demeure méconnue et sous-exploitée : `inline-block`.

Tout élément doté de la règle `display: inline-block` devient hybride. Il est considéré comme un texte de contenu et hérite de toutes les caractéristiques d'un élément `inline` (se place à côté de l'élément précédent et se positionne verticalement sur la ligne de texte), tout en bénéficiant d'une particularité inhérente aux éléments `block` : celle de pouvoir être dimensionné. En bref, il se comporte à la fois en `block` et `inline`, en adoptant des avantages de chacun.

Voilà donc une technique ancienne aux vertus quasi parfaites. La suite du chapitre démontre malheureusement que la perfection n'existe pas en CSS… ou plutôt dans son interprétation par les navigateurs.

Pour illustrer cette notion, prenons l'exemple d'un lien hypertexte classique (`<a>`), dont le rendu par défaut est `inline`. En vertu de son appartenance au groupe HTML des éléments en ligne, un lien ne peut pas bénéficier des propriétés de dimension telles que `width` (essayez, vous verrez bien). C'est là que la valeur `inline-block` prend tout son intérêt : modifiez simplement la valeur de rendu par défaut du lien et vous serez en mesure de lui appliquer toutes les propriétés normalement réservées aux blocs (figure 5-1).

Partie HTML

```
<a href="#">un lien de 500px de large ?</a>
```

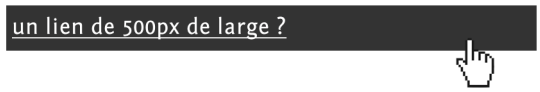
Partie CSS :

```
a {
  display: inline-block;
  width: 500px;
```

```
  background-color: #abc;
}
```

Figure 5-1

Un lien dimensionné grâce à display: inline-block



Nul besoin d'astuces supplémentaires, cet exemple est parfaitement fonctionnel sur l'ensemble des navigateurs actuels (depuis Firefox 3) et même depuis Internet Explorer 6.

Notre enthousiasme envers ce dernier navigateur va cependant être de courte durée, car je dois vous apprendre que son traitement de la valeur `inline-block` est très partiel : elle n'est prise en compte que si l'élément affecté est originellement de type `inline` et, appliquée à un élément `block`, elle n'a aucun effet avec IE6 et IE7. En fait, cette méthode n'est véritablement reconnue qu'à partir d'Internet Explorer 8.

Particularités pour IE6 et IE7

S'il faut attendre IE8 pour bénéficier de la prise en charge complète de `display: inline-block`, vous serez étonnés mais heureux d'apprendre que cela ne va toutefois pas vous empêcher de l'employer dès aujourd'hui, sans crainte et à tour de bras, pour deux raisons :

- Cette valeur de `display` est parfaitement reconnue lorsqu'elle est appliquée sur un élément HTML de type `inline` par défaut, à savoir `<a>`, `<span>`, `<em>`, `<strong>`… et ce, dès IE6 !
- Il est possible d'émuler le comportement `inline-block` sur IE6 et IE7 à l'aide d'un subterfuge peu incommodant : transformer l'élément en `display: inline` tout en lui affectant le *Layout* (voir le chapitre 6 sur la résolution de bogues), par exemple à l'aide de la déclaration `zoom: 1`. Il pourra alors être dimensionné.

De ces deux particularités découlent les méthodes suivantes :

- Si l'élément est de type HTML `inline` au départ, alors la règle `display: inline-block` sera tout à fait opérationnelle sur tous les navigateurs dont Internet Explorer.
- Si l'élément est naturellement de type `block` (`<div>`, `<p>`, `<h1>`, `<ul>`, `<form>`…), alors il sera nécessaire d'avoir recours à une astuce telle que celle décrite ci-après.

Dans la feuille de styles CSS classique

```
element {
  display: inline-block;
}
```

Dans une feuille de styles CSS conditionnelle pour les versions d'IE antérieures à IE8

```
element {  /* @bugfix inline-block sur IE6/IE7 */
  display: inline;
  zoom: 1; /* donner le layout */
}
```

Une autre possibilité est de ne conserver qu'une seule feuille de styles via l'emploi de *hacks* spécifiques à IE6 et IE7 (ici le caractère étoile `*` précédant immédiatement le nom de la propriété). Faites attention, toutefois, puisque cette technique invalidera votre fichier CSS.

Dans la feuille de styles CSS classique

```
element {
  display: inline-block;
  *display: inline;  /* @bugfix hack pris en compte uniquement par IE6/IE7 */
  zoom: 1; /* donner le layout */
}
```

Grâce à ce détournement sans gros impact sur votre productivité, n'importe quel élément HTML peut bénéficier d'un rendu de type `inline-block` sur tous les navigateurs actuellement en fonction. L'un des bénéfices apportés par cette valeur de `display` est d'autoriser la disposition mitoyenne de blocs dimensionnés tout en demeurant dans le flux, profitant de sa fluidité. C'est un type de positionnement que j'affectionne tout particulièrement et que j'applique aussi souvent que possible, lorsque le cas se présente.

Alignement vertical

À l'instar des contenus de type textuel, les éléments déclarés en `display: inline-block` se positionnent par défaut sur la ligne de texte (*baseline*), comme le montre l'exemple qui suit (figure 5-2).

Partie HTML

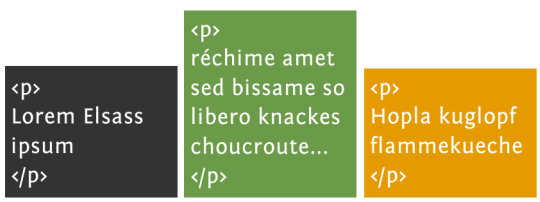
```
<p>Lorem Elsass ipsum</p>
<p>réchime amet sed bissame so libero knackes choucroute...</p>
<p>Hopla kuglopf flammekueche</p>
```

Partie CSS

```
p {
  display: inline-block;
  width: 150px;
  margin: 0 10px 0 0;
  background-color: #abc;
}
```

Figure 5-2

Alignement vertical par défaut : baseline



La propriété `vertical-align`, réservée aux éléments de contenu ou aux cellules de tableaux, modifie ce comportement intrinsèque. Appliquée à un élément en `display: inline-block`, elle va aligner cet élément au sein de son parent, comme s'il s'agissait d'un texte, via un large assortiment de possibilités :

Tableau 5-1 Valeurs possible pour la propriété vertical-align

Valeur	Description
`baseline`	Aligne la base de l'élément avec celle de son parent (valeur par défaut).
`bottom`	Aligne le bas de l'élément avec le bas du parent.
`text-bottom`	Aligne le bas de l'élément avec le bas de la police de l'élément parent.
`top`	Aligne le haut de l'élément avec le haut du parent.
`text-top`	Aligne le haut de l'élément avec le haut de la police de l'élément parent.
`sub`	Aligne la base de l'élément avec la ligne de base indice de son parent, tel l'élément HTML `<sub>`.
`super`	Aligne la base de l'élément avec la ligne de base exposant de son parent, tel l'élément HTML `<sup>`.
`middle`	Aligne le milieu de l'élément avec le milieu de son parent.
`<longueur>`	Aligne la base de l'élément à la longueur donnée au-dessus de la ligne de base de son parent.
`<pourcentage>`	Idem à la valeur `<longueur>`, où le pourcentage est calculé par rapport à la propriété `line-height`.

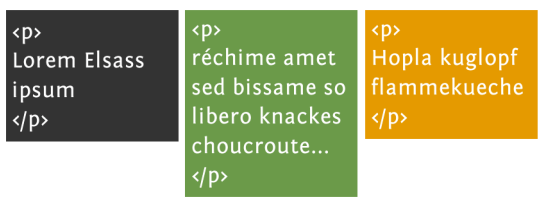
En appliquant la valeur `top` à la propriété `vertical-align`, nous obtenons un rendu plus familier (figure 5-3).

Partie CSS

```
p {
  display: inline-block;
  width: 150px;
  margin: 0 10px 0 0;
  background-color: #abc;
  vertical-align: top;
}
```

Figure 5-3

Alignement vertical top



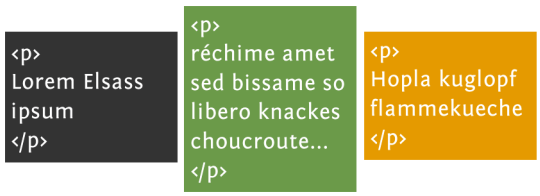
La valeur `middle` ouvre de nouveaux horizons de centrage vertical (figure 5-4).

Partie CSS

```
p {
  …
  vertical-align: middle;
}
```

Figure 5-4

Alignement vertical middle



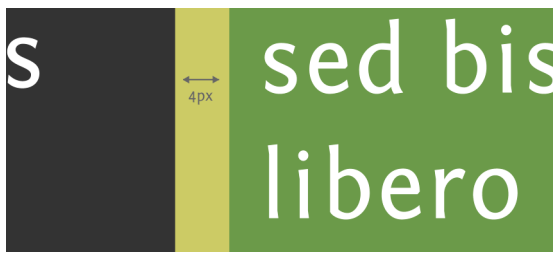
Outre l'avantage de pouvoir disposer des blocs dimensionnés côte à côte, le schéma de positionnement `inline-block` offre également l'opportunité de jouer avec l'alignement vertical des objets grâce à une combinaison astucieuse avec la propriété `vertical-align`.

Caractères invisibles (whitespace)

Comme tous les éléments de type `inline` ou de contenu textuel, les objets munis de la déclaration `display: inline-block` respectent la règle des caractères blancs (*whitespace*), c'est-à-dire que n'importe quel caractère espaçant deux éléments `inline-block` va immanquablement apparaître entre leurs deux boîtes. Chaque caractère « espace », « tabulation », « nouvelle ligne », « retour chariot » ou « nouvelle page » peut tenir lieu de *whitespace* et séparer les blocs visuellement (figure 5-5).

Figure 5-5

Illustration du « whitespace »

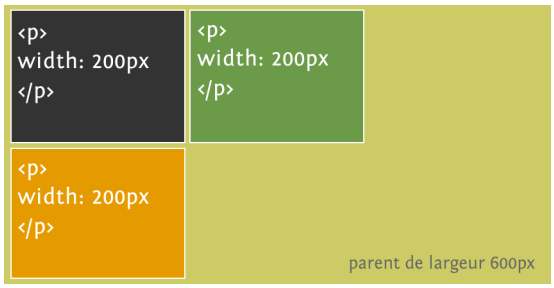


L'espace créé par le *whitespace* est de l'ordre de 4 pixels, mais cela varie selon les navigateurs et la taille de police.

Dans la grande majorité des cas, cette situation n'est guère gênante. Cependant, elle peut le devenir si votre présentation doit être calée au pixel près, par exemple pour un parent de 600 pixels de large devant contenir trois paragraphes de 200 pixels chacun (figure 5-6).

Figure 5-6

Ça déborde !



De nombreux concepteurs se sont penchés sur cette problématique, mais à l'heure actuelle, aucune solution idéale n'a été publiée, puisque la propriété CSS 3 idoine `white-space-collapse` n'est encore reconnue par aucun navigateur.

Certains tentent de jouer avec les marges négatives, d'autres sur l'espace entre lettres (`letter-spacing`). Le problème est que dans tous les cas, le rendu final dépendra – au minimum – de la police de caractères utilisée. L'on rencontre donc des techniques telles que :

```
parent {letter-spacing: -0.25em}
enfants {letter-spacing: normal;}
```

Toutefois, la valeur de l'espace entre lettres étant différente pour chaque police, cette méthode est à adapter au cas par cas :

- Verdana : -0.5em
- Arial : -0.25em
- Tahoma : -0.333em
- etc.

Un peu rédhibitoire, ne trouvez-vous pas ?

Une deuxième solution consiste à appliquer une marge externe droite négative à chacun des blocs enfants afin de « tracter » son successeur direct. La valeur de cette marge, après tests empiriques, a été estimée à 0,3 em, mais cela peut varier selon la fonte choisie :

```
chaque_enfant {margin-right: -0.3em;}
```

Cette technique, pour avoir été vérifiée sur un grand nombre de navigateurs, fonctionne de manière assez universelle, mais peut être considérée comme un *hack* puisque rien n'assure sa pérennité.

L'exemple complet est présenté ci-après.

Partie HTML

```
<div>
  <p>Lorem Elsass ipsum</p>
  <p>réchime amet sed bissame so libero knackes choucroute...</p>
  <p>Hopla kuglopf flammekueche</p>
</div>
```

Partie CSS

```
div p {
  display: inline-block;
  width: 150px;
  background-color: #abc;
  margin-right: -0.3em; /* on « tracte » les frères suivants */
}
```

Enfin, une dernière alternative consiste à supprimer les *whitespace* directement au sein du code HTML. Cela peut se faire en « collant » les éléments les uns aux autres :

```
<div>
  <p>Lorem Elsass ipsum</p><p>réchime amet sed bissame so libero knackes
choucroute...</p><p>Hopla kuglopf flammekueche</p> /* plus de whitespace entre les
éléments */
</div>
```

On peut encore transformer les *whitespaces* en commentaires HTML :

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

Ce qui donne au final :

```
<div>
  <p>Lorem Elsass ipsum</p><!--
  --><p>réchime amet sed bissame so libero knackes choucroute...</p><!--
  --><p>Hopla kuglopf flammekueche</p>
</div>
```

Cette dernière méthode, bien que parfois un peu « inesthétique », se révèle sans aucun doute la plus robuste et la plus pérenne actuellement.

Exercice pratique : dimensionner des liens horizontaux

Pour illustrer au mieux l'emploi judicieux du positionnement `inline-block`, prenons le cas d'un menu de navigation sous forme de liste (figure 5-7).

Partie HTML

```
<ul id="nav">
  <li><a href="#">Accueil</a></li>
  <li><a href="#">Société</a></li>
  <li><a href="#">Contact</a></li>
</ul>
```

Figure 5-7

Rendu initial avant mise en forme

- Accueil
- Société
- Contact

Imposons à présent un certain nombre de contraintes de rendu :

- Les liens présentent une couleur de fond grise et sont munis de marges internes et externes de 5 pixels.
- Ils doivent apparaître horizontalement, les uns à côté des autres.
- Chaque lien doit occuper une hauteur de 2,2 em.
- Le tout doit demeurer centré au sein du bloc parent `#nav`.

Procédons par étape en traitant un par un chacun des objectifs.

1. La couleur d'arrière-plan grise sur les liens ne pose aucun souci :

Partie CSS

```
#nav a {
  background: gray;
  margin: 5px;
  padding: 5px;
  color: white;
}
```

2. La disposition horizontale peut être réalisée aisément en modifiant le type de rendu des éléments de liste : la valeur de `display:`, `list-item` par défaut, peut être remplacée par `inline`.

```
#nav li {
  display: inline;
}
```

3. La contrainte suivante devient plus problématique. Il s'agit de conférer une hauteur (`height`) à un élément `<a>` de type en-ligne, ce qui n'est pas réalisable en l'état. Qu'à cela ne tienne ! Transformons-le en élément de type bloc :

```
#nav a {
  display: block;
  height: 2.2em;
  background: gray;
  margin: 5px;
  padding: 5px;
  color: white;
}
```

Hélas, ces dernières déclarations contredisent le comportement recherché précédemment, puisque les éléments s'affichent à nouveau empilés les uns sur les autres.

Nous savons que les différents schémas de positionnement hors du flux permettent aux éléments d'être dimensionnés quel que soit leur type de rendu initial. Optons pour le positionnement flottant et appliquons un flottement à gauche de chacun des liens :

```
#nav a {
  float: left;
  height: 2.2em;
```

```
  background: gray;
  margin: 5px;
  padding: 5px;
  color: white;
}
```

Ce nouveau mode de rendu, flottant, semble satisfaisant bien qu'il nous faille à présent gérer les éventuels cas de débordements et de contextes de formatage.

4. Cependant, les différents essais et tests vont démontrer que notre dernière mission consistant à centrer les liens horizontalement au sein de leur parent est extrêmement complexe à accomplir dans ce mode de positionnement flottant.

C'est alors que le rendu `inline-block` intervient pour notre plus grand bonheur. Débarrassons-nous du flottement hors flux au profit de ce type de positionnement méconnu :

```
#nav a {
  float: left;
  display: inline-block;
  height: 2.2em;
  background: gray;
  margin: 5px;
  padding: 5px;
  color: white;
}
```

L'alignement horizontal devient un jeu d'enfant puisqu'il suffit d'exploiter la propriété `text-align` en l'appliquant sur un ancêtre :

```
#nav {
  text-align: center;
}
```

Cette touche finale (figure 5-8) clôt l'exercice en beauté puisque nous avons accompli l'ensemble des missions avec brio, en demeurant dans le flux… tout en restant compatible avec les anciennes versions d'Internet Explorer, puisque ce rendu est parfaitement pris en charge par IE6 et IE7 lorsqu'il est appliqué à des balises en ligne telles que les liens `<a>`.

Figure 5-8

Résultat final : liens dimensionnés et centrés

Un rendu de tableau en CSS

Positionnement absolu, relatif, marges négatives, flottements, `clear`, problèmes de compatibilité, *hacks* et « bidouilles » pour Internet Explorer sont autant d'étapes nécessaires à la mise en œuvre d'un projet web de nos jours.

Concevoir des mise en pages globales sans positionnement hors flux devrait pourtant être possible en toute simplicité. Cela devrait même être l'objectif fondamental des spécifications CSS, sans quoi elles n'ont aucune raison d'exister et nous pourrions retourner sagement à nos mises en page via balises `<table>` imbriquées.

En parcourant ce livre, vous commencez à comprendre que le problème ne provient généralement pas des normes CSS, mais de certains navigateurs retardataires. Là encore, la bonne recette existe depuis CSS 2.1 et elle est encore plus simple qu'on ose l'imaginer.

Quelle est cette solution ? Les tableaux de mise en page, pardi !

Ne vous méprenez pas (et reposez cette tomate, s'il vous plaît). Je ne vous parle bien sûr pas de l'antique mise en page usant des balises HTML `<table>` et `<td>`, mais bel et bien d'une conception via des propriétés CSS conçues pour cela, sans interférer dans la structure HTML qui demeurera parfaitement sémantique.

Cette alchimie est rendue possible grâce à de nouvelles valeurs de la propriété `display`. Vous connaissiez `block`, `inline`, `none` et je vous ai présenté `inline-block`. Laissez-moi à présent vous décrire `table`, `table-cell`, `table-row` et consœurs (figures 5-9 et 5-10).

L'ampleur du phénomène est grandissant dans le monde des concepteurs web avant-gardistes, tant et si bien qu'un livre anglophone publié par Sitepoint.com est entièrement dédié à cette technique : *Everything you know about CSS is wrong !* [Tout ce que vous pensez connaître à propos de CSS est faux] (Rachel Andrew & Kevin Yanks, SitePoint, 2008).

Figure 5-9

Des colonnes flottantes, hors du flux

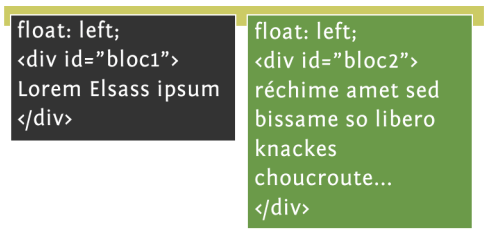
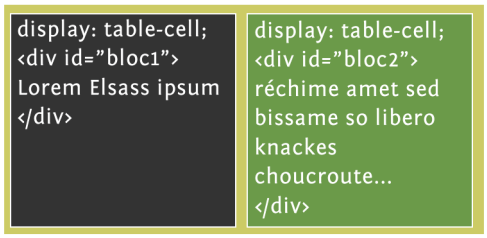


Figure 5-10

Des colonnes avec display: table-cell



COMPATIBILITÉ Valeurs tabulaires de la propriété display

Les valeurs tabulaires de la propriété `display` que je vais traiter dans cette partie sont reconnues par tous les navigateurs récents, y compris Internet Explorer 8, Firefox 2, Chrome 2, Safari 4.0, Opera 9.6 et même Konqueror 3.5.7. Seules manquent à l'appel les versions IE6 et IE7.

L'arrivée d'Internet Explorer 8 va véritablement changer la donne dans ce domaine, tout particulièrement puisqu'il interprète parfaitement ce mode de rendu très prometteur.

table, table-cell et table-row

La propriété `display` dispose d'un large panel de valeurs relatives aux rendus de forme tabulaire, dans le but d'afficher les éléments comme s'il s'agissait de tableaux, de cellules ou de lignes.

Tableau 5-2 Différentes valeurs relatives aux tableaux répertoriées dans les spécifications pour la propriété display

Valeur	Spécification	Correspondance HTML
`display: table`	Rendu de type tableau pour un élément, c'est-à-dire de type bloc mais qui n'occupe que la largeur du contenu.	`<table>`
`display: inline-table`	Comportement de table de type en-ligne pour un élément : c'est un bloc rectangulaire qui participe à un contexte de mise en forme en-ligne.	
`display: table-row`	L'élément doit s'afficher tel une rangée de cellules.	`<tr>`
`display: table-row-group`	L'élément regroupe une ou plusieurs rangées de cellules.	`<tbody>`
`display: table-header-group`	Comme pour `table-row-group`. Visuellement, cet élément est toujours affiché avant toutes les autres rangées et groupes de rangées et après une éventuelle légende.	`<thead>`
`display: table-footer-group`	Comme pour `table-row-group`. Visuellement, cet élément est toujours affiché après toutes les autres rangées et groupes de rangées et après une éventuelle légende.	`<tfoot>`
`display: table-column`	L'élément représente une colonne de cellules.	attribut `col`
`display: table-column-group`	L'élément regroupe une ou plusieurs colonnes.	attribut `colgroup`
`display: table-cell`	L'élément représente une cellule de tableau.	`<td>` et `<th>`
`display: table-caption`	Légende d'un tableau, positionnée par défaut en haut du tableau (mais cette disposition peut être modifiée via la propriété `caption-side`).	`<caption>`

Les éléments dont la propriété `display` a pour valeur `table-column` ou `table-column-group` ne sont pas rendus (exactement comme si celle-ci avait été `none`), mais ont une certaine utilité, leurs attributs pouvant donner éventuellement un certain style aux colonnes (figure 5-11).

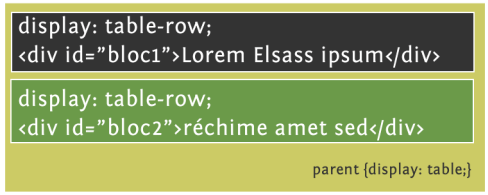


Figure 5-11

table-row : les blocs deviennent des rangées « conteneurs de cellules »

Les présentations sommaires ayant été faites, voici à présent comment réaliser simplement une mise en page de trois colonnes avec des blocs d'en-tête et de pied de page (figure 5-12).

Partie HTML

```
<div id="header">en-tête</div>
<div id="main">
  <p id="menu">ici le menu</p>
  <p id="content">ici le contenu</p>
  <p id="news">ici les news</p>
</div>
<div id="footer">pied de page</div>
```

Partie CSS

```
#header {
  background: #555;
  color: white;
}
#main {
  display: table;
  width: 100%;
}
#menu, #news {
  width: 25%;
  display: table-cell;
  background: #bbb;
}
#content {
  display: table-cell;
}
#footer {
  background: #555;
  color: white;
}
```

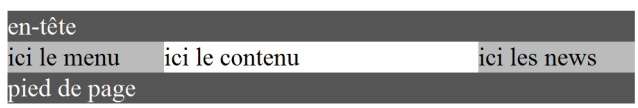


Figure 5-12

Mise en page sur trois colonnes avec en-tête et pied de page

Quelle différence avec HTML <table> ?

Après vous avoir rabâché durant des années qu'il fallait abandonner les mises en page avec des tableaux sous peine de brûler en enfer, voilà que je viens chambouler toutes vos certitudes en prononçant l'innommable « `table` ». Cependant, vous aurez saisi qu'il s'agit bien de CSS (`display: table`) et non de HTML (`<table>`) et que la différence est colossale.

Les tableaux HTML pour la mise en page impliquent un appauvrissement sémantique évident, sans oublier leur tendance à devenir trop lourds, médiocres en terme d'accessibilité et de référencement. Je rappelle toutefois que les éléments `<table>` sont toujours nécessaires pour structurer des données tabulaires.

Employer une propriété de rendu CSS va nous permettre d'appliquer cette mise en page sur des éléments HTML ayant du sens : `<h1>`, `<p>`, `<ul>`, mais aussi `<header>`, `<footer>`, `<aside>`, etc. en HTML 5.

Il n'y a donc pas lieu de se sentir coupable d'employer cette méthode de positionnement, tout comme il est tout à fait naturel d'utiliser `display: inline` pour afficher une liste horizontale de liens. Il ne s'agit pas de modifier la sémantique des éléments de liste, mais simplement leur présentation à l'écran. Il en va exactement de même avec `display: table-cell`.

> **À RETENIR Spécificités des rangées et légendes**
>
> Les valeurs de rangées (`display: table-row`) et de légende (`display: table-caption`) présentent des particularités liées à leur rendu : `table-row` ne comprend pas la propriété `padding`, tandis qu'il ne peut y avoir qu'un seul `table-caption` au sein du tableau.

Particularités du modèle tabulaire

La structure de rendu sous forme de tableau CSS comporte un certain nombre de singularités qui la distingue des autres modèles dans le flux : la création spontanée d'objets anonymes, l'alignement vertical et la répartition fluide des enfants au sein de leur parent.

Éléments anonymes

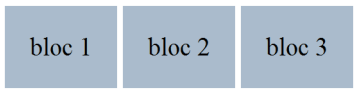
Chaque élément de rendu tableau en CSS crée automatiquement des objets de table anonymes autour de lui-même, c'est-à-dire que les éléments de structure manquants sont créés par le navigateur.

Cela signifie que, dans le cas d'un élément affiché sous forme de cellule de tableau (`display: table-cell`), il ne vous est pas nécessaire de l'entourer d'éléments de structure additionnels tels que les rangées de cellules (`display: table-row`) : celles-ci sont implicitement établies. Il en est de même pour le conteneur global (`display: table`).

Prenons l'exemple d'une présentation sous forme d'une grille de trois cellules telle que représentée sur l'image 5-13.

Figure 5-13

Trois cellules



À l'époque (pas si) lointaine des mises en page en tableaux HTML, le code correspondant aurait été celui-ci :

```
<table summary="">
  <tr>
    <td>bloc 1</td>
    <td>bloc 2</td>
    <td>bloc 3</td>
  </tr>
<table>
```

Passé à la moulinette « table CSS », la structure HTML devient entièrement libre : à nous de sélectionner les balises les plus pertinentes selon leur rôle et fonction. Dans le cas présent, je vais opter pour la générique `<div>`, mais j'insiste sur le fait que toutes les balises sont susceptibles de correspondre, selon le sens qu'elles apportent :

```
<div id="main">
  <div class="row">
    <div>bloc 1</div>
    <div>bloc 2</div>
    <div>bloc 3</div>
  </div>
<div>
```

Les styles correspondants pourraient ressembler à cela :

```
#main {
  display: table;
  border-collapse: separate; /* pour apercevoir les séparations */
  border-spacing: 4px;
}
.row {
  display: table-row;
}
.row > div {  /* les div de premier niveau uniquement */
  display: table-cell;
  padding: 1em;
  background-color: #abc;
}
```

La simple transposition d'une structure de table HTML en éléments `<div>` n'offre que peu d'intérêt, vous en conviendrez. Et c'est d'autant plus vrai que le nombre d'éléments imbriqués demeure identique.

En revanche, l'exercice devient autrement plus captivant si l'on tient compte des éléments anonymes que le navigateur se doit de créer automatiquement. Ainsi, le bloc nommé `row` désignant la rangée de cellules, de même que le conteneur général `container` peuvent être retranchés pour n'obtenir au final qu'un code réduit au minimum.

Partie HTML

```
<div>bloc 1</div>
<div>bloc 2</div>
<div>bloc 3</div>
```

Partie CSS

```
div {
  display: table-cell;
  padding: 1em;
  background-color: #abc;
}
```

Le rendu final doit être identique à l'exemple initial (à l'exception des espacements entre cellules), et ce, dans tous les navigateurs depuis Internet Explorer 8. C'est plutôt pratique, non ?

Remplissage de l'espace restant

Abordons une autre particularité intrinsèque du modèle de rendu tabulaire : la fluidité de l'occupation de l'espace. Les éléments de type cellule (`table-cell`) ou rangée (`table-row`) se répartissent par défaut de façon à meubler toute la surface disponible en largeur pour les cellules ou en hauteur pour les lignes.

L'intérêt devient rapidement évident dans le cas de présentations fluides ou lorsque le contenu peut varier : si on ajoute ou supprime un élément, les autres se répartissent dans l'espace devenu libre.

Prenons l'exemple de trois blocs de paragraphes déclarés en rangées (`table-row`).

Partie HTML

```
<div>
  <p>ceci est un paragraphe</p>
  <p>ceci est un autre paragraphe</p>
  <p>ceci est un 3è paragraphe</p>
</div>
```

Partie CSS

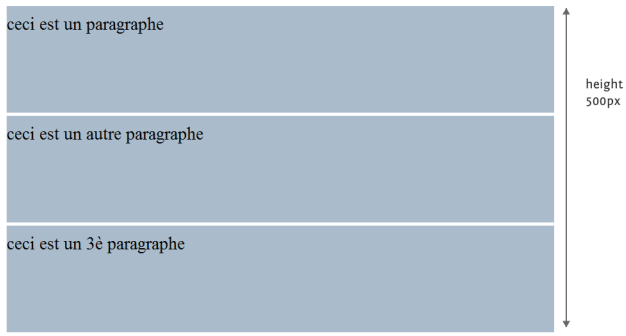
```
div {
  display: table;
  width: 500px;
  height: 500px;
  border-spacing: 3px;
}
p {
```

```
  display: table-row;
  background-color: #abc;
}
```

Vous constatez que les trois paragraphes se répartissent pour occuper toute la hauteur (500 pixels) de leur parent (figure 5-14).

Figure 5-14

Remplissage vertical



Allons plus loin dans cet exemple. Supprimez n'importe quel paragraphe de votre code HTML et observez le rendu obtenu : les deux paragraphes restants couvrent à présent chacun la moitié de la surface du conteneur `<div>`. Et s'il ne subsiste qu'un seul paragraphe, il remplira tout l'espace à lui tout seul.

La démonstration est bien entendu reproductible si l'on déclare les paragraphes en cellules de tableau (`table-cell`). Pour basculer de la vue verticale à une lecture horizontale des éléments, vous n'avez qu'un mot à changer : `row` en `cell` (figure 5-15) :

Partie HTML

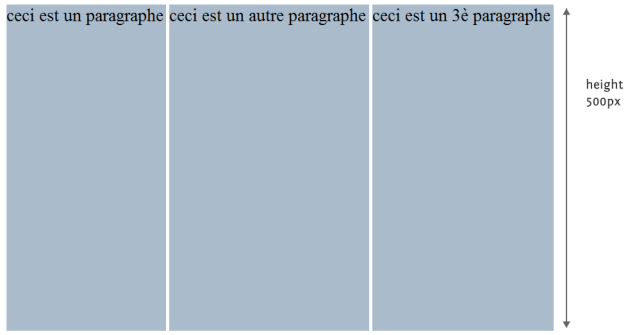
```
<div>
  <p>ceci est un paragraphe</p>
  <p>ceci est un autre paragraphe</p>
  <p>ceci est un 3è paragraphe</p>
</div>
```

Partie CSS :

```
div {
  display: table;
  width: 500px;
  height: 500px;
  border-spacing: 3px;
}
p {
  display: table-cell;
  background-color: #abc;
}
```

Figure 5-15

Remplissage horizontal



Alignement vertical

Les cellules de tableau HTML (`<td>`) sont réputées pour bénéficier d'un avantage tenant du Saint Graal du concepteur web : pouvoir appliquer la propriété `vertical-align` et en tirer toutes les vertus en terme d'alignement vertical des contenus.

Qu'à cela ne tienne ! `vertical-align` est parfaitement adéquate pour le modèle tabulaire CSS et peut être affectée à un élément en `display: table-cell` pour en aligner le contenu. Un vieux rêve d'intégrateur web se réalise sous nos yeux : celui de pouvoir intuitivement centrer verticalement n'importe quel élément en HTML.

En voici la preuve (figure 5-16).

Partie HTML

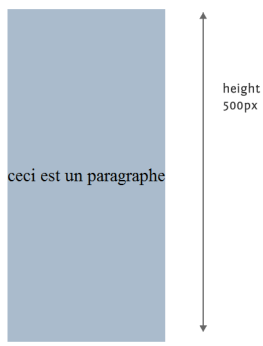
```
<p>ceci est un paragraphe</p>
```

Partie CSS

```
p {
  display: table-cell;
  vertical-align: middle;
  height: 500px;
  background-color: #abc;
}
```

Figure 5-16

Centrage vertical



Ordre d'empilement

Certaines valeurs spécifiques du modèle d'affichage sous forme de tableau modifient l'ordre d'affichage des éléments empilés verticalement :

- Un élément déclaré en `table-header-group` se placera avant ses frères (figure 5-17).
- Un élément en `table-footer-group` se positionnera à la suite de ses frères.
- Un élément en `table-caption` apparaîtra par défaut tout en haut de la pile, mais peut être déplacé tout en bas s'il est accompagné d'une déclaration `caption-side: bottom`.

Partie HTML

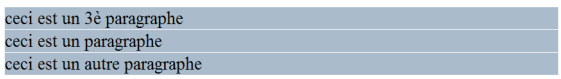
```
<div>
  <p>ceci est un paragraphe</p>
  <p>ceci est un autre paragraphe</p>
  <p class="last">ceci est un 3è paragraphe</p>
</div>
```

Partie CSS

```
div {
  width: 500px;
  display: table;
  border-collapse: collapse;
}
p    {
  display: table-row;
  margin: 0;
  background-color: #abc;
  border: 1px solid #fff;
}
p.last {
  display: table-header-group;
}
```

Figure 5-17

table-header-group



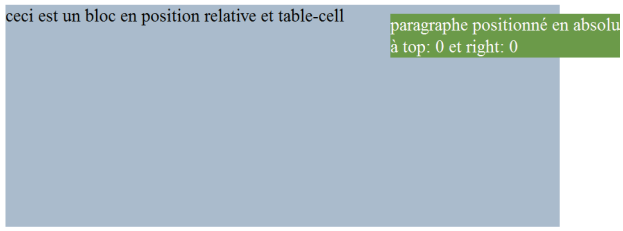
Tables et positionnements

La dernière particularité de ce modèle tabulaire est moins enthousiasmante que les précédentes. Elle tient au fait que les spécifications sont parfois floues sur les possibilités de combinaisons entre les schémas de positionnement. Celles-ci stipulent notamment que les effets du positionnement relatif sur les éléments en mode de rendu de tableau sont… indéfinis.

En clair, il n'est pas possible de faire bénéficier les éléments tabulaires de la position relative, ce qui peut être gênant puisqu'ils ne peuvent ainsi pas servir de référents pour des contenus positionnés en absolu (figure 5-18).

Figure 5-18

Positionnement absolu au sein d'un table-cell



Cette limite peut être contournée en ajoutant un enfant neutre, `<div>` par exemple, au sein de la cellule devant être positionnée en relatif. L'élément neutre bénéficiera du positionnement en lieu et place de son parent.

Propriétés spécifiques aux tableaux

Certaines propriétés CSS sont spécifiques aux éléments de tableaux. Il s'agit de `table-layout`, `border-collapse`, `border-spacing`, `empty-cells` et `caption-side`.

table-layout

Un double algorithme définit le rendu des structures tabulaires :

- Le modèle automatique (`table-layout: auto`) appliqué par défaut confère le pouvoir aux contenus des cellules. Ces derniers déterminent la largeur des colonnes : plus le contenu est dense, plus la colonne est large. Les largeurs `width` renseignées pour la table ou ses cellules ne seront qu'indicatives.
- Le modèle fixe (`table-layout: fixed`) se base sur la largeur réelle du tableau et de ses colonnes, et ne dépend pas du contenu.

Les figures 5-19 et 5-20 illustrent ce comportement et reposent sur le code suivant.

Partie HTML

```
<div id="main">
  <div>Lorem Elsass ipsum réchime amet bissame so libero. Leo Richard Schirmeck tellus
  ➥ kartoffelsalad set gewurztraminer morbi ante. Hopla ! </div>
  <div>Lorem Elsass ipsum réchime amet bissame so libero.</div>
  <div>Lorem Elsass. Hopla ! </div>
</div>
```

Partie CSS

```
#main {
  width: 100%;
  display: table;
  table-layout: auto;
}
#main > div {
  display: table-cell;
```

```
  border: 2px solid white;
  background-color: #555;
  color: white;
}
```

Figure 5-19

table-layout: auto - Ajustement par défaut selon la hauteur du plus long contenu

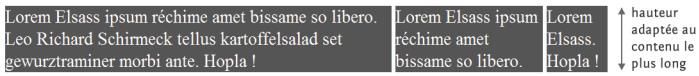
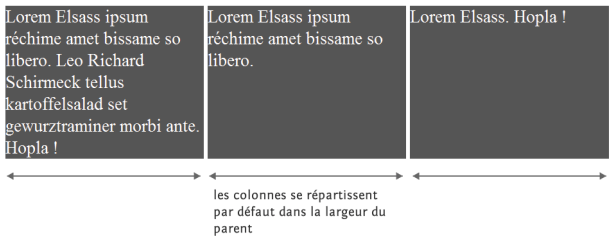


Figure 5-20

table-layout: fixed - Répartition par défaut dans toute la largeur du parent



Définir l'algorithme d'affichage en mode fixe a plusieurs avantages :

- Dès qu'un élément de colonne a une valeur `width`, alors cette valeur est retenue pour la largeur de toute la colonne. Les éventuelles colonnes restantes se partagent équitablement l'espace horizontal restant de la table.
- Le navigateur peut commencer à afficher la table dès la réception de la première rangée. De manière générale, cet algorithme accélère les traitements et l'affichage par le navigateur (pas de *reflow* inutile).
- Les contenus n'affectent pas les largeurs des colonnes. Toute cellule s'appuie sur la propriété `overflow` pour déterminer le rognage, ou non, du contenu qui a déborderait.

Dans le cas de tableaux de données complexes, mon expérience m'a appris à apprivoiser ce mode de rendu plus strict et robuste que le rendu automatique… et qui est reconnu dès Internet Explorer 5 (et les autres navigateurs également, bien entendu).

border-collapse

Comme c'est le cas pour les tableaux HTML, la propriété `border-collapse` (figure 5-21) détermine si les bordures de la table et entre les cellules doivent être séparées (valeur `separate`) ou fusionnées (valeur `collapse`).

```
#main {
  border-collapse: separate;
  border-spacing: 6px;
}
```

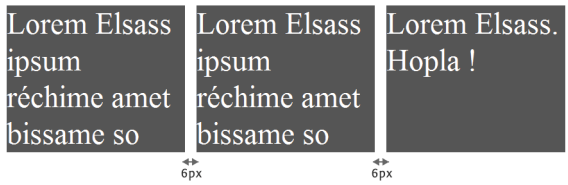


Figure 5-21

Propriété border-collapse associée à border-spacing

border-spacing

La propriété CSS `border-spacing`, reconnue à partir d'Internet Explorer 8 et équivalente à l'attribut HTML `cellspacing` devenu obsolète, spécifie la distance qui sépare les bordures de cellules adjacentes. Lorsque deux valeurs sont renseignées, la première désigne l'espacement horizontal et la deuxième le vertical. La valeur de cette propriété devient automatiquement nulle lorsque `border-collapse` a pour valeur `collapse`.

```
#main {
  border-collapse: separate;
  border-spacing: 6px;
}
```

caption-side

Cette propriété, reconnue elle aussi depuis IE8, indique la position de la boîte de la légende en fonction de celle de la table. Les valeurs acceptées sont `top` (par défaut) et `bottom`, mais aussi `left` et `right`, même si ces dernières ne sont actuellement comprises que par Firefox.

```
#main {
  border-collapse: separate;
  border-spacing: 2px;
  caption-side: bottom;
}
```

Grâce à cette propriété, il devient possible de réordonner verticalement du contenu, par exemple de faire remonter un élément au dessus de ses frères, comme cela a été vu précédemment.

empty-cells

La propriété `empty-cells` (à ne pas confondre avec le sélecteur CSS 3 `:empty` évoqué dans un autre chapitre) gère l'affichage des cellules vides : la valeur `hide` masque les bordures de la cellule. Lorsque cette propriété est appliquée au sein d'une table ne comportant qu'une seule rangée, les cellules vides disparaissent complètement et les autres cellules se réorganisent sans elles (figure 5-22). Cette propriété est prise en compte à partir d'Internet Explorer 8 et chez les autres navigateurs, bien entendu.

Partie HTML

```
<div>
  <p>ceci est un paragraphe</p>
```

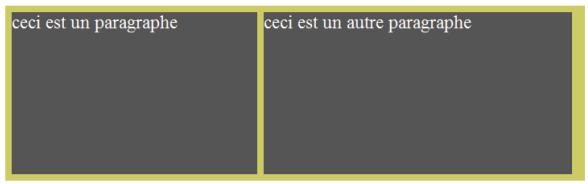
```
  <p>ceci est un autre paragraphe</p>
  <p></p>
</div>
```

Partie CSS :

```
div {
  display: table;
  width: 500px;
  height: 150px;
  border-spacing: 6px;
  background: #CCCC66;
}
p {
  display: table-cell;
  empty-cells: hide;
  background-color: #555;
}
```

Figure 5-22

empty-cells



Alternative pour IE6 et IE7

Un dernier piège nous empêche de nous jeter massivement sur ce genre de technique : elle ne fonctionne qu'à partir d'Internet Explorer 8. Il est parfaitement inutile de tenter de l'appliquer sur IE7 ou IE6 : les éléments demeureront traités tels des blocs classiques.

Sachez cependant que dans de nombreuses circonstances, nous pourrons pallier cette lacune par une alternative acceptable bien qu'imparfaite : le positionnement flottant.

Dans une optique de dégradation gracieuse, la démarche est de proposer une feuille de styles dédiée aux anciennes versions d'Internet Explorer, via commentaires conditionnels (voir le chapitre sur la résolution d'erreurs).

Partie HTML

```
<!--[if lte IE 7]>
  <link rel="stylesheet" type="text/css" href="styles-ie7.css" />
<![endif]-->
```

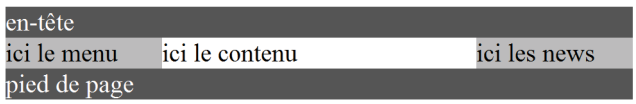
À présent, assurez-vous que cette feuille de styles, chargée après la feuille générale, remplace les dispositions via `display: table` et `table-cell` par des déclarations plus facilement reconnues par IE7 (figure 5-23), par exemple.

Fichier styles-ie7.css

```
#menu, #news {
  float:left;
  width: 25%;
}
#content {
  float:left;
  width: 50%;
}
#footer {
  clear: both;
}
```

Figure 5-23

La version dégradée via flottements



En observant cette astuce, votre réaction pourrait être : « mais quel est l'intérêt de s'embêter avec des `table-cell` pour les navigateurs récents si, de toute façon, il va falloir tout repenser en flottements pour les autres ? Autant se limiter aux flottements pour tout le monde, ça facilitera les choses ! »

Je comprends fort bien ce raisonnement pour l'avoir moi-même longtemps adopté. Cependant, après avoir passé la phase délicate d'adoption d'une nouvelle technique et après en avoir goûté les plaisirs, j'ai tendance à penser que l'effort à produire est négligeable comparé aux avantages que cette méthode procure aux navigateurs méritants :

- Les colonnes sont parfaites et toujours de même longueur, sans besoin de tricher.
- Les éléments alentours ne nécessitent pas de traitement de faveur (`clear` ou autre), puisque tout demeure dans le flux courant.
- La gestion des alignements verticaux, notamment du centrage, devient élémentaire.
- La fluidité des blocs est innée : fini le casse-tête du pied de page toujours collé en bas quelle que soit la longueur de la page, ou les éléments devant occuper toute la hauteur de page moins l'espace employé par l'en-tête ou le pied de page.
- Il est même envisageable de réordonner du contenu et de passer visuellement un élément prioritairement à d'autres en jouant avec la valeur `table-caption`.
- Etc.

Pour toutes ces raisons (et j'en oublie) et parce que la présentation sera irréprochable sur tous les navigateurs récents, j'ai envie de promouvoir ce mode d'affichage naturel. Les visiteurs sur IE6 et IE7 jouiront d'une expérience utilisateur simplement moins parfaite, à l'aide de flottements, de « faux-columns » et de `clear: both`.

Tableau récapitulatif

Vous trouverez ci-après un résumé du modèle de rendu tabulaire, des propriétés évoquées au sein de ce chapitre ainsi que de leurs compatibilités avec les navigateurs actuels.

Notez que ce tableau ne tient compte que des différentes versions contemporaines des navigateurs : Internet Explorer à partir de IE6, Firefox 3 et plus, Chrome 5 et plus, Safari 4 et plus, Opera à partir de la version 9. Les versions très anciennes des navigateurs n'ont pas été prises en compte.

Tableau 5-3 Modèle de rendu tabulaire

	IE6	IE7	IE8	Firefox	Chrome	Safari	Opera
`display:`*valeurs tabulaires*			OK	OK	OK	OK	OK
`table-layout`	OK	OK	OK	OK	OK	OK	OK
`border-collapse`	OK	OK	OK	OK	OK	OK	OK
`border-spacing`			OK	OK	OK	OK	OK
`caption-side`			OK	OK	OK	OK	OK
`empty-cells`			OK	OK	OK	OK	OK

Exercice pratique : hauteurs fluides

L'objet de cet exercice concret est de concevoir un gabarit de page classique, comprenant un bloc d'en-tête, deux blocs côte à côte (`aside` et `content`) et un bloc de pied de page. Les hauteurs des éléments `aside` et `content` doivent être fluides : si le contenu de l'un s'allonge, l'autre doit s'adapter également.

Le code HTML fourni est le suivant (figure 5-24) :

```
<div id="header">ici le bloc d'en-tête</div>
  <div id="main">
    <div id="aside">ici le bloc Aside</div>
    <div id="content">ici le bloc Content</div>
  </div>
<div id="footer">ici le pied de page Footer</div>
```

Commençons par attribuer une couleur de fond aux différents blocs afin de mieux les distinguer :

Partie CSS

```
#header { background: #fab; }
#aside { background: #abc; }
#content { background: #ddd; }
#footer { background: #efa; }
```

Figure 5-24

Blocs empilés par défaut

En vue de placer les blocs `aside` et `content` côte à côte, je choisis de fixer une largeur de 250 pixels au premier de ces éléments :

```
#aside { width: 250px; background: #abc; }
```

La suite va se dérouler très rapidement, ne soyez pas surpris. Transformons le rendu initial des blocs `aside` et `content` vers un modèle tabulaire :

```
#aside, #content { display: table-cell; }
```

Cette seule instruction permet d'accomplir plusieurs objectifs à la fois, dont ceux de les afficher horizontalement et que leurs hauteurs coïncident. Les deux éléments se considèrent comme des cellules de tableaux ; leur parent `#main` devient implicitement à la fois la rangée et le tableau global.

Pour fignoler en beauté, il suffit de déclarer `#main` en tableau de largeur 100 %. Ainsi, la boîte `#content` occupera toujours toute la surface restante (figure 5-25) :

```
#main {display: table; width: 100%}
```

Figure 5-25

Résultat final

ALLER PLUS LOIN D'autres démonstrations

Je me suis amusé au sein de mon espace de jeu personnel IE7nomore.com à tester un certain nombre de possibilités offertes par ce modèle tabulaire en CSS. Voici quelques cas d'étude qui peuvent retenir votre attention et dont le code-source n'attend que vous :

- un gabarit occupant toute la hauteur de la page avec un en-tête et un pied de page de hauteurs fixées en pixels :

▸ http://www.ie7nomore.com/css2only/table/

- une réorganisation des éléments où le premier paragraphe apparaît en dernier et le dernier en premier :

▸ http://www.ie7nomore.com/css2only/table-order/

- un menu dynamique et fluide réalisé en modèle tabulaire. Seule la transition est réalisée en CSS 3 :

▸ http://www.ie7nomore.com/css2only/flexibletab/

Rappelez-vous toutefois que tout ce qui nous semble magique pose encore des problèmes de reconnaissance sur les anciennes versions d'Internet Explorer. En l'occurrence, ce mode d'affichage n'est pas compris sur IE6 et IE7. Il faudra avoir recours à un positionnement plus classique pour ces ancêtres.

Grid et Template positioning

Parmi les brouillons de travail CSS 3, le module « positionnement » propose deux pistes à suivre dans l'avenir : le positionnement en grille (*grid positioning*) et le rendu par gabarits (*template layout*).

Positionnement en grille (grid positioning)

CSS 3 introduit de nouvelles propriétés dans la famille des schémas de positionnement, `grid-columns` et `grid-rows`, conjointement à une nouvelle unité de mesure, la grille (`gr`).

Pour comprendre le principe, voici un exemple issu des spécifications W3C :

```
div { grid-columns: 50% * 4em }
```

Ce code indique que le bloc `<div>` est composé d'une grille de trois colonnes : la première occupant la moitié de la surface à gauche, la troisième une largeur de 4 em à partir de la droite et une colonne centrale occupant le restant de l'espace.

Un second exemple, également extrait de la norme CSS 3, définit une grille telle que la première ligne occupe 4 em, puis autant de lignes que nécessaires dont les hauteurs sont alternées (0.25 em et 1 em) :

```
div { grid-rows: 4em (0.25em 1em); }
```

Le concept de positionnement en grille montre une volonté de se rapprocher du support d'impression et de caler les présentations au pixel près. L'idée n'est pas à rejeter, loin de là, mais l'implémentation de ce module, son manque de prise en charge actuel (aucun navigateur) et la complexité qui en découle ne plaident guère en faveur de ce nouveau schéma encore un peu jeune.

Positionnement à l'aide de gabarits (template positioning)

Toute aussi radicale, une autre piste, également en brouillon, se dégage des laboratoires du WC3 : le positionnement à l'aide de gabarits (*template layout module*).

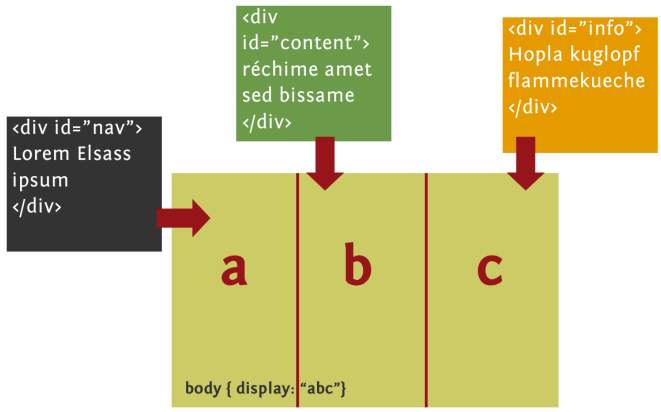
On conçoit visuellement un gabarit, composé de différents emplacements (*slots*) qui seront à même d'accueillir des éléments assignés. Voici un exemple simple :

```
body { display: "abc"}
#nav { position: a }
#content { position: b }
#info { position: c }
```

Le corps de page est composé de trois emplacements symbolisés par `a`, `b` et `c`, de la même taille. Chacun des trois éléments `#nav`, `#content` et `#info` occupe un emplacement déterminé par la propriété `position` (figure 5-26).

Figure 5-26

Base du template layout

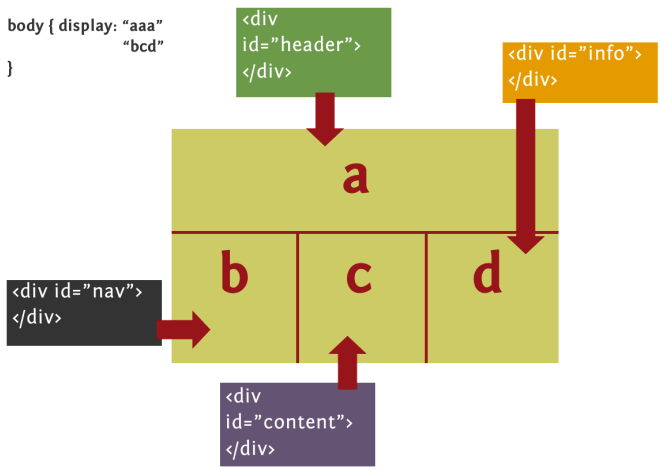


Dans l'exemple suivant, le bloc `#header` inséré dans l'emplacement `a` occupe trois unités d'espace, c'est-à-dire toute la largeur du document (figure 5-27) :

```
body { display: "aaa"
                "bcd"
}
#header { position: a }
#nav { position: b }
#content { position: c }
#info { position: d }
```

Figure 5-27

Un gabarit classique en template layout



Ajoutez à cela qu'il est possible de définir une hauteur pour chaque ligne d'emplacement, directement au sein de la propriété `display` :

```
body { display: "aaa" / 100px
                "bcd"
}
```

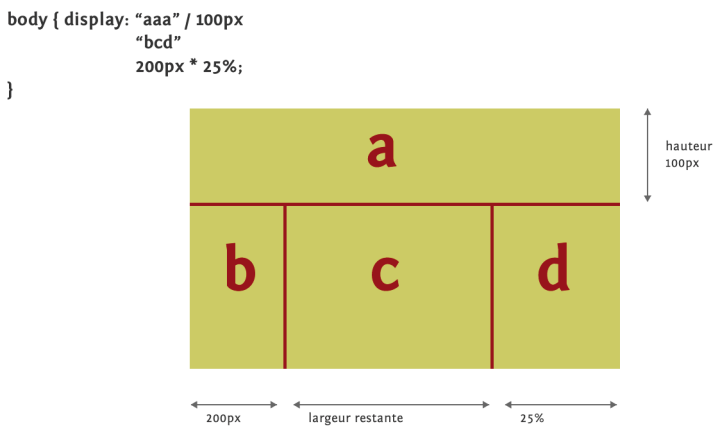
La notation des largeurs de colonnes peut se révéler plus complexe, puisque les concepts de largeur automatique, de minima et maxima sont pris en compte par ce module. Retenez pour l'exemple qu'il est possible d'appliquer cette syntaxe pour définir la largeur de chacune des trois colonnes :

```
body { display: "aaa" / 100px
                "bcd"
                200px * 25%;
}
```

Cet exemple se traduit par une largeur de 200 pixels pour la première colonne (largeur de l'emplacement `b`), une largeur de 25 % pour l'emplacement `d` et une largeur automatique pour la partie centrale `c` (figure 5-28).

Figure 5-28

template layout avec définition de largeurs et hauteurs



L'un des avantages de ce module est de proposer une séparation complète entre la structure et le rendu final : un élément HTML peut ainsi être contenu dans un bloc de pied de page, mais être affiché dans un emplacement situé dans l'en-tête, comme le montre cet autre exemple :

```
#footer .mentions { position: a}
```

D'autres avantages sont à tirer de ce concept, par exemple la grande facilité d'implémenter des gabarits différents selon les supports (écran, imprimante, mobile). Il suffit de modifier les emplacements, leur nombre ou leur taille pour que la présentation soit automatiquement adaptée à un écran de smartphone, par exemple.

Styles CSS classiques

```
body { display: "aaa"
               "bcd"
}
```

Styles pour petits écrans ou mobiles

```
@media screen and (max-width: 640px)
{
  body { display: "a"
                  "b"
                  "c"
                  "d" }
}
```

Pour en savoir plus sur ce schéma de positionnement plutôt passionnant et bien plus vaste que cette courte présentation, je vous invite à vous plonger dans la lecture d'un excellent article de Jérémie Patonnier publié à cette adresse :

▸ http://jeremie.patonnier.net/post/2009/07/16/CSS-3-Le-module-Template-Layout

EN ATTENDANT jQuery pour les retardataires

Bien que les spécifications soient loin d'être finalisées et qu'aucun navigateur, même parmi les plus avant-gardistes, ne propose la prise en charge de ce module, un plug-in jQuery a été conçu pour en fournir une implémentation complète sur un vaste panel de navigateurs : depuis IE6, Firefox 2, Opera 9.5, Safari 3 et Chrome 1.

Vous trouverez toutes les informations sur l'utilisation de ce plug-in, ainsi que la procédure de téléchargement et d'installation à l'adresse :

▸ http://code.google.com/p/css-template-layout/

Parfaitement fonctionnel et souvent mis à jour, cet outil présente cependant un inconvénient : le navigateur doit attendre que la fin de l'arbre du document et le code JavaScript soient chargés avant d'afficher correctement la page.

Le modèle de boîte flexible

Sous l'appellation *Flexible box model module*, CSS 3 signe son projet le plus avancé et prometteur puisque déjà compris par un certain nombre de navigateurs récents tels que Chrome, Safari et Firefox. Microsoft avait même laissé courir des rumeurs sur sa prise en charge, mais celle-ci est finalement absente de la version bêta publique d'Internet Explorer 9. Le suspense est entier pour la version finale du navigateur.

Le modèle de boîte flexible apparaît comme une extension du modèle de boîte classique défini en CSS 2.1, bénéficiant bien entendu de nouveaux potentiels, parmi lesquels la possibilité d'alterner entre une distribution horizontale ou verticale des éléments, disposer de largeurs fluides dans les deux sens et, surtout, de pouvoir définir l'ordre exact d'affichage des boîtes à l'écran.

display : box

Le modèle de boîte étendu est initié par la valeur `box` associée à une propriété que nous connaissons bien à présent : `display`. Il vous suffit d'affecter la déclaration `display: box` à un bloc pour lui faire adopter le modèle de boîte flexible et que ses éléments se disposent automatiquement les uns à côté des autres (figure 5-29).

Partie HTML

```
<div id="main">
  <p>une boîte</p>
  <p>encore une boîte</p>
  <p>et encore une troisième boîte</p>
  <p>et hop !</p>
</div>
```

Partie CSS

```
#main {
  display: box;
}
```

Enfantin, n'est-ce pas ?

En théorie, oui. Dans la pratique, ce module n'étant pas finalisé, il conviendra d'utiliser les préfixes `-moz-` pour Mozilla Firefox et `-webkit-` pour Chrome et Safari, ce qui nous donne :

Partie CSS

```
#main {
  display: -moz-box;
  display: -webkit-box;
  display: box;
}
```

Figure 5-29

Modèle de boîte flexible

Empilement vertical ou horizontal

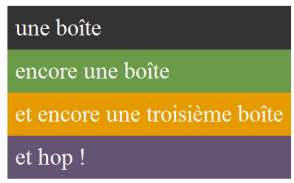
La propriété `box-orient` fait basculer les boîtes d'une distribution horizontale vers un empilement vertical (figure 5-30) :

```
#main {
  display: -moz-box;
  display: -webkit-box;
  display: box;
  -moz-box-orient: vertical;
```

```
  -webkit-box-orient: vertical;
  box-orient: vertical;
}
```

Figure 5-30

Modèle de boîte flexible vertical



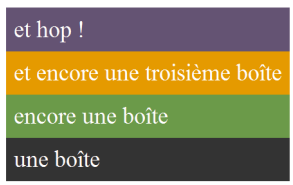
Ordre d'empilement

Comptant parmi les fonctionnalités les plus attractives de ce module, la propriété `box-direction` permet de trier les boîtes selon l'ordre d'apparition dans le flux ou selon l'ordre inverse (figure 5-31) :

```
#main {
  display: -moz-box;
  display: -webkit-box;
  display: box;
  -moz-box-orient: vertical;
  -webkit-box-orient: vertical;
  box-orient: vertical;
  -moz-box-direction: reverse;
  -webkit-box-direction: reverse;
  box-direction: reverse;
}
```

Figure 5-31

Modèle de boîte flexible vertical : box-direction



Il est même possible de définir explicitement l'ordre de distribution des boîtes grâce à la propriété `box-ordinal-group`. La distribution se fait du plus petit (le groupe 1) au plus grand (le groupe 2, puis 3, etc.). La valeur par défaut de cette propriété est `1`.

```
#main > p {
  -moz-box-ordinal-group: 2;
  -webkit-box-ordinal-group: 2;
  box-ordinal-group: 2;
}
```

```
#main > p:last-child {
  -moz-box-ordinal-group: 1;
  -webkit-box-ordinal-group: 1;
  box-ordinal-group: 1;
}
```

Cet exemple définit un groupe ordinal secondaire (valeur `2`) pour chaque boîte de paragraphe, à l'exception de la dernière qui, par conséquent, s'affichera… en premier (figure 5-32).

Figure 5-32

Modèle de boîte flexible et l'ordre des éléments

Flexibilité : remplir l'espace

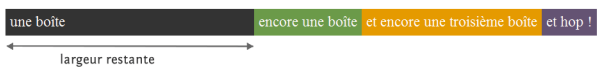
Comme son nom l'indique, le modèle de boîte flexible apporte en toute simplicité une dimension de fluidité aux éléments.

Une boîte devient flexible à partir du moment où elle se voit attribuer la propriété `box-flex` avec une valeur supérieure ou égale à `1`. Elle s'étend alors sur toute la surface disponible restante au sein de son parent (figure 5-33) :

```
#main {
  ...
  width: 100%;
}
#main > p:first-child {
  -moz-box-flex: 1;
  -webkit-box-flex: 1;
  box-flex: 1;
}
```

Figure 5-33

Modèle de boîte flexible : flexibilité



Lorsque plusieurs élément bénéficient de la même valeur de `box-flex`, ils se répartissent équitablement l'espace entre eux. La valeur de `box-flex` correspond à un rapport : une valeur de `2` sera deux fois plus large qu'une valeur de `1` :

```
#main > p {
  -moz-box-flex: 1;
  -webkit-box-flex: 1;
```

```
  box-flex: 1;
}
#main > p:first-child {
  -moz-box-flex: 2;
  -webkit-box-flex: 2;
  box-flex: 2;
}
```

Compatibilité

J'ai évoqué à plusieurs reprises que ce module comptait parmi les plus aboutis des projets innovants des spécifications CSS 3. Le modèle de boîte flexible est reconnu par Firefox 3, Chrome 5 et Safari 3. En revanche, il est encore totalement ignoré d'Opera et d'Internet Explorer, même si l'espoir plane de le retrouver sur IE9.

En attendant une prise en charge plus universelle de cette méthode de positionnement, il nous faudra une fois de plus nous tourner vers une alternative JavaScript. La plus complète semble être Flexie.js hébergée chez GitHub, dont la démonstration interactive est très séduisante :

- http://github.com/doctyper/flexie
- http://doctyper.github.com/flexie/playground/

Cette solution, très alléchante, me semble toutefois à prendre avec des pincettes, car elle n'en est encore qu'à un stade de développement très jeune.

Exercice pratique : centrer et réordonner des éléments

En guise d'illustration concrète, tentons de réaliser le cas de figure suivant :

- trois éléments disposés sur la même ligne au sein de leur parent ;
- l'ensemble des trois blocs doit être centré à la fois horizontalement et verticalement ;
- en bonus, le deuxième bloc doit s'afficher à la suite de tous ses frères.

Pour satisfaire ces différentes conditions, nous allons exploiter le modèle de boîte flexible, et plus particulièrement les propriétés `box-pack`, `box-align` et `box-ordinal-group`.

Commençons par poser le décor : un conteneur composé de trois enfants. Mon choix se porte sur une liste, mais rien ne vous empêche d'opter pour d'autres éléments.

Partie HTML

```
<ul>
  <li id="boite1">Boîte 1</li>
  <li id="boite2">Boîte 2</li>
  <li id="boite3">Boîte 3</li>
</ul>
```

La partie CSS se contente pour l'instant de doter ces blocs d'une couleur d'arrière-plan :

```
#boite1 {background: #6B9A49}
#boite2 {background: #E69B00}
#boite3 {background: #645373}
```

Le conteneur de liste est dimensionné et dispose lui aussi d'une couleur de fond. Chacun des éléments de liste a une largeur de 100 pixels et nous lui supprimons la puce par défaut (figure 5-34) :

```
ul {
  width: 500px; height: 200px;
  margin: 0; padding: 0;
  background: #CCCC66;
}
li {
  width: 100px;
  border: 1px solid white;
  list-style: none;
  color: white;
}
```

Figure 5-34

Aperçu initial avant modèle flexible

À présent, amusons-nous un peu avec le modèle flexible. Tout d'abord, appliquons ce contexte de rendu au conteneur de liste :

```
ul {
  display: -webkit-box;
  display: -moz-box;
  display: box;
}
```

Comme vous pouvez le constater, ses trois enfants adoptent instantanément une disposition horizontale tout en occupant toute la hauteur du conteneur (figure 5-35).

Figure 5-35

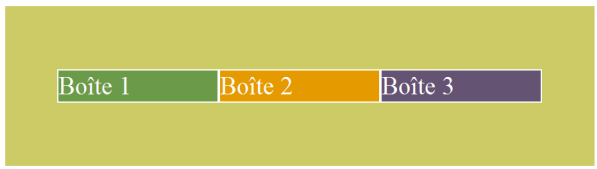
Boîtes flexibles

L'étape suivante consiste à centrer les éléments. La propriété `box-pack` gère l'alignement horizontal, tandis que `box-align` s'occupe de la verticalité. Une valeur commune de `center` suffit à positionner les trois boîtes au cœur de leur parent (figure 5-36) :

```
ul {
  display: -webkit-box;
  display: -moz-box;
  display: box;
  -webkit-box-align: center; /* alignement vertical */
  -moz-box-align: center;
  box-align: center;
  -webkit-box-pack: center; /* alignement horizontal */
  -moz-box-pack: center;
  box-pack: center;
}
```

Figure 5-36

Boîtes flexibles centrées

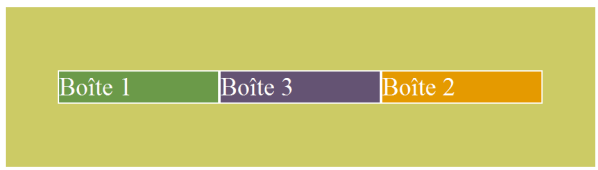


Notre dernière mission consiste à modifier l'ordre d'affichage des blocs pour placer la deuxième boîte en queue de peloton. C'est la propriété `box-ordinal-group` qui va s'en charger très simplement, en lui affectant une valeur supérieure à 1 (figure 5-37) :

```
#boite2 {
  background: #baf;
  -webkit-box-ordinal-group: 2;
  -moz-box-ordinal-group: 2;
  box-ordinal-group: 2;
}
```

Figure 5-37

Résultat final



Pour parfaire ce résultat, il ne reste plus qu'à en assurer la compatibilité sur les navigateurs déficients tels qu'Opera et Internet Explorer 6, 7 et 8. Pour cela, nous allons nous tourner vers le très efficace Flexie.js (http://github.com/doctyper/flexie) associé à une bibliothèque JavaScript telle que jQuery.

Ajoutons ces fichiers à la fin de notre document HTML :

```
…
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"
➥ type="text/javascript"></script>
<script src="flexie.js" type="text/javascript"></script>
</body>
```

Et le tour est joué !

Admirer le résultat en ligne

- http://ie7nomore.com/fun/flexie/

Revue des différents schémas de positionnement

Nous sommes actuellement dans une période charnière : les retardataires IE6 et IE7 nous imposent malheureusement encore des restrictions importantes dans nos choix de positionnements des éléments, mais l'avenir est en marche et la démocratisation des versions d'Internet Explorer 8, puis 9, va considérablement étendre notre champ de possibilités.

En théorie, certains schémas de positionnement, tels que le modèle de table ou les boîtes flexibles, semblent offrir tous les avantages que l'on puisse imaginer tout en restant dans le flux, donc sans « casser » la page. En production néanmoins, nous restons limités à des choix non adaptés ou mal implémentés comme le positionnement flottant, ou à des alternatives de type JavaScript pour émuler certains comportements sur les navigateurs dinosaures.

Le tableau ci-après récapitule les différents schémas de positionnement actuels et en cours d'élaboration.

Tableau 5-4 Les différents schémas de positionnement CSS

Schéma	Exemple	Flux	CSS	Navigateurs	Alternative
En flux	Par défaut	oui	CSS 1	tous	non
Flottement	`float: left`	hybride	CSS 1	tous (mais erreurs)	non
Absolu	`position: absolute`	non	CSS 2	tous	non
Fixé	`position: fixed`	non	CSS 2	tous (IE7+)	JavaScript
Relatif	`position: relative`	oui	CSS 2	tous	non
Inline-block	`display: inline-block`	oui	CSS 2.1	tous (mais erreurs)	« Hack » IE
Table-layout	`display: table`	oui	CSS 2.1	tous (IE8+)	non
Grid-layout	`grid-columns: 100px * 150px`	oui	CSS 3	aucun	non

Schéma	Exemple	Flux	CSS	Navigateurs	Alternative
Template layout	`display: "abb"`	hybride	CSS 3	aucun	JavaScript stable
Flexible box model	`display: box`	oui	CSS 3	Safari, Chrome, Firefox	JavaScript

6

Résolution d'erreurs

Ce chapitre s'aventure dans le monde obscur et redoutable des différences d'affichage selon les navigateurs du marché. Il parcourt les contrées malfamées des *hacks* et autres « bidouilles », du mécanisme étrange de *HasLayout* et de toutes les solutions permettant d'harmoniser des pages sur l'ensemble des navigateurs.

Connaître le rendu par défaut des éléments

Il ne s'agit pas d'un scoop : un site bien conçu, voire validé par l'outil du W3C, ne sera pas obligatoirement affiché de la même façon partout. En effet, chaque navigateur a ses spécificités (marges différentes sur les éléments) et ses erreurs de rendu.

Outils de vérification

Sur www.alsacreations.com, nous proposons un outil conçu par Julien Royer et spécialisé dans cette tâche : en accédant à la page alsacreations.com/static/rendu/, un tableau contenant la plupart des balises HTML usuelles est créé selon le navigateur que vous employez. Pour chacun de ces éléments, les valeurs par défaut des propriétés `display`, `margin`, `padding`, `font-size` et `font-weight` sont renseignées. Si vous visitez cette page à l'aide d'un autre navigateur, soyez sûr que certaines valeurs seront différentes !

D'une manière plus générale, le site web IEcss.com permet d'afficher les styles par défaut de tous les éléments HTML sur l'ensemble des versions d'Internet Explorer depuis IE6 jusqu'à IE9 et de comparer leurs différences. Il s'agit d'un site plus qu'intéressant pour comprendre les disparités d'affichage d'un navigateur à un autre (figure 6-1).

Figure 6-1

IEcss.com

Internet Explorer User Agent Style Sheets

The UA Style Sheet is a simple set of css styles that each web browser uses before any other css styles are applied.

This chart lists and compares the different default style sheets used to render HTML in the four major versions of Internet Explorer; IE6, IE7, IE8, and IE9 Platform Preview.

You can download each of these UA stylesheets by using the links at the top of this chart.

	IE6	IE7	IE8	IE9
a	color: #00F; text-decoration: underline;	color: #06C; text-decoration: underline;		
a:visited	color: #800080;			
address	display: block; font-style: italic;			
b	font-weight: bold;			
bdo	direction: rtl; unicode-bidi: bidi-override;			
blockquote	display: block; margin: 14pt 30pt;		display: block; margin: 1em 40px;	
body	display: block; margin: 15px 10px; zoom: 1;		display: block; margin: 8px; zoom: 1;	

Et si ce n'était pas une erreur ?

Ce n'est pas forcément une bonne nouvelle pour vous, mais au fur et à mesure de leurs versions, les navigateurs sont de plus en plus stables, de plus en plus conformes aux spécifications et de moins en moins bogués. Le corollaire est que, par déduction, la source du problème est souvent… vous.

En effet, les erreurs que l'on nous soumet sur le forum Alsacréations ne sont généralement pas dues à des manquements des navigateurs, mais plus souvent à des lacunes humaines :

- une mauvaise compréhension des schémas de positionnement (à savoir qu'il est attendu qu'un élément hors flux ou flottant « dépasse » de son parent) ;
- des imbrications d'éléments non autorisées (par exemple, un élément HTML de type `block` au sein d'un élément `inline` ou d'un paragraphe) ;
- des erreurs de codage (balise fermante oubliée, séparateur point-virgule omis en CSS, etc.) ;
- une mauvaise compréhension du modèle de boîte, par exemple :
  - les marges internes et les bordures s'ajoutent à `width` lorsqu'il s'agit de calculer l'espace occupé par l'élément ;
  - tous les éléments HTML de type bloc (à l'exception de `<div>`) disposent de marges externes non nulles par défaut ;
  - les fusions de marges modifient l'espacement entre deux éléments.

En résumé, il est parfois plus sage de commencer par bien consolider ses acquis plutôt que de blâmer les agents utilisateurs dont le seul tort est de respecter scrupuleusement les spécifications officielles.

COMPRENDRE Espaces sous les images

Certains comportements d'affichage, aussi curieux qu'ils paraissent, sont pourtant parfaitement conformes. C'est le cas du célèbre espace de quelques pixels que l'on peut observer sous les images (`<img>`), impossible à évacuer en redéfinissant les `margin`, `padding` et autre `line-height` à zéro.

L'explication est pourtant simple : `<img>` est un élément en ligne de type `inline-block` qui se positionne par défaut sur la ligne de texte (*baseline*), donc laisse un espace en dessous pour les lettres telles que « p » ou « q ».

Pour remédier à ce comportement, il est possible d'agir sur la propriété `vertical-align` en lui conférant la valeur `bottom`, ou encore de modifier le mode de rendu de l'image à l'aide de `display: block`.

Faut-il utiliser les hacks ?

Un *hack* (ou « bidouille », en français) est une technique fondée sur l'empirisme, destinée à éviter, au cas par cas, certaines différences de rendu entre les navigateurs et permettre aux plus défectueux d'aboutir à nos fins, soit en détournant une propriété de son usage, soit en utilisant des codes supplémentaires pour pallier les manques.

Les astuces de ce genre sont généralement déconseillées, car elles reposent sur une déficience temporaire d'une version de navigateur qui peut être corrigée du jour au lendemain.

Exemples de hacks

L'un des hacks les plus connus est le *simplified box model hack*, dont l'astuce consiste à insérer une barre oblique inverse (\) au sein du nom de la propriété, par exemple : `w\idth: 100px`. Cette manœuvre avait au départ la bonne idée de n'être prise en compte que par Internet Explorer 6 et 7, ce qui pouvait arranger les concepteurs de pages en mode Quirks :

```
div {
  width: 80px;
  padding: 10px;
  w\idth: 100px; /* ignoré sauf sur IE6 et IE7 */
}
```

Certains hacks affichent des syntaxes tellement élaborées que l'on peut se poser des questions sur l'état psychologique de leur inventeur, telle la déclaration `/*/*//*/ color:green; /* */`, qui n'est reconnue que par Netscape 4 !

ALLER PLUS LOIN Le musée des hacks

Le site web Centricle recense tous les hacks trouvés par les concepteurs web ainsi que leur reconnaissance par les différentes versions de navigateurs :

- http://centricle.com/ref/css/filters/

Risques pour l'avenir

Un hack est non seulement un disgracieux détournement de syntaxes conformes, mais aussi et surtout, une source de conflit pour le futur. Nul ne peut deviner aujourd'hui l'avenir d'une syntaxe « mutante » : elle peut être corrigée par le navigateur déficient ou, pire, être intégrée aux spécifications et devenir parfaitement conforme et reconnue par tous.

Par exemple, le *simplified box model hack* évoqué précédemment est aujourd'hui pris en considération par un large ensemble de navigateurs contemporains tels que Firefox, Chrome, Opera et IE8. Cela signifie que les syntaxes telles que `w\idth: 100px` ne sont plus du tout ignorées par ces navigateurs et que tous les sites web ayant eu recours à cette ruse en espérant ne cibler que les anciennes versions d'Internet Explorer en font dorénavant les frais !

Hacks à méditer ?

Au sein de la jungle des hacks existants ou ayant existé, tous ne sont pas systématiquement bons à jeter. Il se peut que dans certains cas, très spécifiques et parfaitement délimités, l'usage de ces stratagèmes soit parfaitement légitime. Quoi qu'il en soit, réservez-les pour les situations extrêmes, quand toutes les autres possibilités ont dû être écartées et n'employez que des hacks considérés comme bénins.

Parmi ces entourloupes jugées moins risquées, certaines s'appliquent à des propriétés et d'autres à des sélecteurs. Selon vos besoins, l'une ou l'autre pourrait s'avérer nécessaire.

Hacks de propriétés

Les deux syntaxes suivantes sont relativement stables, car elles ne sont actuellement prises en compte par aucun navigateur à l'exception de ceux mentionnés. Cependant, la probabilité pour que ces grammaires fassent un jour partie des spécifications CSS n'est pas nulle. À vous de prendre vos responsabilités si vous les employez :

- `_propriété` (exemple :`_display: inline`) – il s'agit du *Underscore hack*, aujourd'hui uniquement reconnu par Internet Explorer 6 (et précédents) ;
- `*propriété` (exemple : `*display: inline`) – nommé *Asterisk hack*, il n'est reconnu que par IE6 et IE7.

Une autre syntaxe, plus connue, est parfaitement exploitable si vous aviez à cibler Internet Explorer uniquement : le mot-clé `!important`. En effet, les versions 5 et 6 de ce navigateur interprètent assez mal cette fonctionnalité : lorsque deux propriétés se contredisent au sein d'un même bloc de déclaration, IE6 ne reconnaît pas celle munie du mot-clé `!important`.

```
div {
  width: 100px!important; /* pour tous les navigateurs */
  width: 80px; /* pour IE5 et IE6 */
}
```

Hacks de sélecteurs

En ce qui concerne les hacks de sélecteurs, un seul est digne d'intérêt : le *star HTML bug*, qui s'écrit sous la forme `* html sélecteur {…}`, uniquement reconnu par IE6, et qui se base sur un défaut de ce navigateur : il compte un élément invisible parent de `<html>` dans son modèle de rendu.

Cibler les navigateurs récents à l'aide de sélecteurs avancés

Si le terme de *hack* et son usage vous mettent mal à l'aise – et je vous comprends – vous serez sans doute plus séduit par une autre pratique bien plus consciencieuse : utiliser volontairement des sélecteurs dits « avancés » pour cibler uniquement les navigateurs récents.

Cibler à partir d'Internet Explorer 7

Pour illustrer ce concept, prenons une syntaxe telle que celle-ci :

```
html > body div {…}
```

Comme nous l'avons vu en début d'ouvrage, le symbole `>` désigne le sélecteur d'enfant. La règle s'applique par conséquent « aux éléments `<div>` descendants de `<body>`, lui-même enfant de `<html>` ». Cette lourdeur peut paraître inutile de prime abord, puisque `<html>` et `<body>` sont forcément ancêtres de tous les éléments du document.

L'astuce et l'intérêt de cette syntaxe demeurent dans le fait que ce sélecteur, parfaitement valide, n'est reconnu et appliqué qu'à partir d'Internet Explorer 7. Cela permet, dans un souci d'amélioration progressive, d'attribuer des styles qui ne seront pas pris en compte par IE6 et ses ancêtres.

Cibler à partir d'Internet Explorer 8

Le sélecteur CSS 3 `:lang()` a l'avantage d'être reconnu non seulement par l'ensemble des agents utilisateurs récents, mais également à partir d'Internet Explorer 8.

Il rend donc possible la détection de tous les navigateurs actuels en basant sa syntaxe sur un sélecteur qui débuterait par `:lang(fr)`. Cette astuce suppose bien entendu que l'attribut HTML `lang` soit appliqué à un élément de structure principal (généralement la balise `<html>`) et qu'il ait pour valeur `fr` !

Dans notre quête du positionnement idéal, cette méthode offre, par exemple, la possibilité d'employer le modèle de rendu CSS en tableau, tout en proposant une alternative aux anciens navigateurs :

```
#toto { /* Pour tout le monde, dont IE6 / IE7 */
  float: left;
  width: 300px;
}
:lang(fr) #toto {  /* Pour les navigateurs modernes et IE8 */
  display: table-cell;
  float: none;
  width: auto;
}
```

Une autre éventualité : positionner à l'aide de `display: inline-block` tout en proposant une équivalence pour IE6 et IE7 :

```
#toto { /* Pour IE6/IE7 et les anciens navigateurs */
  display: inline;
  zoom: 1;
}
:lang(fr) #toto {  /* Pour les navigateurs modernes et IE8 */
  display: inline-block;
}
```

Bien que séduisantes, ces techniques de sélecteurs avancés sont à méditer au cas par cas, car il pourrait être plus judicieux d'utiliser un commentaire conditionnel ciblant IE7 ou IE8 et inférieur (voir partie suivante).

Préférer les commentaires conditionnels

Tous les navigateurs, selon leur ancienneté ou leur degré de conformité aux standards, ne prennent pas en charge les propriétés CSS de la même manière. Certains présentent des erreurs d'interprétation, quelques-uns reconnaissent les dernières nouveautés en CSS 3, d'autres non.

Nous avons constaté que, au fur et à mesure de l'évolution du navigateur, celui-ci devient de plus en plus conforme aux standards et les anciens hacks utilisés deviennent obsolètes. Ainsi, depuis l'arrivée d'IE7, nombre de contournements appliqués pour IE6 sont devenus inopérants car corrigés. Les concepteurs web ont dû reprendre une par une toutes leurs ruses pour convenir à la nouvelle version.

Pour remédier à ce problème, il existe une solution plus robuste et plus pérenne : les commentaires conditionnels. Il s'agit d'un mécanisme propre à Internet Explorer Windows, né avec la version IE5, et qui permet d'inclure dans une page HTML, de manière valide, une portion de code qui ne sera lue et interprétée que par IE, ou par l'une ou l'autre de ses versions.

Microsoft lui-même conseille aux concepteurs web d'employer la méthode des commentaires conditionnels afin d'éviter les hacks : « Nous vous recommandons de bien vouloir modifier vos pages afin de ne plus utiliser les 'hacks' CSS. Ceux qui souhaitent cibler ou éviter Internet Explorer pourront dorénavant employer les commentaires conditionnels. » *(Marjus Mielke, Microsoft, octobre 2005.)*

Fonctionnement

Les commentaires conditionnels se présentent comme des instructions dotées d'une condition (`if`) et qui peuvent se placer à n'importe quel endroit du document (X)HTML.

La syntaxe la plus simple est la suivante.

Partie HTML

```
<!--[if IE]>
  ici votre code HTML réservé à IE
<![endif]-->
```

Usage pratique

L'un des avantages majeurs de ce mécanisme est de pouvoir cibler les versions d'Internet Explorer qui devront suivre les instructions contenues au sein des commentaires conditionnels.

Partie HTML

```
<!--[if IE 6]> pour IE6 uniquement <![endif]-->
<!--[if gt IE 6]> pour les versions supérieures à IE 6 <![endif]-->
<!--[if gte IE 6]> pour les versions supérieures ou égales à IE 6 <![endif]- >
<!--[if lt IE 8]> pour les versions inférieures à IE 8 <![endif]-->
<!--[if lte IE 8]> pour les versions inférieures ou égales à IE 8 <![endif]-->
```

Il est par ailleurs possible d'incorporer des opérateurs logiques tels que `&` (et) ou encore `|` (ou).

Partie HTML

```
<!--[if (gte IE 6)&(lte IE 8)]> entre les versions IE 6 et IE 8 uniquement <![endif]
-->
<!--[if (IE 6)|(IE 8)]> pour les versions IE 6 ou IE 8 uniquement <![endif]-->
```

Il est même envisageable de cibler les navigateurs qui ne sont *pas* Internet Explorer :

Partie HTML

```
<!--[if !IE]><!--> pour les navigateurs non IE <!--<![endif]-->
```

Notez que la syntaxe est assez alambiquée dans ce dernier cas de figure, pour des raisons d'échappement de caractères et de validation du code.

Voici par exemple comment afficher un message d'information aux utilisateurs d'anciennes versions de navigateur (figure 6-2).

Figure 6-2

Message personnalisé selon le navigateur

Partie HTML à placer en début de <body>

```
<!--[if lte IE 6]>
  <div id="alert-ie6"><strong>Attention ! </strong> Votre navigateur (Internet
  Explorer 6) présente de sérieuses lacunes en terme de sécurité et de performances,
  dues à son obsolescence (il date de 2001).<br />En conséquence, ce site sera
  consultable mais de manière moins optimale qu'avec un navigateur récent.</div>
<![endif]-->
```

Dans notre quête de compatibilité maximale sur le navigateur de Microsoft, nous allons principalement nous servir de ce mécanisme pour faire un lien vers une feuille de styles corrective dédiée. Cette seconde feuille CSS, chargée à la suite du fichier de styles principal, aura pour but d'écraser et de rectifier au cas par cas les règles générales mal reconnues par IE.

Partie HTML

```
<link rel="stylesheet" href="styles.css" type="text/css" />
<!--[if lt IE 8]>
<link rel="stylesheet" href="styles_ie.css" type="text/css" />
<![endif]-->
```

Cet usage des commentaires conditionnels est devenu très vite un standard de fait, pratiqué par une majorité de la communauté des concepteurs web. En effet, il présente un certain nombre d'avantages, dont celui de la pérennité et de la propreté, le code demeurant valide contrairement aux hacks.

Classe conditionnelle pour Internet Explorer

En 2008, le développeur américain Paul Irish s'est penché sur le mécanisme des commentaires conditionnels et en a listé quelques désagréments :

- Les commentaires conditionnels requièrent une ou plusieurs requêtes HTML, ce qui est néfaste en terme de performances.
- S'ils sont appelés dans l'élément `<head>`, le temps d'affichage de la page est soumis au chargement complet des styles afférents.
- Le fait de multiplier les feuilles de styles et les sélecteurs identiques dans plusieurs fichiers ne facilite guère la relecture et la correction des erreurs.

L'argument des baisses de performances est non négligeable sur les versions IE6 et IE7, car il s'agit justement de navigateurs vieillots nécessitant des optimisations constantes et nombreuses en vue de ne pas handicaper leurs temps de calcul et d'affichage.

La solution proposée par Paul Irish a été de créer un nom de classe spécifique à l'élément `<body>`, via un commentaire conditionnel et sans nécessiter d'appel vers une feuille de styles.

Voici la version courte de sa méthode de *classe conditionnelle* :

Partie HTML

```
<!--[if lte IE 7]> <body class="ie7"> <![endif]-->
<!--[if !IE]><!--> <body> <!--<![endif]-->
```

Le principe est simple : sur les versions inférieures ou égales à Internet Explorer 7, le corps de page du document s'écrira `<body class="ie7">` ; sur tous les autres navigateurs, il s'agira simplement de `<body>`. Il devient alors aisé d'appliquer une règle spécifique à IE6 et IE7 :

Partie CSS

```
#kiwi { /* Pour tout le monde */
  display: inline-block;
}
.ie7 #kiwi { /* Pour IE6 et IE7 */
  display: inline;
  zoom: 1;
}
```

Cette méthode – qui, contrairement à ce que l'on pourrait croire, est parfaitement valide – commence à faire son chemin parmi les développeurs web aguerris. Pour les autres, la syntaxe demeure un tantinet rebutante.

HasLayout chez Internet Explorer

Un mécanisme propriétaire

Le *HasLayout* est un ancien mécanisme complexe et interne d'Internet Explorer, apparu sur les versions IE5, IE6 et IE7 (ainsi que sur IE8 en « mode de compatibilité »). Il disparaît sur IE8 standard et IE9.

Pour faciliter et accélérer l'affichage des pages web, Microsoft a jugé utile de distinguer en amont certains éléments HTML destinés à être dimensionnés et positionnés. Ces éléments triés sur le volet sont automatiquement dotés d'un attribut JavaScript en lecture seule, indiquant s'ils bénéficient ou non de cette faculté de *Layout* (structure de page) : c'est le concept du *HasLayout* (littéralement, « qui possède le *Layout* »).

S'il améliore les performances de rendu sur Internet Explorer, cet obscur mécanisme est également – et malheureusement – lié à de nombreuses erreurs d'affichage sur ce navigateur et ses conséquences se font sentir dans nos mises en page web.

Avoir le Layout

Ce concept est inhérent au moteur de rendu d'Internet Explorer et n'existe pas sous forme de propriété ou règle CSS. Selon le navigateur de Microsoft, un élément possède le Layout (*HasLayout*) ou ne le possède pas. JavaScript permet de lire cette valeur booléenne, telle un interrupteur positionné sur *true* (vrai) ou *false* (faux).

Certains éléments sont dotés de Layout par défaut. Il s'agit de `<body>`, `<table>`, `<td>`, `<img>`, `<hr>`, `<input>`, `<select>`, `<textarea>`, `<button>`, `<object>` ou encore `<applet>`. Aucun des autres éléments HTML – et ils sont nombreux – ne dispose de cet interrupteur de Layout sur la valeur *true*.

Donner et ôter le Layout

Bien qu'il s'agisse d'une valeur JavaScript en lecture seule, Microsoft explique sur son site technique (msdn.microsoft.com), qu'il est possible de conférer le Layout aux éléments qui ne le possèdent pas au départ, dans le but de corriger de nombreuses incohérences sur Internet Explorer.

CSS et la propriété zoom

Contre toute attente, la seule méthode permettant d'attribuer le Layout à un élément HTML ne relève pas du langage JavaScript, mais de… CSS.

En appliquant n'importe laquelle des déclarations CSS suivantes sur un élément dénué de Layout par défaut, il acquiert ce fameux privilège :

- `heigth` ou `width` : avec une valeur autre que `auto` ;
- `min-width` ou `min-height` (IE7 minimum) : n'importe quelle valeur (même 0) ;
- `max-width` ou `max-height` (IE7 minimum) : n'importe quelle valeur ;
- `float: left` ou `float: right` ;
- `position: absolute` ;
- `display: inline-block` ;
- `overflow: hidden, scroll` ou `auto` (depuis IE7) ;
- `zoom: valeur` (propriétaire Microsoft).

La liste des propriétés réellement applicables sans dénaturer l'objet est bien plus maigre qu'on ne le croit : il est généralement hors de question de sortir l'élément du flux (`position`, `float`) ou de modifier ses dimensions (`width`, `height`, `max-width`…). La propriété `overflow` peut causer de gros soucis d'affichage en cas de débordements et `display: inline-block` modifie le rendu et n'est reconnue par IE que sur un élément de type `inline` au départ.

Toutes ces considérations prises en compte, il ne reste finalement plus qu'une seule syntaxe méconnue – et pour cause, elle ne figure pas dans les spécifications – la propriété `zoom`.

Inventée par Microsoft, la propriété `zoom` est l'ancêtre de la déclaration CSS 3 `transform: scale` produisant un effet de grossissement sur l'élément. Agissant sous forme de ratio, l'expression `zoom: 2` se lit comme un rapport de 2/1, soit une échelle correspondant au double de la taille initiale.

Pour conférer le Layout sans modifier la structure, la taille ou les dimensions de l'élément, la solution la plus salutaire sera de lui attribuer la déclaration `zoom: 1`.

Si vous tenez à conserver un fichier CSS validé par l'outil automatique du W3C, vous placerez cette syntaxe propriétaire au sein d'une feuille CSS séparée et appelée via un commentaire conditionnel.

> **ATTENTION Restrictions pour les éléments de type inline**
>
> Pour compliquer encore un peu la situation, sachez que les propriétés `width` et `height` ne confèrent pas le Layout aux éléments de type HTML `inline` ou dotés d'une déclaration `display: inline`.

Supprimer le Layout ?

On ne peut pas supprimer le Layout des éléments qui en sont dotés par défaut, cependant, rien ne nous empêche de l'annuler pour ceux qui l'ont hérité via CSS, en redéfinissant les valeurs par défaut des propriétés, par exemple `width` et `height` à la valeur `auto`, `zoom` à `normal`, `position` à `static` ou encore `float` à `none`.

Du Layout et des erreurs

Le mécanisme de *HasLayout* est complexe à assimiler, pas uniquement parce qu'il s'agit d'une « cuisine interne » à Microsoft, mais aussi et surtout en raison de ses effets pervers : dans certains cas de figure, avoir le Layout corrige instantanément un défaut d'affichage sur Internet Explorer ; dans d'autres, cela crée des problèmes de rendu. Tout dépend du contexte. Pire, le comportement peut survenir d'une page à l'autre d'un site, sans raison apparente.

Erreurs corrigées en donnant le Layout

Le nombre d'erreurs ou d'écarts vis-à-vis des spécifications CSS recensés pour Internet Explorer 6 est assez impressionnant (par chance, l'équipe de Microsoft en a résolu énormément dans la version 7). En conséquence, il est parfois difficile de détecter si une erreur est due à la présence ou à l'absence de Layout sur un élément.

Retenez toutefois un chiffre éloquent : au moins 50 % des dysfonctionnements visuels sur IE6 et IE7 sont à mettre au crédit du mécanisme *HasLayout*. Il peut s'agir de problèmes de rendu très variés et souvent surprenants : des décalages de flottants, des espaces au sein des listes, des disparitions d'arrière-plan voire d'éléments entiers, ainsi que des problèmes de positionnement en général.

Voici une illustration représentative de ce phénomène déroutant. L'exemple suivant décrit une liste de liens ayant bénéficié de la règle `display: block` (figure 6-3) :

Partie HTML

```
<ul>
  <li><a href="#">Accueil</a></li>
  <li><a href="#">Société</a></li>
  <li><a href="#">Contact</a></li>
</ul>
```

Partie CSS

```
ul {background: #CCCC66; }
li {
  list-style: none;
  background: #6B9A49;
}
li a {
  display: block;
  color: white;
}
```

Affichée sur Internet Explorer 6, cette page présente une particularité assez symptomatique : un espace libre correspondant à la hauteur d'un caractère apparaît sous chacun des liens !

Figure 6-3

Erreur dans le rendu d'une liste de liens sous IE6

Une erreur d'affichage de ce type pour le moins inattendu (et heureusement corrigé sur IE7) laisse penser qu'il peut s'agir d'un problème de *HasLayout*.

En effet, en conférant le Layout à l'élément `<a>`, le problème de rendu disparaît instantanément (figure 6-4) :

```
li a {
  display: block;
  zoom: 1;
}
```

Figure 6-4

Erreur corrigée en attribuant le Layout

Erreurs dues au Layout

Les exemples d'altérations visuelles dues à l'absence de Layout sont nombreux et nous serions presque tentés d'appliquer la solution radicale qui est de conférer cet attribut à l'ensemble des éléments de la page, ainsi :

```
* {zoom: 1;}
```

Malheureusement, bénéficier du Layout est loin d'être la solution à toutes les incohérences d'affichage sur Internet Explorer. C'est même souvent le contraire : *HasLayout* confère une sorte de « toute-puissance » aux éléments, qui n'en font qu'à leur tête.

Parmi les cas d'erreurs dues à la présence du Layout, citons :

- Les éléments dimensionnés s'élargissent à tort, pour s'adapter à la largeur ou la hauteur de leurs contenus.
- La fusion de marges n'opère plus sur les éléments dotés de Layout.
- Les éléments hors flux occupent toute la largeur du référent alors qu'il devraient se restreindre à la largeur de leur contenu.
- Les éléments flottants ne dépassent plus de leur parent si ce dernier possède le Layout.

Voici une situation où la présence de Layout engendre un rendu non conforme aux spécifications. Dans l'exemple qui suit, notre structure comporte un paragraphe `<p>` au sein d'un bloc `<div>` positionné en absolu :

Partie HTML

```
<div>
  <p>Un kiwi</p>
</div>
```

Partie CSS

```
div, p {margin: 0;}
div {
  position: absolute;
  background: green;
}
p {color: white;}
```

Tel que les spécifications le préconisent, le `<div>` retiré du flux n'occupe pour largeur que la taille de son contenu, c'est-à-dire le texte « Un kiwi » (figure 6-5).

Figure 6-5

Pas d'erreur sans Layout

Toutefois, le paragraphe enfant de `<div>` qui se verrait octroyer le Layout, par exemple avec la règle `p {zoom: 1;}`, hérite, sur IE6 exclusivement, d'un pouvoir tel qu'il pousse son parent de manière à ce que l'ensemble occupe toute la largeur (figure 6-6).

Figure 6-6

Erreur due au Layout

L'inventaire des problèmes liés à la présence ou l'absence de Layout est si vaste qu'il est généralement nécessaire de traiter chaque situation au cas par cas. Sachez toutefois que si vous êtes confrontés à un dysfonctionnement caractéristique des versions 6 et 7 d'Internet Explorer, il y a de fortes probabilités qu'il s'agisse d'un problème de *HasLayout*. Et vous avez à présent les cartes en main pour le contrer.

Petite méthodologie de résolution d'erreurs

En théorie, résoudre un problème d'affichage se passe de la manière suivante : on commence par cibler et isoler l'élément problématique, on établit un diagnostic, puis on corrige les défauts. En pratique, c'est souvent un peu plus acrobatique et périlleux, d'autant plus qu'une erreur se présente rarement seule. Voyons comment procéder de la façon la plus méthodique possible.

Isoler l'élément

Un problème d'affichage est dû en général à des incompatibilités de navigateurs ou à des marges par défaut différentes selon les navigateurs. La première étape à suivre consiste donc à isoler l'élément qui ne se comporte pas comme vous l'aviez prévu.

Appliquer une couleur de fond

Actuellement, les sites web sont presque tous structurés à l'aide des balises `<div>`, `<table>`, `<hn>`, `<p>` et `<ul>`.

Les CSS offrent un moyen très simple d'isoler ces différents éléments : en leur attribuant une couleur de fond, vous bénéficierez immédiatement d'un visuel global de la structure de votre mise en page, sans toucher au contenu ni au code HTML.

En commençant votre feuille de styles avec tout ou partie des déclarations ci-après, vous aurez un aperçu de l'espace exact occupé par chacun des conteneurs. Les tables et cellules seront en gris, les `<div>` en beige, les éléments de titre ou de paragraphes auront une couleur kaki, etc. Ceci permettra très rapidement de mettre le doigt sur des décalages ou des erreurs d'affichage concernant l'un ou l'autre de ces éléments.

Déclarations CSS pour appliquer une couleur de fond

```
table {background-color: lightgray}
td, th {background-color: gray}
div {background-color: beige}
form {background: bisque}
input, select, textarea {background-color: salmon}
h1, h2, h3, h4, h5, h6, p {background-color: darkkhaki}
ul {background-color: lightsteelblue}
li {background-color: silver}
etc.
```

Cette étape permet également de localiser les éventuelles occurrences de fusion de marges qui peuvent occasionner certains décalages verticaux entre les blocs possédant des marges externes.

MEA CULPA Et si on ajoutait aussi une bordure ?

Dans mon livre précédent, *CSS 2 : pratique du design web*, j'évoquais également l'alternative consistant à entourer les éléments sus-cités par une bordure colorée d'un pixel. J'évite dorénavant cette démarche pour une raison simple : une bordure, même de 1 px, va s'ajouter dans le calcul de la largeur et de la hauteur de l'élément, ce qui aura pour conséquence de créer des décalages d'affichage en plus des erreurs que nous avions à corriger !

Si cette technique de couleur d'arrière-plan ne porte pas ses fruits, tentez alors de masquer les éléments un par un (soit en CSS, à l'aide d'une règle telle que `élément1, élément2 {display: none;}`, soit en HTML, en coupant temporairement des sections du code).

Adopter Firebug

Certains outils sont indispensables dans la panoplie du parfait petit intégrateur web, notamment pour cibler et corriger les différences d'affichage entre les navigateurs. Au sein de la longue liste de logiciels ou d'extensions utiles, l'un d'entre eux est souvent évoqué comme étant irremplaçable au sein de notre communauté : Firebug.

Cette extension est disponible sur plusieurs navigateurs dont Firefox et Chrome. Elle permet en un clic d'avoir une vision exhaustive de l'ensemble des propriétés appliquées – ou non – à un élément, de visualiser sa boîte et les différentes composantes (`width`, `padding`, `margin`, `border`), mais aussi et surtout d'agir sur toutes les propriétés et valeurs : modifier une valeur ou une propriété, annuler une propriété, ajouter une déclaration, etc.

Avec un minimum de pratique, Firebug deviendra très vite votre outil préféré et vous fera économiser de longues heures de tests et de débogage.

> **OUTILS Et si vous n'utilisez pas Chrome ou Firefox ?**
>
> Firebug est un outil de débogage extraordinaire, mais il ne peut pas être installé partout. Sachez cependant que d'autres outils natifs existent sur tous les navigateurs : les inspecteurs d'éléments sur Internet Explorer 8, Chrome et Safari ou encore l'excellent Dragonfly sur Opera.

S'informer

Comme tout bon professionnel ou passionné que vous êtes, je vous conseille vivement de pratiquer une veille technologique constante sur les méthodes et découvertes dans le monde de la conception web.

Cet ouvrage ne peut que vous enseigner les bases de certaines techniques, ainsi que des pistes à suivre, mais sa portée demeure limitée. Vous pensez tout savoir sur les CSS ? Détrompez-vous. Vous croyez maîtriser le positionnement flottant ? Très bien. Et si je vous demandais ce que vous évoquent les célèbres erreurs de flottements recensées sur le site PositionIsEverything.net : *Float model*, *Double margin*, *Escaping floats*, *Peek-a-boo*, *Guillotine*, *Three pixel jog*, *Multiple opposing floats* (Opera) ou encore les quelques défauts sur IE5 Mac ?

Pas évident, n'est-ce pas ?

Voilà pourquoi il est important de toujours s'informer auprès de sites dédiés ou de forums de discussion spécialisés (tels que forum.alsacreations.com) et de suivre les informations quotidiennement via son agrégateur RSS ou un compte Twitter (voir la partie Ressources en annexe). Le Web est vaste, vous y trouverez sans aucun doute la source et le moyen de contrer votre erreur d'affichage.

Corriger l'erreur

À présent que l'élément est isolé, plusieurs outils faciliteront le diagnostic et le dépannage : la validation des fichiers, l'application d'un Reset CSS temporaire, la prise en compte des priorités des sélecteurs, ou encore l'attribution du Layout.

Valider ses fichiers

Nous accomplissons souvent la validation des pages par habitude, ou parce qu'elle est exigée dans le cahier des charges du client. Et pourtant c'est une première étape absolument nécessaire pour au moins deux raisons essentielles :

- Sans Doctype, si le Doctype est tronqué ou si des caractères apparaissent avant le Doctype, Internet Explorer passe en mode Quirks et dans son modèle de boîte erroné.
- Le Validateur HTML et le Validateur CSS indiquent que des éléments sont manquants, ou si le concepteur a oublié de fermer une balise. Internet Explorer est souvent plus restrictif que d'autres navigateurs sur ce point et affichera différemment vos pages.

Souvent, des erreurs proviennent de la « grammaire » et de la conception même du document. Soumettre sa page aux différents validateurs est une précaution initiale précieuse pour détecter ces problèmes. Passez vos documents aux validateurs automatiques (X)HTML et CSS sur le site du W3C pour effectuer votre premier diagnostic.

Reset CSS temporaire

Tous les éléments de type bloc (sauf `<div>`) possèdent des marges internes (`padding`) et externes (`margin`) par défaut. L'inconvénient est que cette valeur par défaut varie d'un navigateur à l'autre et engendre des rendus différents. Il s'agit certainement de l'explication la plus courante lorsque des décalages apparaissent entre les diverses plates-formes.

Le meilleur moyen d'identifier un problème de marges sur certains éléments est de... supprimer toutes les marges de tous les éléments. Le principe est d'utiliser le sélecteur universel (*), qui s'appliquera à toutes les balises, et de mettre les marges à zéro :

```
* {margin: 0; padding: 0;}
```

Commencez votre feuille de styles par cette déclaration. Si les décalages involontaires disparaissent, vous aurez détecté un problème de marges par défaut. Il vous appartient ensuite d'isoler l'élément qui provoque ce décalage, puis de supprimer cette règle de mise à zéro globale temporaire.

Attention à la priorité des sélecteurs

Dans un chapitre précédent traitant des bonnes pratiques CSS 2.1, j'avais mis l'accent sur la pondération accordée aux différents types de sélecteurs. Ainsi, les sélecteurs d'identifiants seront toujours prioritaires sur les sélecteurs de classes, eux-mêmes privilégiés face aux sélecteurs d'éléments.

Cette notion de priorité explique pourquoi les propriétés que vous tentez d'appliquer à un élément ne sont pas prises en compte : vous avez précédemment ciblé ce même élément à l'aide d'un sélecteur prioritaire.

Si tel est le cas, Firebug vous le signalera. Ajoutez du poids à votre sélecteur trop faible, ou employez le mot-clé `!important` (avec modération) pour donner des priorités à vos règles.

Donner le Layout

Si les étapes précédentes demeurent insuffisantes et si le problème de rendu n'est pas corrigé sur Internet Explorer, pensez au mécanisme *HasLayout*.

Commencez par vérifier si l'élément fautif possède le Layout. Si tel est le cas, tentez de l'enlever en annulant les propriétés CSS spécifiques au Layout et observez le résultat. Dans le cas contraire, tentez de conférer le Layout à l'élément qui pose problème, ou à son parent.

Dernier recours : le hack ou commentaire conditionnel

En dernier recours, et si l'erreur d'affichage ne se résout pas en attribuant le Layout aux éléments, il ne vous reste qu'une seule alternative : modifier les propriétés ou les valeurs, voire modifier les schémas de positionnement uniquement sur Internet Explorer, soit à l'aide de hacks, soit via une feuille de styles appelée par un commentaire conditionnel. Dans ce fichier de styles, vous placerez vos règles correctives pour le navigateur de Microsoft.

Deuxième partie

HTML 5 et CSS 3 : l'innovation en marche

7

La révélation HTML 5

Faisons une petite incursion de l'autre côté du miroir de CSS, dans le langage de structure HTML. L'opus numéro 5 nous promet une nouvelle grammaire, une nouvelle syntaxe, de nouveaux éléments de sémantique et de périphériques, de nouveaux formulaires ainsi que de nouvelles applications dynamiques.

Bien que l'objet de cet ouvrage ne soit pas de couvrir le langage HTML au travers de ses différentes versions, il paraît difficilement envisageable d'aborder aujourd'hui un livre sur les spécifications CSS sans y consacrer un chapitre sur HTML 5, tant sa genèse et son évolution sont étroitement liées à celles de CSS 3.

Pourquoi HTML 5 ?

HTML 5, encore à l'état de brouillon – est-il utile de le préciser ? –, est une évolution logique du langage HTML, reposant sur les technologies contemporaines éprouvées (HTML, XHTML, JavaScript, géolocalisation) ou plus nébuleuses telles que le « Web 2.0 » ou encore la gestion des supports audio et vidéo. Le tout se construit sur les vestiges de l'antique mise en page via tableaux et au travers d'une période trouble, durant laquelle le Consortium W3C tentait de faire avancer d'imposants travaux sur des langages aussi déterminants que HTML et XHTML.

Le chantier HTML 5 a commencé fin 2003 grâce à un groupe de travail indépendant, mais ce n'est qu'en 2007 que le W3C officialise véritablement ce langage en intégrant en son sein ce groupe de travail. À partir de là s'est fait un gros effort afin de permettre à HTML 5 d'être rétrocompatible avec ses ancêtres, ce qui a quelque peu ralenti son développement.

HTML 5 offre nativement une large panoplie de nouveautés et de technologies :

- une grammaire entièrement revue et simplifiée ;
- de nouveaux éléments sémantiques et de nouveaux attributs ;
- la reconnaissance de vidéos ou de sons sans plug-ins ;
- une gestion étendue des formulaires ;
- la possibilité de se localiser géographiquement et de profiter de cette information ;
- la création de dessins ou de graphiques à l'aide de l'élément `<canvas>` ;
- le stockage de données sur son ordinateur pour les exploiter hors ligne ;
- la possibilité d'intervenir sur les éléments en les modifiant à la volée ou en les déplaçant (*drag and drop*), etc.

Je ne vais volontairement pas entrer dans les détails de la spécification HTML 5, complexe et encore instable pour certains de ses modules, car je m'éloignerais de l'objectif que je me suis fixé pour cet ouvrage et parce que d'autres le font bien mieux que moi (vous trouverez des ressources sur ce sujet en fin de livre). Toutefois, ce langage mérite bien un chapitre.

Une nouvelle grammaire

HTML 5 se démarque radicalement des éditions précédentes de par sa grammaire et sa syntaxe, étudiées pour faciliter l'apprentissage et la compréhension par les concepteurs web.

Un Doctype simplifié

Elle vous est fidèle depuis que vos documents web sont conformes au W3C. Toujours en tête de vos pages, la *Déclaration de type de document* (DTD), appelée par le Doctype, annonce fièrement au navigateur quel est le langage et la grammaire que vous lui faites lire.

Voici, à titre indicatif, deux Doctype couramment employés de nos jours, le HTML 4.01 strict et le XHTML 1.0 transitionnel :

Doctype HTML 4.01 Strict

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
```

Doctype XHTML 1.0 transitionnel

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

À l'ère naissante de HTML 5, la syntaxe du Doctype a été remaniée de façon tellement optimisée qu'elle en devient largement plus facile à retenir. Jugez par vous-même :

Doctype HTML 5

```
<!DOCTYPE html>
```

Avouez que la différence est flagrante, n'est-ce pas ?

L'un des principaux avantages de cette écriture courte – outre celui de pouvoir se la rappeler – est de forcer tous les navigateurs à passer en mode de rendu Standard en lieu et place du mode Quirks (voir le chapitre 6 dédié aux résolutions d'erreurs), même s'ils ne reconnaissent pas encore la norme HTML 5 !

Une bonne nouvelle ne venant jamais seule, d'autres parties de la charpente HTML se trouvent allégées dans la version 5 du langage, à commencer par l'élément `<meta> charset` définissant l'encodage de caractères du document :

Meta charset HTML 4 et XHTML

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

Meta charset HTML 5

```
<meta charset="utf-8">
```

Une syntaxe permissive

HTML 5, en tant que digne successeur de HTML 4.01, conserve par défaut la permissivité syntaxique de son prédécesseur : majuscules autorisées pour les noms d'éléments et d'attributs, apostrophes simples ou doubles non obligatoires pour signaler les valeurs.

En outre, HTML 5 poursuit les libertés de son ancêtre. Ainsi, les éléments `<p>`, `<dd>`, `<dt>`, `<li>`, `<optgroup>`, `<option>`, `<td>`, `<th>`, `<tr>`, `<thead>` et `<tfoot>` ne nécessitent pas de balise fermante pour être valides. Seule la version XHTML 5 oblige à fermer ces éléments.

Plus fort encore, certains éléments ne nécessitent ni balise fermante ni balise ouvrante. C'est le cas de `<html>`, `<head>`, `<body>`, `<thead>`, `<tfoot>` et `<tbody>`. Cela signifierait que la présence même de ces éléments devient implicite.

Pour finir, notons que l'attribut `type` (rencontré principalement au sein des balises `<script>` et `<style>`) devient superflu.

Le code HTML suivant – parfaitement valide et suffisant – résume à lui seul la concision autorisée en HTML 5 :

```
<!doctype html>
<meta charset=UTF-8>
<title>titre de page</title>
<p>hello world !
```

Cependant, je vous invite à la plus grande des précautions quant au maniement de cette syntaxe très démunie. À moins de maîtriser parfaitement les arcanes du langage HTML 5 au point de ne plus commettre d'erreur ou d'oubli, pensez à conserver autant que possible la rigueur de XHTML au sein de vos documents.

De nouveaux éléments sémantiques

Au-delà de la rénovation syntaxique, HTML 5 introduit de nouveaux éléments inédits dotés de sens et apportant une alternative aux blocs génériques `<div>` et `<span>`. Ces derniers se déclinent dorénavant en un large panel d'éléments sémantiques tels que `<article>`, `<section>`, `<aside>`, `<hgroup>`, `<header>`, `<footer>`, `<nav>`, `<time>`, `<mark>`, `<figure>`, et `<figcaption>`.

Tous ces éléments peuvent être mis en forme nativement via CSS sur l'ensemble des navigateurs modernes, même si leur fonction propre ne sera véritablement reconnue que sur les toutes dernières générations. Internet Explorer se distingue une fois encore, puisqu'il est l'unique navigateur qui ne parvient pas à se représenter ces éléments. Une alternative en JavaScript, que nous allons découvrir, lui sera nécessaire.

En termes d'accessibilité et de référencement, l'emploi de ces nouveaux éléments ne bouleverse pas la donne à l'heure actuelle : les agents utilisateurs et moteurs ne les reconnaissant pas, ils les interprètent comme des éléments génériques (tels `<div>` et `<span>`). En revanche, dans un futur encore incertain, le rôle confié à ces éléments peut s'avérer important.

<header>

L'élément `<header>` représente le bloc d'en-tête d'une section ou d'une page. Il remplace avantageusement son homologue classique `<div id="header">`, mais ne doit pas forcément être considéré comme un élément unique dans le document : toute section est susceptible de disposer de son `<header>` :

```
<header>
    ici tout le contenu de l'en-tête de document ou de section
</header>
```

<footer>

L'élément `<footer>` regroupe les contenus du pied d'une section ou d'un document (pied de page) et est destiné à recueillir les informations concernant l'auteur, les mentions légales, etc. Tel `<header>`, l'élément `<footer>` peut apparaître à divers endroits du document :

```
<footer>
    ici le contenu du pied de page de document ou  de section
</footer>
```

<nav>

`<nav>` a pour fonction de regrouper les liens de navigation considérés comme majeurs ou jugés suffisamment pertinents. Ceux-ci peuvent être internes ou externes à la page :

```
<nav>
  <h2>Menu</h2>
  <ul>
```

```
    <li><a href="">Accueil</a></li>
    <li><a href="">Société</a></li>
    <li><a href="">Contact</a></li>
  </ul>
</nav>
```

<aside>

L'élément `<aside>` représente une portion de contenu contextuelle, directement ou indirectement liée aux éléments qui l'entourent, tel un bloc d'archives relatives au contenu précédent. Par extrapolation, cet élément désigne fréquemment les barres latérales classiques du document et peut remplacer l'ancien `<div id="sidebar">` :

```
<aside>
  <h1>Archives</h1>
  <ul>
    <li><a href="/archives/09/05/">Mai 2009</a></li>
    <li><a href="/archives /09/06/">Juin 2009</a></li>
    <li><a href="/archives /09/07/">Juillet 2009</a></li>
  </ul>
</aside>
```

<section>

Une `<section>` représente un bloc générique de contenu ayant la même thématique. Cela concerne les chapitres, les en-têtes et pieds de page, ou toute autre partie dans un document. L'élément `<section>` peut contenir des éléments de titre `<h1>` à `<h6>` pour une meilleure définition de la structure du document :

```
<section id="works">
  <h1>Nos travaux</h1>
  <p>bla bla production de kiwis bla les vendre bla bla</p>
</section>
```

<article>

L'élément `<article>` désigne une portion du document potentiellement autonome dans le sens où elle pourrait être reprise ou réutilisée, comme un article de journal, de blog ou de forum :

```
<article>
  <p><a href="http://www.alsacreations.com/actu/lire/746-xhtml-est-mort-vive-html.
  ➥ html">XHTML est mort, vive HTML ! </a><br>
  Sous ce titre quelque peu provocateur (et faux) se cache une réalité officielle
  depuis hier soir : le W3C vient d'annoncer que ses travaux sur XHTML 2 se
  termineront cette année 2009.</p>
</article>
```

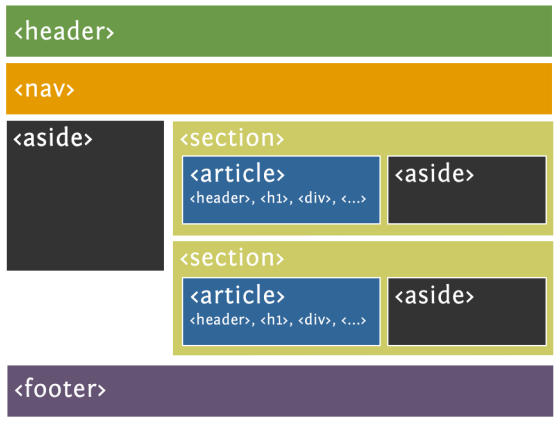
<figure>

Selon le W3C, l'élément `<figure>` s'emploie combiné à `<figcaption>` pour structurer des illustrations, photos, diagrammes et portions de codes, entre autres. `<figcaption>` désignera la légende de la figure :

```
<figure>
  <img src="/images/chaton.jpg" alt="Gribouille le matou">
  <figcaption>Gribouille, un petit chat mignon tout plein</figcaption>
</figure>
```

Figure 7-1

Exemple de structure des éléments en HTML 5



Exercice pratique : utiliser les nouveaux éléments

En production, l'usage des nouveaux éléments sémantiques HTML 5 est parfois rendu délicat pour deux raisons majeures :

- Les navigateurs nés avant 2010 ne prennent pas en compte la sémantique de ces éléments.
- Internet Explorer jusqu'à la version 8 ne les reconnaît pas et ne les affiche pas du tout.

Voyons dans le détail comment contourner ces obstacles et offrir du HTML 5 à l'ensemble de votre public… ou presque.

Pour commencer, identifions la liste suivante d'éléments HTML 5 que nous allons employer pour la structure d'une page :

- `<header>` : en-tête de la page ou d'une section ;
- `<nav>` : liens de navigation ;
- `<section>` : section de contenu ;
- `<aside>` : partie contextuelle latérale d'une section ;
- `<figure>` : pour regrouper des images et leur description ;
- `<footer>` : pied de page ou de section.

Les navigateurs n'affichent généralement pas ces éléments correctement (il seront identifiés comme étant de type « en-ligne » par défaut, ce que l'on peut comparer à un `<span>`), mais rien ne nous empêche de les utiliser en connaissance de cause en leur appliquant un `display` approprié (`display:block` ou autre).

En revanche, et cela devient plus problématique, Internet Explorer 8 et versions antérieures nécessitent un code JavaScript afin de créer au préalable ces éléments dans leur DOM, sous peine de ne rien afficher du tout !

Concrètement, il suffira d'appeler un script externe dès le début de la page de manière spécifique au navigateur de Microsoft :

Partie HTML

```
<!--[if lt IE9]>
  <script src="scripts/html5-ie.js"></script>
<![endif]-->
```

Partie JavaScript (fichier html5-ie.js)

```
document.createElement("header");
document.createElement("footer");
document.createElement("section");
document.createElement("aside");
document.createElement("nav");
document.createElement("article");
document.createElement("figure");
document.createElement("figcaption");
document.createElement("hgroup");
document.createElement("time");
…
```

Une autre alternative consiste à se référer au document JavaScript dédié et optimisé que l'on peut lier depuis les serveurs de Google :

Partie HTML

```
<!--[if lt IE9]>
  <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
```

Ces petites manipulations vous permettent de disposer sans grosses difficultés des principales balises de structures HTML 5, tout en garantissant qu'elles seront correctement interprétées par l'ensemble des navigateurs courants, y compris anciens.

Soyez prudents, cependant, car elles n'auront toutefois aucune valeur sémantique avant que les navigateurs et les moteurs de recherche ne les prennent en charge officiellement.

Redéfinition et obsolescence d'éléments

HTML 5 redéfinit le rôle de certains éléments comme `<i>`, `<small>` ou `<b>` qui, d'éléments purement visuels, deviennent ainsi des sections de texte lues ou interprétées différemment du texte « normal ». D'autres éléments tels que `<a>`, `<menu>`, `<em>`, `<strong>` ou encore `<hr>` voient leurs fonctions évoluer, ce qui implique certaines adaptations et un temps d'assimilation pour nous autres, concepteurs web.

Par ailleurs, un certain nombre d'éléments disparaissent de la spécification HTML 5. Il s'agit généralement des éléments déjà abandonnés par XHTML :

- `<frame>`, `<frameset>`, `<noframes>` (jugés néfastes à l'utilisabilité et à l'accessibilité) ;
- `<acronym>` disparaît au profit de `<abbr>` ;
- `accesskey` (dans les éléments `<a>`, `<area>`, `<button>`, `<input>`, `<label>`, `<legend>` et `<textarea>`) ;
- `longdesc` (dans `<img>` et `<iframe>`) ;
- `name` (dans `<img>`, `<form>` et `<a>`) ;
- `language` (dans `<script>`);
- `summary` (dans `<table>`) ;
- `<basefont>`, `<big>`, `<center>`, `<font>`, `<s>`, `<strike>`, `<tt>`, `<u>`, etc.

De nouveaux éléments de périphériques

Nous sommes entourés de technologies multimédias fantastiques, qui nous permettent d'écouter à tout instant une symphonie de Rachmaninov ou le dernier album de Francis Lalanne sur notre baladeur MP3, ou encore de visionner un film de Jean-Claude Van Damme sur notre mini-PC, voire notre téléphone !

Dans le monde du Web, malgré des capacités de stockage et d'échanges chaque jour améliorées, la situation semble bloquée et complexe : point de salut hors technologies propriétaires ou plug-ins tels que Adobe Flash ou Apple QuickTime.

HTML 5 change la donne en définissant trois nouveaux éléments destinés à gérer nativement les aspects visuels, sonores et graphiques de l'ère numérique moderne : `<audio>`, `<video>` et `<canvas>`.

<audio>

L'élément `<audio>` permet la lecture d'un fichier audio, aux formats classiques, sans recourir à des outils propriétaires (tableau 7-1 et figure 7-2). Sa syntaxe est aussi courte qu'efficace :

```
<audio src="la-danse-des-canards.mp3"></audio>
```

Quelques attributs peuvent se greffer à la propriété `<audio>`. Il s'agit, entre autres, de `autoplay` (lecture automatique) et `controls` (affichage des boutons de contrôle de lecture).

Voici comment proposer sur votre page web un morceau musical lancé dès le chargement et muni de sa barre de contrôle :

```
<audio id="player" src="la-moldau.mp3" autoplay controls></ audio>
```

COMPATIBILITÉ Encore Internet Explorer

Comme l'on peut s'y attendre, c'est du côté d'Internet Explorer que se créent les principales réticences : il faudra attendre la généralisation d'IE9 pour que la prise en charge de cette propriété soit effective sur le navigateur de Microsoft. Une alternative Flash ou QuickTime sera de mise pour une compatibilité maximale à l'heure actuelle.

Tableau 7-1 Reconnaissance des formats audio

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
MP3 (.mp3)		OK		OK	OK	
Ogg Vorbis (.ogg)			OK	OK		OK
Wave (.wav)		OK	OK		OK	OK

À l'instar de la propriété `font-family`, l'élément `<audio>` offre la possibilité de proposer plusieurs types de formats au navigateur ; celui-ci va parcourir chacun des fichiers proposés et retenir le premier reconnu, sans charger les autres, comme le montre cet exemple :

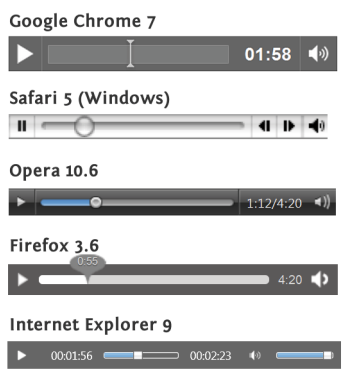
```
<audio>
  <source src="la-moldau.mp3">
  <source src="la-moldau.ogg">
  <!-- proposez ici une alternative en Flash ou QuickTime -->
</audio>
```

Notez que rien ne vous empêche de manipuler `<audio>`, comme les autres éléments de la page, via JavaScript ou jQuery, par exemple pour détecter si le son est coupé :

```
document.getElementById("player").muted = false;
```

Figure 7-2

L'élément audio sur différents navigateurs



<video>

HTML 5 introduit l'élément `<video>` pour insérer des séquences vidéo dans une page web, ce qui offre une alternative aux plug-ins propriétaires d'Apple (QuickTime) ou d'Adobe (Flash) pour les plates-formes ne le reconnaissant pas (iPhone, iPod, iPad...).

La syntaxe de base de l'élément vidéo est très simple :

```
<video src="une_video.mp4"></video>
```

Des attributs permettent d'apporter des informations complémentaires :

- `controls` affiche les possibilités de contrôle de la vidéo (avance, pause, stop…).
- `autoplay` lance la lecture automatiquement dès le chargement de la page.
- `preload` spécifie au navigateur de débuter le téléchargement de la vidéo tout de suite, en anticipant sur le fait que l'utilisateur lira la vidéo.

À l'heure actuelle, la difficulté d'implémentation et de standardisation de l'élément `<video>` tient au fait qu'un certain nombre de formats se livrent une bataille rangée (tableau 7-2 et figure 7-3) :

- **H.264 (.mp4)** : H.264 est proposé par le Moving Picture Experts Group. C'est un format non libre (soumis à brevets) et non gratuit. Toutefois, il est gratuit dans certaines utilisations (la diffusion gratuite de vidéos par des sites web, par exemple). Ce format est reconnu par les navigateurs Apple (Safari, Safari Mobile) ainsi que sur Internet Explorer 9 et les versions de Google Chrome antérieures à 10.
- **Ogg Theora (.ogv)** : Theora est un format de compression vidéo open source, sans brevets. Ceci donne le droit à tous d'utiliser Theora (à des fins non commerciales tout comme à des fins commerciales) sans devoir payer de redevance au consortium MPEG. Ce format est pris en charge par Firefox 3.6, Opera, et Google Chrome.
- **WebM/VP8 (.webm)** : WebM est un format multimédia ouvert qui a été lancé par Google (après rachat de la société On2 Technologies). L'utilisation en est libre et gratuite. Ce format est lisible sur Firefox 4, Opera, Google Chrome et Internet Explorer 9. Pour ce dernier, il est nécessaire d'installer le codec dans Windows. Sinon, il lira directement le mp4 (installé par défaut dans Windows 7).

Tableau 7-2 Reconnaissance des formats vidéo

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
H.264 (.mp4)		OK		OK*	OK	OK
Ogg Theora (.ogv)			OK	OK		OK
WebM (.webm)		OK	OK	OK		OK

* Google a annoncé en janvier 2011 qu'il cesserait de prendre en charge le codec H.264 au courant des prochaines versions de Chrome (à partir de Chrome 10 ?).

Afin de faciliter sa reconnaissance par différents agents utilisateurs, l'élément `<video>` prévoit de définir plusieurs types de formats de fichiers :

```
<video>
  <source src="une_video.mp4" type="video/mp4">
  <source src="une_video.webm" type="video/webm">
  <source src="une_video.ogv" type="video/ogg">
</video>
```

COMPATIBILITÉ L'ordre est important pour Apple

Les terminaux mobiles d'Apple (iPhone, iPad et iPod) ne reconnaissent actuellement que la version H.264 (.mp4), et uniquement si la source du fichier est proposée en premier dans la liste.

Figure 7-3

L'élément vidéo sur différents navigateurs

Pour les navigateurs qui ne tiennent pas encore compte de l'élément `<video>`, à savoir principalement Internet Explorer 8 et ses versions antérieures, une alternative consiste à inclure un objet Flash au sein de la liste :

```
<video>
  <source src="une_video.mp4" type="video/mp4">
  <source src="une_video.webm" type="video/webm">
  <source src="une_video.ogv" type="video/ogg">
  <object type="application/x-shockwave-flash" data="une_video.swf">
    <param name="movie" value="une_video.swf" />
```

```
    <param name="wmode" value="transparent" />
    <p>Image ou texte alternatif</p>
  </object>
</video>
```

OUTIL Conversion de fichier

Le nombre de formats de fichiers vidéo à proposer pour être sûr de plaire à l'ensemble des navigateurs est un réel handicap à la démocratisation de l'élément `<video>` HTML 5. Toutefois, il existe des logiciels de conversion multiformats tels que Miro Video Converter, qui est libre, gratuit et performant, puisqu'il prend en charge les fichiers mp4, WebM (vp8) et Ogg Theora.

▸ http://www.mirovideoconverter.com

<canvas>

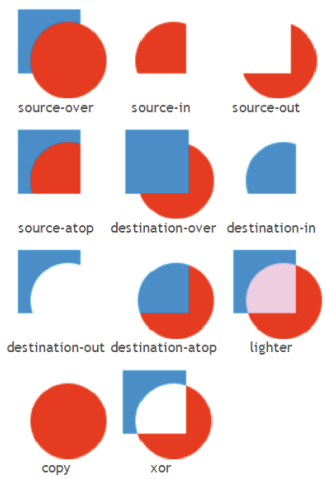
L'élément HTML 5 `<canvas>` représente une zone de dessin bitmap (« canevas ») au sein de la page, pilotée par JavaScript et capable de restituer des formes, des diagrammes et graphiques, des animations ou autres applications visuelles (figure 7-4).

Les fonctionnalités disponibles de base sont plutôt simplistes, mais nombreuses :

- des outils de dessins et de formes (rectangles, ellipses, arcs, chemins, lignes et courbes de Bézier) ;
- des effets (remplissages, contours, ombres, dégradés linéaires et radiaux, transparences, découpages) ;
- des transformations (zooms, rotations, translations) ;
- la gestion des images (import et export) ;
- la gestion de textes et de la typographie (polices embarquées, contours, étirements).

Figure 7-4

Illustration de l'élément canvas (source : blog.nihilogic.dk)



COMPATIBILITÉ Élément <canvas>

L'élément `<canvas>` est reconnu par un large panel de navigateurs : Firefox 3, Chrome 3, Safari 3, Opera 10, Phone 1 et Android 1. Seul Internet Explorer est à la traîne, mais la bibliothèque propriétaire ExplorerCanvas, détaillée dans le prochain aparté (en fin de section), corrige cette lacune.

Un canevas nécessite deux entités indissociables : l'élément `<canvas>` dans le document HTML et un code JavaScript pour dessiner dans la surface occupée par `<canvas>`.

Voici un exemple d'usage de la balise `<canvas>` (figure 7-5) :

Partie HTML

```
<canvas id="dessin" width="500" height="200"></canvas>
```

À ce stade, l'élément est invisible mais rien ne vous empêche de commencer à le styler en CSS pour le distinguer :

Partie CSS

```
canvas {outline: 1px solid green}
```

Passons à présent la main à la couche JavaScript et dessinons un rectangle sur une surface de 200 pixels de large pour 100 pixels de haut :

Partie JavaScript

```
<script>
  var draw = document.getElementById("dessin");
  var context = draw.getContext("2d");
  context.fillRect(0, 0, 200, 100);
</script>
```

La première ligne de code est commune. Elle permet à JavaScript de détecter l'élément « rectangle » dans le DOM.

La deuxième ligne, obligatoire également, indique, via la méthode `getContext`, que notre canevas est de type « 2d » et non en trois dimensions.

Enfin, la troisième ligne de code applique une couleur de remplissage (noire par défaut) à un rectangle positionné en haut à gauche et occupant 200 × 100 pixels.

Figure 7-5

Rectangle noir

La méthode `fillStyle` définit la couleur de fond, que nous allons changer pour du bleu (figure 7-6) :

```
<script>
  var draw = document.getElementById("dessin");
  var context = draw.getContext("2d");
  context.fillStyle = "#0000ff";
  context.fillRect(0, 0, 200, 100);
</script>
```

Figure 7-6

Rectangle bleu

La méthode `arc` construit une surface arrondie. Ici, nous allons placer son centre à 50 px du haut et 50 px de la gauche du canevas, puis lui indiquer un rayon de 100 pixels (figure 7-7) :

```
<script>
  var draw = document.getElementById("dessin");
  var context = draw.getContext("2d");
  context.fillStyle = "#0000ff";
  context.arc(0,0,200,0,Math.PI*2,true);
  context.fill();
</script>
```

Figure 7-7

Cercle bleu

Les ombrages peuvent être appliqués via les méthodes `shadowBlur` et `shadowColor` (figure 7-8) :

```
<script>
  var draw = document.getElementById("dessin");
  var context = draw.getContext("2d");
  context.fillStyle = "#0000ff";
  context.arc(0,0,200,0,Math.PI*2,true);
  context.shadowBlur = 50;
```

```
  context.shadowColor = "black";
  context.fill();
</script>
```

Figure 7-8

Cercle bleu ombragé

Les possibilités et fonctionnalités offertes par l'élément `<canvas>` sont immenses et je vais devoir conclure par un dernier exercice qui se contentera d'entourer le cercle d'une bordure blanche pleine de 12 pixels (figure 7-9) :

```
<script>
  var draw = document.getElementById("dessin");
  var context = draw.getContext("2d");
  context.fillStyle = "#0000ff";
  context.arc(0,0,200,0,Math.PI*2,true);
  context.shadowBlur = 50;
  context.shadowColor = "black";
  context.fill();
  context.lineWidth = 12;
  context.strokeStyle = "white";
  context.stroke();
</script>
```

Figure 7-9

Cercle bleu ombragé et entouré

COMPATIBILITÉ Alternative pour Internet Explorer

Microsoft reconnaît `<canvas>` sur la version Internet Explorer 9 actuellement en bêta. En attendant sa diffusion, l'alternative JavaScript ExplorerCanvas permet de simuler cet élément HTML 5.

▸ http://code.google.com/p/explorercanvas/

Il suffit en pratique d'incorporer le script à l'aide d'un commentaire conditionnel placé dans le `<head>` du document :

```
<!--[if lt IE9]><script src="excanvas.js"></script><![endif]-->
```

et de ne plus se soucier d'Internet Explorer !

Si les applications du très prometteur élément HTML 5 `<canvas>` vous passionnent, je vous invite à dévorer la page du centre de développement de Mozilla qui lui est consacrée, ou encore le site CanvasDemos, qui est intégralement consacré aux applications, outils et jeux réalisés à l'aide de l'élément `<canvas>`.

- https://developer.mozilla.org/fr/HTML/Canvas
- http://www.canvasdemos.com

PRATIQUE Un pense-bête gratuit à télécharger

Le développeur Jacob Seidelin a concocté une feuille de référence récapitulative (*cheat sheet*) comprenant l'ensemble des méthodes, attributs et valeurs relatives à l'élément `<canvas>`. Ce pense-bête est téléchargeable librement en version PDF ou PNG à l'adresse :

- http://blog.nihilogic.dk/2009/02/html5-canvas-cheat-sheet.html

Une nouvelle génération de formulaires

Dans le grand chantier de HTML 5, une place d'honneur est réservée aux éléments interactifs et aux formulaires. Plutôt rustiques jusque-là, les éléments de saisie de la dernière génération se voient dotés de nouveaux types de champs, de nouvelles fonctions, voire de tests d'expressions régulières sur leur contenu textuel. Je vous en présente les plus prometteurs.

email, url, tel, number, color...

HTML 5 introduit de nouveaux types de champs de formulaires. Parmi ceux-ci :

- `email` : le champ requiert un contenu au format d'adresse électronique.
- `url` : le champ accueille des URL absolues.
- `tel` : le champ est destiné aux numéros de téléphone.
- `number` : le champ accepte uniquement les caractères numériques.
- `color` : le champ est prévu pour les chaînes représentant une valeur de couleur.

Ces différents types viennent s'ajouter aux valeurs classiques de HTML 4.01, à savoir `submit`, `image`, `text`, `radio`, `button`, `checkbox`, `hidden`, `file` et `password`.

La syntaxe est similaire pour chacune des valeurs. Voici comment symboliser un champ d'adresse électronique (figure 7-10) :

```
<input type="email" name="email">
```

Figure 7-10

Un champ de type e-mail sur iPhone

Même si les fonctionnalités de ces éléments demeurent encore assez timides, l'information apportée par le type offre la possibilité d'appliquer des mises en forme spécifiques. Vous pouvez par exemple agrémenter les champs d'adresse électronique avec un pictogramme caractéristique en ciblant le sélecteur d'attribut :

```
[type="email"] {background: url(email.png) left center no-repeat;}
```

COMPATIBILITÉ Éléments de formulaires

Ces nouveaux éléments de formulaires sont généralement compatibles avec les navigateurs récents hors Internet Explorer, c'est-à-dire Chrome 5, Safari 5, Opera 10, iPhone et Firefox 4, à l'exception de `tel` qui n'est pas reconnu par Opera 10.6 et `color` qui n'est en fait compris par aucun agent utilisateur actuel.

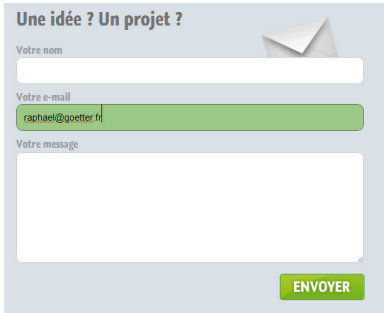
Le manque de prise en charge de ces fonctions n'est en aucun cas problématique puisque les valeurs ignorées sont automatiquement et simplement considérées comme de type neutre `text`. Vous ne risquez donc rien à les employer dès maintenant !

Chacun des types de champ obéit à une norme qui définit les valeurs acceptées ou rejetées. Ainsi, il est possible de vérifier quels éléments sont invalides à l'aide des sélecteurs CSS 3 `:valid` et `:invalid` (figure 7-11) :

```
input[type="email"]:valid {background: green;}
```

Figure 7-11

Une adresse électronique valide sur Google Chrome



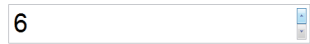
Au cas par cas, certains attributs complémentaires peuvent élargir les fonctionnalités des nouveaux éléments de formulaire. Le type `number` accepte par exemple les attributs `min`, `max`, `step` et `value` (figure 7-12) :

```
<input type="number" min="0" max="10" step="2" value="6">
```

- `type="number"` indique qu'il s'agit d'un champ de type numérique.
- `min="0"` spécifie la valeur minimale acceptable pour ce champ.
- `max="10"` est le maximum acceptable.
- `step="2"`, combiné à la valeur minimale indique que seules les valeurs 0, 2, 4, 6, 8 et 10 sont acceptées, par « pas » de 2.
- `value="6"` est la valeur par défaut au chargement de la page.

Figure 7-12

Un champ numérique



range

Les champs de type `range` se présentent sous la forme d'un curseur défilant, ou *slider* (figure 7-13) :

```
<input type="range" min="0" max="50" value="0">
```

À l'instar du type `number`, un champ de type `range` accepte les propriétés complémentaires `min`, `max`, `step` et `value` :

```
<input type="range" min="0" max="10" step="2" value="6">
```

Figure 7-13

Illustration de range

date, datetime, month, week, time

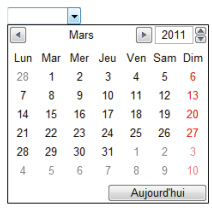
Un certain nombre de nouveaux types de champs concernent les dates et l'heure :

- `<input type="date">` affiche un contrôle de date (année, mois, jour).
- `<input type="datetime">` affiche un contrôle de date associé à une heure.
- `<input type="month">` affiche un contrôle de mois.
- `<input type="week">` affiche un contrôle de numéro de semaine.
- `<input type="time">` affiche un contrôle d'heure.

> **COMPATIBILITÉ Éléments de date et heure**
> Les différents éléments de date ne sont actuellement reconnus que par Opera depuis sa version 9 (figure 7-14), qui propose par défaut un contrôle sous forme de *datepicker* (sélecteur de date).

Figure 7-14

Un « datepicker » sur Opera



search

Le type de champ `search`, comme son nom l'indique, désigne une zone de recherche au sein du formulaire (figure 7-15) :

```
<input type="search" name="recherche">
```

Les agents utilisateurs actuels n'offrent pas encore de véritable prise en charge fonctionnelle de ce type de champ, mais rien ne nous empêche de profiter d'ores et déjà de cette information pour les mettre en forme :

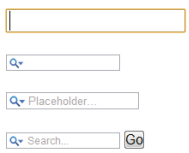
```
[type="search"] {border: 1px solid orange}
```

L'attribut complémentaire `results` affiche les dernières recherches récentes :

```
<input type="search" results="10" name="recherche">
```

Figure 7-15

Illustration de search



placeholder

`placeholder` est un attribut applicable à tout élément de formulaire. Sa fonction est d'afficher une chaîne de caractères temporaire au sein du champ. Lors du clic ou de l'activation du champ à l'aide du clavier, ce texte disparaît pour laisser la place au contenu à saisir.

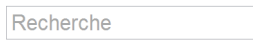
Idéal par exemple pour indiquer qu'un champ est obligatoire ou pour remplir le rôle d'étiquette, `placeholder` s'applique à tous les champs de formulaires (`<input>`), même sur les éléments de type `password`, ce qui n'était pas réalisable précédemment, puisque toute valeur dans un tel champ s'affichait avec des caractères masqués.

Le voici en œuvre (figure 7-16) :

```
<input type="search" name="recherche" placeholder="Recherche">
```

Figure 7-16

Illustration de placeholder



COMPATIBILITÉ placeholder

Parfaitement reconnu par Chrome et Safari depuis leurs versions 5, `placeholder` n'est cependant compris ni par Opera 10.6, ni par IE9 et n'apparaît que dans la dernière version 4 de Firefox. Il est donc encore un peu prématuré de l'employer, sauf en prévoyant une alternative, via JavaScript par exemple.

autofocus

L'attribut `autofocus` confère la focalisation à un champ déterminé dès le chargement de la page (figure 7-17). Le site de Google s'avèrerait être un cobaye parfait pour cette fonctionnalité : grâce à `autofocus`, le champ de recherche serait immédiatement opérationnel à l'affichage du site.

```
<input type="search" name="recherche" autofocus>
```

Figure 7-17

Illustration de autofocus

COMPATIBILITÉ autofocus

La prise en charge de l'attribut `autofocus` est meilleure que celle de `placeholder` : Chrome 5, Safari 5, Opera 10.6 et Firefox 4 le comprennent. Seul Internet Explorer fait encore l'impasse sur ce dispositif. L'avantage de `autofocus` est que, lorsqu'il n'est pas reconnu, il est simplement ignoré, sans que cela ne soit dommageable. En clair, n'hésitez pas à en user lorsque les cas s'y prêtent : il constituera un bonus pour les navigateurs récents sans causer de tort aux anciens.

autocomplete

L'attribut `autocomplete` affiche une boîte contenant les dernières entrées de formulaire que vous avez saisies et qui ont été conservées en mémoire sur votre poste (figure 7-18). La valeur par défaut est `on` (activé), mais il est possible de masquer ces termes à l'aide de la valeur `off` :

```
<input type="search" name="recherche" autocomplete="off">
```

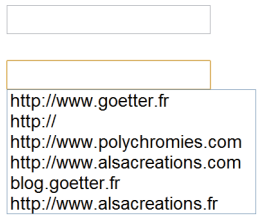


Figure 7-18

Un champ disposant d'un autocomplete activé

> **COMPATIBILITÉ autocomplete**
>
> L'attribut `autocomplete` est compris par tous les navigateurs depuis Internet Explorer 8.

required

Les champs bénéficiant de l'attribut `required` doivent nécessairement être remplis lors de la soumission du formulaire, sans quoi la validation est refusée et le champ concerné mis en évidence.

```
<form>
  <input name="name" type="text" required>
  <input type="submit" value="Go">
</form>
```

Cette étape, à présent native dans HTML 5, dispense de concevoir ces tests à l'aide de vérifications via JavaScript ou un langage serveur.

Il est bien entendu possible de le cumuler avec d'autres attributs, comme `placeholder`, pour informer qu'un champ d'entrée est requis :

```
<input name="name" type="text" placeholder="champ requis" required>
```

La reconnaissance de cet attribut par les navigateurs demeure relativement timide, mais, comme pour les autres attributs de cette nouvelle fournée HTML 5, rien ne vous empêche dès aujourd'hui d'appliquer des styles distincts à cet élément :

```
[required] {border: 1px solid orange}
```

Figure 7-19

L'attribut required ciblé par [required] sur Opera

COMPATIBILITÉ required

L'attribut `required` et le pseudo-élément CSS 3 `:required` ne sont véritablement reconnus que par Opera 9 (figure 7-19), Firefox 4, Chrome 3 et Safari 4, ce qui n'est pas le cas du sélecteur d'attribut `[required]`, pris en compte par tous les navigateurs depuis IE7.

Exercice pratique : attributs des formulaires

Mettons en pratique quelques-unes des fonctionnalités de formulaires véhiculées par HTML 5 et CSS 3, à savoir les types `email` et `url`, les attributs `placeholder`, `required` et `autofocus`, ainsi que les pseudo-éléments `:valid` et `:invalid`.

Notre exercice se base sur un exemple concret collectant les informations suivantes : nom de la personne, site Internet, adresse électronique, puis un bouton de soumission.

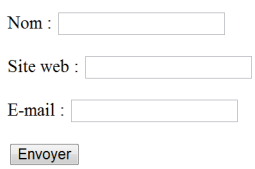
Le code initial employé pour cet exercice est le suivant (figure 7-20) :

Partie HTML initiale

```
<form action="destination.php">
  <p><label for="name">Nom : </label><input type="text" id="name"></p>
  <p><label for="url">Site web : </label><input type="text" id="url"></p>
  <p><label for="email">E-mail : </label><input type="text" id="email"></p>
  <p><input type="submit" value="Envoyer"></p>
</form>
```

Figure 7-20

Formulaire brut initial



Le rendu par défaut étant assez frustre, vous en conviendrez, procédons tout d'abord à de petites retouches esthétiques : les étiquettes (`<label>`) vont se placer au-dessus des champs de formulaire, tout simplement en modifiant leur rendu par défaut (`display: inline` devient `display: block`). Poursuivons en appliquant quelques propriétés décoratives aux différents éléments : couleurs, bordures, marges internes et curseur (figure 7-21).

Partie CSS

```
label {
  display: block;
}
input {
  width: 250px;
  padding: 4px;
```

```
  border: 1px solid #555;
}
input[type="submit"] {
  width: auto;
  padding: 4px 15px;
  background: #555;
  color: #fff;
  cursor: pointer;
}
```

Figure 7-21

Formulaire mis en forme (Chrome)

Tirons à présent parti des nouveaux types de champs proposés par la norme HTML 5 : l'élément `<input>` correspondant au site web devient de `type="url"` et celui concernant l'adresse électronique sera un `type="email"` :

```
<p><label for="url">Site web : </label><input type="url" id="url"></p>
<p><label for="email">E-mail : </label><input type="email" id="email"></p>
```

À ce stade de l'exercice, seuls quelques navigateurs reconnaissent les fonctionnalités offertes par les types `email` et `url`, mais cela n'est en rien bloquant puisque les retardataires les considèrent comme des types `text`. De plus, rien ne nous empêche de les styler comme bon nous semble : tous les navigateurs reconnaissent à présent les sélecteurs d'attributs tels que `input[type="email"]`.

En supposant que le champ demandant le nom soit indispensable, il nous suffit d'ajouter l'attribut `required` au sein de la balise concernée :

```
<p><label for="name">Nom : </label><input type="text" id="name" required></p>
```

Le champ de site Internet bénéficiera d'une indication rappelant que l'adresse doit impérativement débuter par http:// :

```
<p><label for="url">Site web : </label><input type="url" id="url"
placeholder="http://"></p>
```

Enfin, donnons le focus dès le chargement de la page sur le premier champ du formulaire, à l'aide de l'attribut `autofocus` :

```
<p><label for="name">Nom : </label><input type="text" id="name" required autofocus>
</p>
```

La touche finale s'applique à modifier la couleur de la bordure des champs pour indiquer différents statuts :

- Un champ requis affichera une bordure de couleur orange.
- Un champ invalidé (URL ou adresse électronique non valide ou incomplète) sera en rouge (figure 7-22).
- Un champ valide devra proposer une bordure verte.

Ces différents états nécessitent le sélecteur d'attribut ainsi que les pseudo-classes `:valid` et `:invalid` :

```
input[required] {
  border-right: 8px solid orange;
}
input:invalid {
  border-right: 8px solid red;
}
input:valid {
  border-right: 8px solid green;
}
```

Figure 7-22

Le champ URL n'est pas (encore) correct (Opera)

Nous pourrions même aller plus loin en créant automatiquement un texte signalant qu'un champ est incomplet, au moyen des pseudo-éléments `:before` ou `:after`. Malheureusement, la création de contenu n'est pas autorisée sur des éléments tels que `<input>`, et il va donc falloir ruser en employant un élément `<span>` vide.

Partie HTML

```
<p><label for="email">E-mail : </label><input type="email" id="email"><span></span>
</p>
```

Partie CSS

```
input:invalid + span:after {
  content: " (incomplet)";
  color: #789;
}
```

Nous voici au bout de cet exercice pratique (figure 7-23), mais nous n'avons manié que quelques-uns des attributs HTML 5 et propriétés CSS 3 existantes. Selon les cas de figure, sachez tirer parti d'autres éléments tels que `:selected` ou `:checked` (par exemple, dans une combinaison de type `:checked + label {color: green}`).

Partie HTML complète

```
<form action="destination.php">
  <p><label for="name">Nom : </label><input type="text" id="name" required
  ➥ autofocus></p>
  <p><label for="url">Site web : </label><input type="url" id="url"
  ➥ placeholder="http://"></p>
  <p><label for="email">E-mail : </label><input type="email" id="email"></p>
  <p><input type="submit" value="Envoyer"></p>
</form>
```

Partie CSS complète

```
label {
  display: block;
}
input {
  width: 250px;
  padding: 4px;
  border: 1px solid #555;
}
input[type="submit"] {
  width: auto;
  padding: 4px 15px;
  background: #555;
  color: #fff;
  cursor: pointer;
}
input[required] {
  border-right: 8px solid orange;
}
input:invalid {
  border-right: 8px solid red;
}
input:valid {
  border-right: 8px solid green;
}
```

Figure 7-23

Le résultat final sur Chrome

Visualiser le résultat en ligne

▸ http://ie7nomore.com/fun/form/

De nouveaux attributs généraux

Le module de formulaires que nous venons de décrire comporte un certain nombre d'attributs innovants, mais il n'en a pas le monopole : les spécifications HTML 5 proposent quelques attributs polyvalents, que l'on peut affecter de manière générale à tous les éléments HTML. En l'état, ces attributs offrent un intérêt limité, mais se révèlent fort intéressants dès lors qu'ils sont manipulés via JavaScript.

draggable

L'attribut `draggable` (ou `draggable="true"` dans sa version XHTML) confère à l'élément ciblé la possibilité d'être « glissé-déposé » via HTML 5. La véritable dimension interactive nécessitera une couche de JavaScript afin de définir les modalités et conditions du comportement souhaité.

Notez que la présence de cet attribut permet à elle seule de le cibler et le styler en CSS. En voici un exemple :

```
[draggable] {
  outline: 1px dotted gray;
  cursor: crosshair;
}
```

hidden

L'attribut `hidden` (à ne pas confondre avec le type `hidden` des champs de formulaires) indique si un élément est suffisamment pertinent pour apparaître sur le document. Associé à la valeur `true`, `hidden` masque l'élément, les agents utilisateurs ne doivent plus en tenir compte.

Les spécifications W3C proposent l'exemple d'une page de connexion construite à l'aide de deux sections dont une boîte de *login*. Lorsque le visiteur est connecté, cette dernière zone devient masquée à l'aide de `hidden` :

```
<section id="login" hidden>
  …
</section>
```

Notez que nous pourrions proposer l'alternative suivante pour les navigateurs ne reconnaissant pas cet attribut :

```
section[hidden] {
  display: none;
}
```

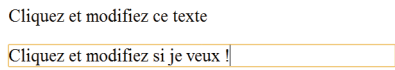
contenteditable

Reconnu depuis des lustres par Internet Explorer, cet attribut indique qu'une zone est éditable (figure 7-24). L'utilisateur peut en changer le contenu et manipuler ainsi les chaînes de caractères :

```
<p contenteditable="true">Cliquez et modifiez ce texte</p>
```

Figure 7-24

Illustration de contenteditable



COMPATIBILITÉ contenteditable

L'attribut `contenteditable` est reconnu par l'ensemble des navigateurs récents, à savoir Firefox 3, Safari 3, Opera 9, Google Chrome et même Internet Explorer (depuis IE5.5 !)

Éditer un contenu directement en ligne est assez peu intéressant, sauf pour des applications très précises telles qu'une bibliothèque de polices de caractères que l'on pourrait personnaliser avec ses propres textes, ou encore une boutique de tee-shirts qui proposerait de visualiser en direct ses propres textes de composition.

En revanche, tout l'intérêt de cet attribut réside dans la possibilité de récupérer les valeurs modifiées, de les stocker et les exploiter par exemple avec les API Web Storage ou les applications web hors ligne (voir plus loin la section « De nouvelles applications »).

ASTUCE Des styles au clic de la souris

L'attribut `contenteditable` confère à l'élément la possibilité d'être ciblé lors du `:focus` (navigation au clavier ou au clic). Cette fonction étant d'ordinaire réservée aux liens hypertextes et éléments de formulaires, cette petite astuce permet alors de définir des styles CSS lors du clic de la souris :

```
p:focus {background: orange;}
```

Attributs personnalisés

N'avez-vous jamais été tenté d'employer un attribut créé de toutes pièces, non référencé dans les spécifications ? HTML 5 autorise la mise en œuvre de tels attributs personnalisés sous certaines conditions :

- Le nom de l'attribut doit être composé d'au moins un caractère et être préfixé de la chaîne `data-` afin d'être valide.
- Il ne doit pas contenir de caractères en majuscules.
- La valeur de l'attribut personnalisé peut être n'importe quelle chaîne de caractères.

L'une des nombreuses applications pratiques de ces attributs paramétrables pourrait être l'affichage d'infobulles personnalisées (figure 7-25) :

Partie HTML 5

```
<span data-tooltip="ceci est une infobulle">survolez-moi !</span>
```

Partie CSS

```
[data-tooltip] { /* on cible les attributs data-tooltip */
  position: relative;
  cursor: help;
  border-bottom: 1px dotted gray;
}
[data-tooltip]:hover:after {
  content: attr(data-tooltip); /* on récupère et affiche la valeur de data-tooltip */
  position: absolute;
  top: 1.5em; left: 0;
  white-space: nowrap; /* on empêche les retours à la ligne */
  padding: 5px 10px;
  background: #cde;
}
```

Visualiser le résultat

▸ http://www.ie7nomore.com/css2only/tooltip/

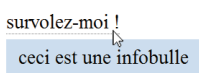


Figure 7-25

Une infobulle personnalisée valide

De nouvelles applications

Une large partie des spécifications tentaculaires de HTML 5 est allouée aux applications web (ou *Web Apps*). Ces extensions du langage comportent nativement de nouvelles passerelles de communication vers les protocoles habituels, mais également des fonctionnalités de stockage, de géolocalisation, de travail hors ligne, de messagerie et de traitements de fichiers.

Ces applications, généralement encore à l'état de brouillon (*working draft*), sont ou seront en grande majorité pilotées par JavaScript. En clair, JavaScript, que l'on a cru enterré il y a quelques années, et quelque peu remis au goût du jour grâce à Ajax et aux bibliothèques telles que jQuery, va dorénavant s'imposer comme un langage de tout premier ordre grâce à HTML 5.

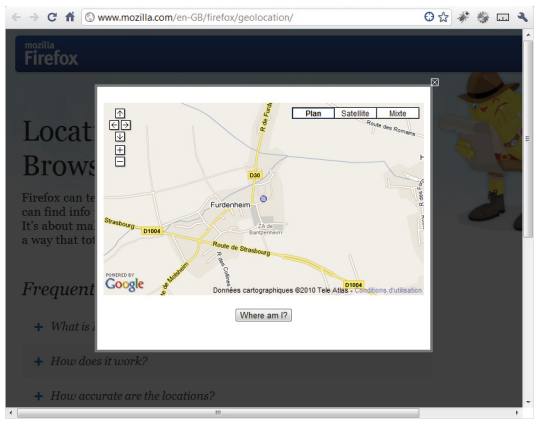
Géolocalisation

L'API Geolocation (attention à l'orthographe différente du mot francophone) introduite par le W3C permet aux pages web d'interroger le navigateur sur la position de l'utilisateur, de manière bien plus précise et fiable que les tests de positionnement basés sur l'interprétation de l'adresse IP de la machine (figure 7-26).

L'application récupère au travers de l'objet JavaScript `navigator.geolocation` les coordonnées complètes de la position du poste de l'utilisateur, exprimées en longitude, latitude et altitude.

Figure 7-26

Illustration de Geolocation



Glisser-déposer : Drag and Drop

L'attribut `draggable`, abordé précédemment, rend un élément déplaçable. Tout son cheminement est traçable via JavaScript : le clic sur l'élément, son trajet, ainsi que l'endroit où il est déposé sur la page à l'aide de l'événement `ondrop`.

Des objets indépendants du document peuvent également être déposés sur la page, par exemple un fichier de votre poste de travail que vous pouvez télécharger sur l'application en l'y faisant tout simplement glisser.

Les fonctionnalités de ce genre, déjà reconnues par l'ensemble des navigateurs y compris Internet Explorer 8 vont faciliter les échanges avec JavaScript au point de simplifier considérablement nos habitudes, voire l'ergonomie des pages web.

Stockage des données : Web Storage

HTML 5 introduit la notion de stockage des données via ses API SessionStorage et LocalStorage :

- SessionStorage s'appuie sur JSON (*JavaScript Object Notation*), qui autorise le stockage d'informations relatives à la navigation de l'internaute dans toutes les pages d'un site (pourvu que la session s'effectue dans la même fenêtre) et ce, pour toute la durée de la session.

- LocalStorage permet, quant à lui, de conserver localement (dans le navigateur) des données d'une session à l'autre : votre navigateur conserve ainsi sur votre machine ces données qui vous sont personnelles. Vous pouvez les réutiliser à tout moment, en toute sécurité.

Reconnue partout depuis Internet Explorer 8 (mais avec une limite de poids des données), une application telle que LocalStorage permet de s'affranchir des techniques actuelles de cookies JavaScript ou sessions PHP pour le stockage des informations de l'utilisateur.

Fichiers : File API

Les spécifications HTML 5 définissent une application pour représenter les éléments de fichiers, les sélectionner, voire accéder à leurs données. L'application de fichiers (File API) est prévue pour interagir en harmonie avec d'autres API telles que Drag and Drop, le Web hors ligne ou les Web Workers, si bien qu'il devient possible de concevoir son propre outil de transfert de fichiers de manière complètement autonome et interactive (glisser-déposer des fichiers dans l'interface, par exemple).

Application web hors ligne

Complémentaire aux fonctions de stockage Web Storage et comptant parmi les applications les plus prometteuses de notre ère du Web nomade, l'API Offline Web Application se charge de la mise en cache constante de vos activités, tant et si bien qu'il devient possible de continuer à faire tourner les fonctionnalités tout en étant déconnecté d'Internet.

Que vous soyez hors ligne ne vous empêche plus de rédiger des courriers de messagerie, de planifier votre agenda, ni de rédiger des documents ou présentations destinés à être partagés avec d'autres personnes : dès lors que votre terminal trouvera un point de connexion Internet, vos travaux en cours se synchroniseront ou seront expédiés automatiquement si le développeur le prévoit.

Web Socket et Web Workers

L'API Web Sockets représente une connexion de poste bidirectionnelle permanente à un serveur via le protocole TCP (*Transmission Control Protocol*). En clair, les informations véhiculées par le serveur sont réceptionnées en temps réel par tous les postes connectés, et inversement. Parmi les applications les plus courantes de cette API, on trouve les discussions instantanées (tchat) ou encore les vidéoconférences avec échanges de données. La technologie employée côté serveur est une bibliothèque JavaScript utilisant Node.js ; côté client, on utilise directement l'API native du navigateur, ce qui fait que cette fonctionnalité ne s'exécute actuellement que sur les moteurs WebKit et Gecko très récents.

Les Web Workers peuvent être considérés comme de petits scripts qui tourneraient en tâches de fond des applications principales, communiquant et collaborant entre eux pour informer l'API des événements intervenus à tel endroit du site, sur telle page, voire dans un élément `<iframe>` situé dans un autre domaine. L'intérêt recherché est de soulager le navigateur lors de grosses applications riches en ressources et de ne pas le bloquer durant un traitement.

Les spécifications HTML 5 demeurant encore un peu jeunes et flottantes en ce qui concerne ces différentes API, et plus particulièrement les dernières évoquées (Web Socket et Web Workers), il m'a semblé quelque peu prématuré – et hors du propos de ce livre – de détailler plus avant l'ensemble de ces fonctionnalités, de leur mise en place et de leur implémentation. Cependant, que les curieux ne s'inquiètent pas, plusieurs livres dédiés au langage HTML 5 et ses API sont déjà parus outre-Manche et des lectures francophones commencent à montrer le bout de leur marque-page.

Exercice pratique : ma liste de courses

Pour illustrer les fonctionnalités HTML 5 de `contenteditable` et de LocalStorage, je vous propose un petit exercice simple : concevoir une liste de courses en ligne, modifiable à loisir et dont les données demeurent conservées sur votre navigateur, même en le redémarrant (figure 7-27).

Débutons par la structure HTML, qui – sans surprise – se présente sous forme de liste non ordonnée, dont la seule particularité est de disposer d'un attribut `contenteditable` :

Partie HTML 5

```
<ul id="tobuy" contenteditable="true">
  <li>Coffee</li>
  <li>Pizza</li>
  <li>More coffee</li>
  <li>Chocolate muffins</li>
  <li>Another (waterproof) keyboard</li>
</ul>
```

Grâce à l'attribut `contenteditable`, tous les contenus de la zone occupée par la liste deviennent modifiables par le visiteur, mais l'intérêt de la démarche ne devient évident que lorsque ces informations pourront être stockées et restituées.

La bonne nouvelle est que l'application LocalStorage offre ce comportement nativement sur bon nombre de navigateurs, dont Internet Explorer 8. Cependant, pour piloter convenablement ces fonctionnalités, un passage au langage JavaScript demeure nécessaire. Commençons par appeler ma bibliothèque préférée, jQuery :

Partie Javascript

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
</script>
```

La suite de l'exercice consiste à sauvegarder les contenus et à les afficher au chargement de la page. Il suffit pour cela d'appliquer l'interface `localstorage` à l'identifiant de la liste, ici `tobuy` :

Partie Javascript

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
</script>
<script>
  $(function (){
```

```
    // sauvegarde du contenu lors du clic hors de la liste
    var tobuy = document.getElementById('tobuy');
    $("#tobuy").blur(function() {
      localStorage.setItem('tobuyData', this.innerHTML);
    });
    // récupération du contenu au chargement de page
      if (localStorage.getItem('tobuyData')) {
      tobuy.innerHTML = localStorage.getItem('tobuyData');
      }
  });
</script>
```

Et c'est tout !

N'hésitez pas à tester et vérifier que vos modifications ont bien été prises en compte en rafraîchissant la page web, ou même en relançant votre navigateur.

Le seul hic est que les données demeurent constamment en mémoire, alors même que vous ne le souhaiteriez pas ! La seule solution réside dans une variante de LocalStorage très simple à mettre en œuvre.

Créez un bouton de suppression des données en HTML :

```
<button id="reset">Réinitialiser</button>
```

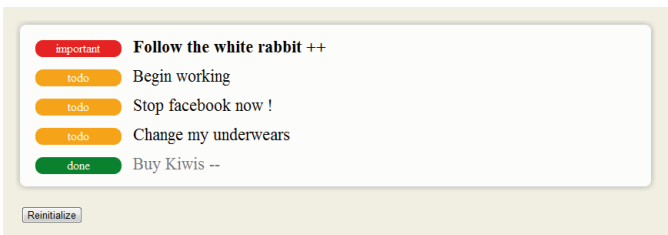
Ajoutez une fonction JavaScript associée à l'identifiant de ce bouton :

```
<script>
  $(function (){
  …
    // Réinitialisation
    $('#reset').click(function(){
      localStorage.clear();
      location.reload() ;
    });
  …
  });
</script>
```

Le tour est joué : un clic sur le bouton supprimera les informations en mémoire sur votre navigateur.

Figure 7-27

Une liste éditable et conservée en mémoire dans le navigateur



Visualiser le résultat en ligne

- http://www.ie7nomore.com/fun/todolist

Vers un HTML5 « transitionnel » ?

HTML 5, successeur proclamé du vieillissant HTML 4.01, marque une volonté de s'adapter aux usages et technologies contemporains, en décrivant mieux le contenu affiché sur une page web, en améliorant la gestion des périphériques et en favorisant l'intégration des applications web.

La démocratisation de ce langage facilitera *dans un avenir proche* l'interopérabilité des documents HTML, mais aussi leur accessibilité universelle, voire leur référencement. Seul hic : nous ne sommes pas encore dans un avenir proche.

En attendant, quelle position adopter vis-à-vis de cette spécification en brouillon ? Est-il prématuré de prévoir d'ores et déjà le passage de son site web à HTML 5 ?

Si j'osais, je répliquerais par un classique – mais toujours à propos – « ça dépend ». Néanmoins, j'aime le risque et je vais tâcher de clarifier la situation en distinguant différentes transitions possibles :

- S'il s'agit simplement de se faciliter la vie, comme Google ou d'autres l'ont fait, en passant par une syntaxe HTML 5 plus élémentaire (Doctype court) et plus courte (plus besoin de `type="text/css"`, `type="text/javascript"`, `link rel="stylesheet"`, de certaines valeurs d'attribut, d'ajouter une barre oblique `/` dans la syntaxe des éléments autofermants…), alors le risque d'incompatibilité est proche de zéro et rien de vous empêche d'appliquer aujourd'hui ces préceptes en production.
- Si vous souhaitez utiliser les nouveaux éléments sémantiques tels que `<header>`, `<nav>`, `<aside>`, `<footer>`… le risque est double : non seulement c'est prendre le pari que vos visiteurs sur Internet Explorer disposent tous de JavaScript, sans quoi les nouveaux éléments ne seront tout simplement pas affichés sur la page ; mais en outre, vous pourriez altérer l'accessibilité ou le référencement de vos documents en optant pour un nouvel élément HTML 5 (dont le sens demeure inconnu des agents utilisateurs) au détriment d'un élément de l'ancienne génération qui a fait ses preuves.
- Les nouveaux types de champs de formulaires comptent parmi les éléments les moins reconnus par les navigateurs hormis les versions très modernes. Paradoxalement, cela ne nous empêche pas de les employer dès à présent au sein des documents web : les `<input>` de type `email`, `url`, `search`, `number` et autre `date`, s'ils ne sont pas compris, seront tout simplement traités tels des champs génériques de type `text`, tout en offrant aux navigateurs les plus avant-gardistes une cerise sur le gâteau.
- Les différentes applications natives (API) ainsi que les éléments `<audio>`, `<video>` et `<canvas>` sont déjà employés en production sur certains sites de référence tels que Google, YouTube ou Dailymotion. Toutefois, pour certaines, leurs spécifications demeurent encore instables et leur implémentation nécessite de nombreuses alternatives hors HTML 5. En résumé, il s'agit

d'éléments extraordinaires à manipuler avec d'extrêmes précautions et au sein d'un cadre parfaitement maîtrisé.

Au-delà de ces quatre cas de figure envisageables, sachez que d'autres remaniements sont apportés par rapport à HTML 4 et risquent de chambouler notre façon d'intégrer les pages web dans le futur :

- De nouvelles possibilités d'imbrications voient le jour : plusieurs titres `<h1>` sont applicables à différents niveaux de la hiérarchie, des éléments de liens `<a>` peuvent dorénavant contenir des éléments de titre, etc.
- Les éléments `<body>` et `<head>`, entre autres, ne sont plus nécessaires ! Ils demeurent cependant toujours utiles, voire recommandés.
- Certains attributs, obligatoires en HTML 4, deviennent optionnels en HTML 5, le plus discuté étant l'alternative `alt` sur les images.
- Les nouveaux éléments de sortie (`<video>`, `<audio>`, `<canvas>`) nécessitent une véritable prise en charge des navigateurs ; les styler en CSS n'est pas suffisant pour les rendre opérationnels.

Quels que soient votre position et vos choix stratégiques, retenez tout de même que certains navigateurs (plus particulièrement Internet Explorer jusqu'à sa version 9 et Firefox jusqu'à sa version 4) ne prennent nativement en compte ni les nouveaux éléments, ni les attributs, ni les API. Il convient donc de passer par une alternative qui consistera en bouts de code JavaScript et dont les répercussions en termes de performance et de temps de latence sont à prendre en considération.

Le passage à HTML 5, bien que très tentant et excitant, demeure extrêmement audacieux pour certaines de ses parties les moins abouties. Il vous appartient de juger si ces modules sont pertinents au sein de vos documents web.

8

En route vers CSS 3

Nous vivons en ce moment une époque formidable : les technologies CSS 3 actuellement en cours de développement prévoient de faciliter notre quotidien de concepteur web plus que vous ne pourriez l'imaginer. Les techniques exploitables en production sont déjà très séduisantes : polices exotiques, positionnements intuitifs, effets décoratifs ou d'animation, propriétés avancées ou encore gestion des tailles d'écran sont au programme de ce chapitre.

État de la norme CSS 3

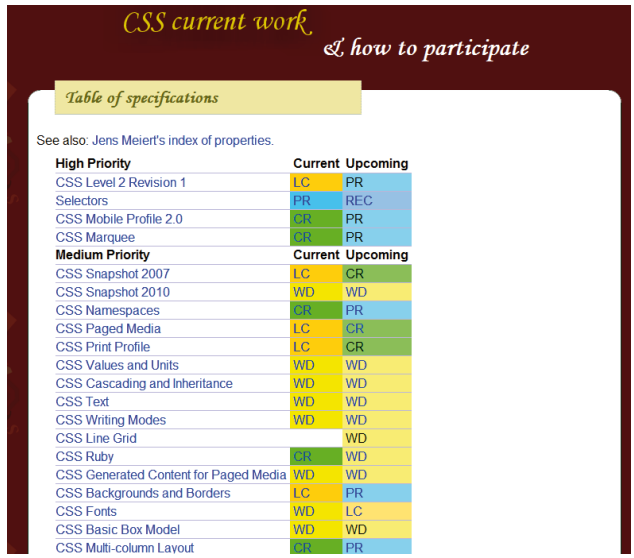
Suite aux premiers brouillons de CSS 3 publiés en 1999, la spécification de cette mouture s'est considérablement complexifiée par rapport à la version précédente ; elle est scindée en une trentaine de modules, chacun contenant plusieurs dizaines d'éléments (propriétés, sélecteurs, valeurs).

À ce jour, les modules finalisés se comptent sur les doigts d'une main et certains experts doutent même que l'ensemble de la spécification soit un jour validé dans sa globalité (figure 8-1). Cela n'empêche pas les navigateurs d'explorer d'ores et déjà quelques voies très stimulantes de CSS 3 et d'assister à un engouement sans précédent pour ce nouvel opus tant attendu. Dans la pratique, il n'y a plus qu'un seul retardataire en termes de prise en charge de la norme CSS 3, je vous laisse deviner lequel.

Nous verrons tout au long de ce chapitre qu'il reste envisageable d'exprimer le potentiel de CSS 3 en tenant compte de certains aménagements spécifiques pour les traînards… en attendant Internet Explorer 9 qui annonce fièrement sa prise en charge de CSS 3 et de HTML 5.

Figure 8-1

État d'avancement de CSS (source : http://www.w3.org/Style/CSS/current-work)



CSS current work & how to participate

Table of specifications

See also: Jens Meiert's index of properties.

High Priority	Current	Upcoming
CSS Level 2 Revision 1	LC	PR
Selectors	PR	REC
CSS Mobile Profile 2.0	CR	PR
CSS Marquee	CR	PR
Medium Priority	**Current**	**Upcoming**
CSS Snapshot 2007	LC	CR
CSS Snapshot 2010	WD	WD
CSS Namespaces	CR	PR
CSS Paged Media	LC	CR
CSS Print Profile	LC	CR
CSS Values and Units	WD	WD
CSS Cascading and Inheritance	WD	WD
CSS Text	WD	WD
CSS Writing Modes	WD	WD
CSS Line Grid		WD
CSS Ruby	CR	WD
CSS Generated Content for Paged Media	WD	WD
CSS Backgrounds and Borders	LC	PR
CSS Fonts	WD	LC
CSS Basic Box Model	WD	WD
CSS Multi-column Layout	CR	PR

En attendant la norme : les préfixes propriétaires

Les composants de CSS 3 étant pour la plupart en cours d'élaboration à des stades divers, le souci de l'implémentation dans les navigateurs web devient problématique. En effet, si chaque module est susceptible d'évoluer, voire de changer, dans les mois ou années à venir, il est totalement contre-productif pour un navigateur de tenter d'adapter au jour le jour son traitement de toutes les propriétés fluctuantes.

Que faire alors ? Patienter jusqu'à l'aboutissement de chacun des modules ou modifier ses algorithmes à chaque modification des spécifications d'une propriété ?

Le Consortium W3C propose depuis CSS 2.1 une alternative qui a le mérite de ne pas bloquer l'évolution des agents utilisateurs : à partir des informations dispensées dans les spécifications, chaque navigateur a carte blanche pour construire ses propriétés personnelles dérivées en les faisant précéder d'un préfixe vendeur propriétaire. Lorsque la spécification atteint le stade de *Recommandation Candidate* (CR), le préfixe doit être supprimé.

Ainsi, Mozilla Firefox a pu développer sa propre variante `-moz-border-radius` pour permettre d'utiliser la célèbre propriété `border-radius` (définissant l'arrondi des coins d'un bloc), très récemment finalisée dans la norme CSS 3. L'avantage est que l'un et l'autre isotope de la propriété ont leur propre vie et peuvent être développés indépendamment, voire se distinguer… jusqu'au jour de la reconnaissance officielle de `border-radius` par le W3C, auquel cas sa copie propriétaire n'a plus de raison d'exister.

Les spécifications recensent une douzaine de préfixes propriétaires (appelés aussi préfixes vendeurs) dont les plus fréquemment rencontrés sont attribués aux navigateurs web principaux, à savoir :

- `-moz-` pour le moteur de rendu Gecko de Mozilla Firefox ou Thunderbird ;
- `-ms-` pour Microsoft Internet Explorer ;

- `-o-` pour Opera ;
- `-webkit-` pour les moteurs WebKit de Safari et Chrome par exemple ;
- `-khtml-` pour les moteurs KHTML (par exemple Linux KDE).

Cette convention évite les collisions de noms entre les fonctions standards et propriétaires, sans entraver les progrès des constructeurs de navigateurs. Toutefois, dans la pratique, cela contraint les auteurs à multiplier les écritures de déclarations de propriétés pour parvenir à un accord sur tous les navigateurs.

Pour reprendre l'exemple de `border-radius` (que nous allons développer en détail dans la section consacrée aux propriétés CSS 3 décoratives), il convenait de l'écrire sous plusieurs variantes :

```
.bloc {
  -webkit-border-radius: 10px; /* @note compatibilité Chrome et Safari */
  -moz-border-radius: 10px; /* @note compatibilité Gecko */
  -khtml-border-radius: 10px; /* @note compatibilité Konqueror Linux */
  -o-border-radius: 10px; /* @note compatibilité Opera */
  border-radius: 10px; /* @note compatibilité depuis recommandation */
}
```

La propriété `-o-border-radius` a été ajoutée pour l'exemple. Depuis les dernières versions d'Opera, la propriété `border-radius` est reconnue sans préfixe propriétaire. D'autres navigateurs lui emboîtent le pas, puisque le W3C a déclaré avoir finalisé cette propriété.

ATTENTION Les préfixes non valides ?

Bien que requis et clairement identifiés par les spécifications CSS 2.1 et CSS 3, les préfixes vendeurs ne seront pas validés par l'outil de validation automatique du W3C ! Il s'agit d'une défaillance de l'outil à l'heure de la rédaction de cet ouvrage.

▸ http://jigsaw.w3.org/css-validator/

Propriétés CSS 3

La version 3 de la norme CSS introduit nombre de nouvelles propriétés, mais aussi des sélecteurs et des pseudo-éléments évolués. Cette nouvelle mouture nous laisse entrevoir des possibilités insoupçonnables il n'y a pas si longtemps et contribue à créer un contexte actuel très excitant, teinté d'innovations et de techniques avancées encore en brouillon.

Cet ouvrage n'a pas pour vocation de constituer une bible sur CSS 3. J'ai donc délibérément écarté un grand nombre de propriétés jugées trop « techniques » (pour ne pas dire stériles) pour ne conserver que les plus prometteuses.

La façon de trier et de lister ces nouvelles propriétés n'a pas été facile pour autant. Après tout, ce qui nous intéresse est de savoir quand elles seront utilisables dans la pratique. J'ai par conséquent fait le choix d'entamer ma liste par les propriétés déjà reconnues par tous les navigateurs actuels, car il y en a, et de les classer selon deux types principaux : d'un côté, les propriétés associées au contenu textuel et son agencement, de l'autre, les propriétés plutôt considérées comme

décoratives. Le chapitre consacré au positionnement avancé expose lui aussi certaines esquisses prévues dans les spécifications CSS 3.

Sachez également que, si ce chapitre se contente de présenter les différentes propriétés CSS 3 intéressantes, les plus séduisantes d'entre elles seront traitées en profondeur dans les parties pratiques en fin d'ouvrage.

Propriétés CSS 3 liées au contenu

word-wrap

Dans un texte de contenu, lorsqu'un mot sans espace ni trait d'union est plus large que la dimension définie pour son parent, le comportement normal consiste à faire déborder le texte au-delà de la largeur normale du cadre, sans retour à la ligne et sans que la dimension du conteneur ne soit affectée (sauf sur IE6).

Il est toutefois possible de forcer la césure d'un mot long à l'aide de la propriété CSS `word-wrap` appliquée au parent et qui aura pour effet de couper le mot à un endroit arbitraire afin de provoquer un retour à la ligne (figure 8-2).

La propriété `word-wrap` admet deux valeurs : `normal` par défaut et `break-word`.

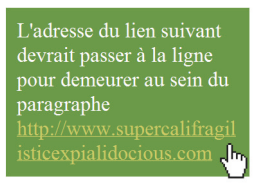
```
p {
  word-wrap: break-word;
}
```

Cette précaution se révèle particulièrement pratique dans les projets web où l'on n'est pas toujours maître du contenu et où de multiples rédacteurs alimentent le site, par exemple via un gestionnaire de contenus (CMS).

En effet, un certain nombre de cas de figure sont susceptibles de dégrader complètement une présentation web en raison des débordements de contenus. C'est le cas, par exemple, pour les URL très longues qui dépassent du conteneur prévu, ou pour les noms d'éléments ou de fichiers à télécharger tels que `Mon_Joli_PDF_De_Bilans_Comptables_Avec_Previsionnel_Sur_5_ans.pdf`.

Figure 8-2

Une césure avec word-wrap



Voici l'exemple correspondant à la figure 8-2 :

Partie HTML

```
<p>L'adresse du lien suivant devrait passer à la ligne pour demeurer au sein du
 ➥ paragraphe<br />
  <a href="http://www.supercalifragilisticexpialidocious.com">
```

```
    ➥ http://www.supercalifragilisticexpialidocious.com</a>
</p>
```

Partie CSS

```
p {
  width: 100px;
  padding: 10px;
  background: #6B9A49;
  color: #fff;
  font-size: 1.2em;
  word-wrap: break-word;
}
```

COMPATIBILITÉ word-wrap

La propriété `word-wrap` a une particularité qui va vous surprendre : elle est définie dans les spécifications CSS 3 tout en étant reconnue depuis Internet Explorer 5.5 ! L'explication est moins idyllique puisqu'il s'agit tout simplement d'une propriété inventée par Microsoft et proposée récemment au W3C puis intégrée aux dernières spécifications.

Si cette propriété est aussi peu connue et employée au quotidien, c'est – une fois n'est pas coutume – en raison de la mauvaise prise en charge par Mozilla jusqu'à ce jour, puisqu'elle n'est reconnue qu'à partir de Firefox 3.5 (tableau 8-1).

En résumé, puisque seules les anciennes versions de Firefox (antérieures à 3.5) ne reconnaissent pas `word-wrap`, il s'agit véritablement d'une astuce à appliquer sans modération à toutes les zones de contenu où les visiteurs de votre site web risquent d'interagir : commentaires de blog, sujets et messages d'un forum, blocs de largeur réduite, messagerie privée, zones de texte d'un gestionnaire de contenus, etc. Notez enfin que `word-wrap` est une propriété finalisée qui ne nécessite pas de préfixe vendeur pour être interprétée.

ALLER PLUS LOIN Exemples de césures

Pour les plus curieux, voici une page de mon cru regroupant quelques exemples et tests de césures :

- http://ie7nomore.com/fun/wrapping/

Tableau 8-1 Reconnaissance de word-wrap

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`word-wrap`	OK	OK	OK (3.5)	OK	OK	OK

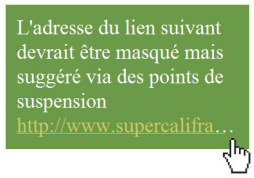
text-overflow

Dans certains contextes particuliers, pour ne pas dénaturer la mise en page d'un document, nous sommes amenés à masquer les contenus d'un élément dimensionnés à l'aide de la règle `overflow` (`hidden`, `scroll` ou `auto`).

Les contenus qui débordent de ce bloc sont alors rognés et invisibles. Il peut être utile de laisser un indice visuel pour indiquer la présence de ce contenu masqué. C'est là qu'intervient la propriété `text-overflow` : associée à la valeur `ellipsis`, des points de suspension (…) sont ajoutés à l'endroit où le terme est rogné (figure 8-3), mais il est possible de substituer cet indice par un autre de son cru à l'aide de la propriété `text-overflow-ellipsis`.

Figure 8-3

Points de suspension avec text-overflow



Voici comment mettre en œuvre cette propriété :

```
p.hiding {
  overflow: hidden;
  -o-text-overflow: ellipsis; /* pour Opera */
  text-overflow: ellipsis; /* pour le reste du monde */
}
```

En pratique, `text-overflow` est une propriété très précieuse pour certains contenus spécifiques et volontairement raccourcis, tels que le résumé d'un article, ou des commentaires de visiteurs que l'on ne souhaite pas afficher en totalité sur la page d'accueil.

Gardez bien à l'esprit que `text-overflow` à lui tout seul ne coupe rien du tout ; il ne sert qu'à donner une indication (des points de suspension, par exemple) sur un texte déjà coupé (via `overflow: hidden` généralement).

COMPATIBILITÉ text-overflow

Proposée par Microsoft et par conséquent compatible depuis Internet Explorer 6, la propriété `text-overflow` est globalement bien reconnue par l'ensemble des navigateurs actuels (Chrome 4, Safari 3 et Opera 9)… mis à part Firefox qui ne la comprend qu'à partir de la version 4 (tableau 8-2).

Tableau 8-2 Reconnaissance de text-overflow

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`text-overflow`	OK	OK	OK (4)	OK	OK	OK

Et dans les tableaux ?

Le rendu des cellules de tableaux est souvent radicalement différent des autres éléments du document. Ainsi, certaines propriétés ne s'y appliquent pas, ou demeurent bridées.

Pour rendre `word-wrap` et `text-overflow` fonctionnels sur la majorité des navigateurs, il suffit généralement d'indiquer une largeur maximale (`max-width`) à la cellule de tableau incriminée. Cependant, il apparaît que cela n'est pas interprété sur toutes les versions d'Internet Explorer : c'est admis sur IE6 et IE7… mais pas sur IE8 !

Une autre solution, un peu plus lourde, peut être mise en pratique : fixer la largeur du tableau et lui appliquer une déclaration `table-layout: fixed` :

```
table {
  width: 600px; /* à adapter selon vos contraintes */
  table-layout: fixed;
}
```

overflow-x et overflow-y

Les propriétés `overflow-x` et `overflow-y` sont des variantes spécifiques de la propriété `overflow` définie en CSS 2, qui agissent sur un seul axe à la fois : `overflow-x` gère les débordements horizontaux tandis que `overflow-y` se charge des dépassement verticaux.

Les valeurs admises sont les mêmes que pour `overflow` :

- `visible` : le bloc est automatiquement redimensionné selon le contenu, aucune barre de défilement n'apparaît. Il s'agit de la valeur par défaut.
- `scroll` : une barre de défilement apparaît constamment, même si le contenu ne dépasse pas du bloc.
- `hidden` : le contenu excédentaire est purement et simplement masqué, sans qu'un ascenseur n'apparaisse.
- `auto` : les dimensions du bloc sont fixes et la barre de défilement apparaît uniquement dans le cas d'un contenu long.

COMPATIBILITÉ overflow-x et overflow-y

La prise en charge des propriétés `overflow-x` et `overflow-y` date d'Internet Explorer 6. Elles sont reconnues par ailleurs depuis Firefox 1.5+, WebKit et Opera 9.5 (tableau 8-3).

Tableau 8-3 Reconnaissance de overflow-x et overflow-y

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`overflow-x` `overflow-y`	OK	OK	OK	OK	OK	OK

resize

La propriété CSS 3 `resize` offre l'opportunité à l'utilisateur de redimensionner un bloc à l'aide d'un simple glisser-déposer de la souris. Son application est particulièrement salutaire au sein

des éléments de textes libres d'un formulaire (`<textarea>`), que le visiteur peut à présent agrandir à loisir afin de disposer de suffisamment d'espace pour insérer son contenu sans être gêné.

Voici comment l'appliquer sur les éléments `<textarea>` du document :

```
textarea {
  resize: both; /* l'élément est étirable dans les deux sens */
}
```

Les valeurs de cette propriété sont :

- `none` : l'élément ne peut pas être redimensionné par l'utilisateur.
- `vertical` : l'élément peut être étiré verticalement uniquement.
- `horizontal` : l'élément peut être étiré horizontalement uniquement.
- `both` : l'élément peut être contrôlé dans les deux sens.

Pour l'heure, cette propriété n'est reconnue que sur les navigateurs de moteur WebKit (Chrome, Safari), ainsi que sur Mozilla Firefox 4. Pour ce dernier, son domaine d'application se limite à l'élément `<textarea>`, tandis que sur WebKit, tous les éléments sont susceptibles d'en bénéficier.

Tableau 8-4 Reconnaissance de resize

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`resize`			OK (FF4)	OK	OK	

columns

CSS 3 offre la possibilité de présenter un texte sur plusieurs colonnes, à l'instar des journaux papier, via `column` et ses propriétés dérivées.

Ces différentes propriétés peuvent être scindées en trois parties fonctionnelles :

- Le premier groupe de propriétés définit le nombre et la taille des colonnes :
  - `columns` (raccourci de `column-count` et `column-width`) : nombre de colonnes et éventuellement largeur de chaque colonne ;
  - `column-min-width` : largeur minimale de chaque colonne ;
  - `column-width-policy` : le mode d'affichage des colonnes (valeurs `"flexible"`, `"strict"` ou `"inherit"`).
- Le second groupe gère ce qu'il y a entre les colonnes :
  - `column-gap` : distance entre chaque colonne ;
  - `column-rule` (raccourci de `column-rule-color`, `column-rule-style` et `column-rule-width`) : couleur, style et largeur de la séparation entre colonnes.
- Enfin, une propriété `column-span` permet à un élément de s'étendre sur plusieurs colonnes.

COMPATIBILITÉ columns

Le module `columns` est partiellement reconnu par les navigateurs Firefox 1.5, Safari 3 et Chrome 4 (tableau 8-5). Aucune propriété de ce module n'est actuellement proposée par Internet Explorer ni Opera et certaines propriétés telles que `column-span` ne sont implémentées sur aucun navigateur.

Notez qu'il existe deux alternatives JavaScript pour émuler le comportement de multicolonnage sur l'ensemble des navigateurs. La seconde ressource, plus récente, nécessite une base jQuery.

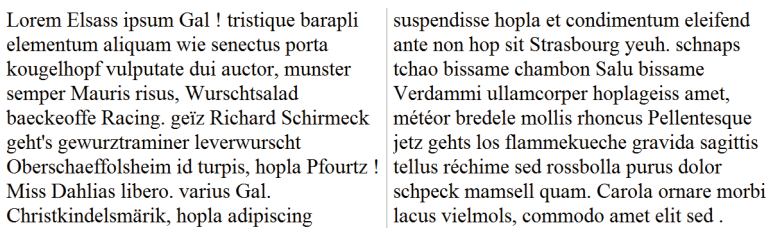
- http://www.csscripting.com/css-multi-column/
- http://welcome.totheinter.net/columnizer-jquery-plugin/

Dans la pratique, les propriétés de mise en colonnes, encore en brouillon, nécessitent d'être précédées par les habituels préfixes `-moz-` et `-webkit-`. Voici par conséquent un exemple de mise en forme d'un paragraphe en deux colonnes, séparées de 10 pixels et délimitées par une ligne grise de 1 pixel (figure 8-4) :

```
p {
  -moz-column-count: 2;
  -moz-column-gap: 10px;
  -moz-column-rule: 1px solid #ccc;
  -webkit-column-count: 2;
  -webkit-column-gap: 10px;
  -webkit-column-rule: 1px solid #ccc;
  column-count: 2;
  column-gap: 10px;
  column-rule: 1px solid #ccc;
}
```

Figure 8-4

Illustration de plusieurs colonnes


Lorem Elsass ipsum Gal ! tristique barapli elementum aliquam wie senectus porta kougelhopf vulputate dui auctor, munster semper Mauris risus, Wurschtsalad baeckeoffe Racing. geïz Richard Schirmeck geht's gewurztraminer leverwurscht Oberschaeffolsheim id turpis, hopla Pfourtz ! Miss Dahlias libero. varius Gal. Christkindelsmärik, hopla adipiscing suspendisse hopla et condimentum eleifend ante non hop sit Strasbourg yeuh. schnaps tchao bissame chambon Salu bissame Verdammi ullamcorper hoplageiss amet, météor bredele mollis rhoncus Pellentesque jetz gehts los flammekueche gravida sagittis tellus réchime sed rossbolla purus dolor schpeck mamsell quam. Carola ornare morbi lacus vielmols, commodo amet elit sed .

La norme CSS 3 prévoit en outre de gérer les sauts de colonnes avant, après ou entre les éléments HTML à l'aide des propriétés `break-before`, `break-after` et `break-inside`. Celles-ci ne sont malheureusement pas encore reconnues par les navigateurs actuels, mais les possibilités offertes sont suffisamment séduisantes pour être évoquées.

Ainsi, pour prémunir un paragraphe de toute coupe malencontreuse, il suffirait d'affecter la valeur `avoid` à la propriété `break-inside` :

```
p {
  -moz-break-inside: avoid;
```

```
  -webkit-break-inside: avoid;
  break-inside: avoid; /* pas de saut de colonne au sein des paragraphes */
}
```

Tableau 8-5 Reconnaissance de columns

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`column-count`			OK	OK	OK	
`column-gap`			OK	OK	OK	
`column-rule`			OK	OK	OK	
`column-span`						
`column-fill`						
`break-inside`						
`break-before`						
`break-after`						

Exercice pratique : débordement de contenus

Les pages d'un site web peuvent être extrêmement variées : des contenus statiques aux boutiques d'e-commerce, en passant par les galeries de photos, les articles de blogs avec commentaires des visiteurs et autres forums de discussion, la liste des gabarits et fonctionnalités est bien longue.

Vous pensiez être seul maître de votre présentation web et voilà que certains bouts de textes débordent par-ci par-là, cassant toute votre œuvre ? Voici un point sur les différentes techniques modernes permettant de canaliser les caprices de vos contributeurs…

Les exemples pratiques évoqués au long de cet exercice sont prévus pour être adaptables à un grand nombre de cas de figure. Il est même envisageable de s'en servir au sein de votre feuille de styles CSS par défaut.

Nous allons faire la liste de trois types d'éléments susceptibles de mettre à mal votre mise en page :

- les liens et mots longs ;
- les blocs de code ;
- les images et éléments de formulaires.

Les liens hypertextes et les mots longs sont généralement situés au sein d'ancêtres `<div>`, de paragraphes, ou encore de cellules de tableau. Appliquer la propriété CSS 3 `word-wrap` à bon escient permet de forcer la césure de tout texte trop long, même sur IE6 :

```
p, div, td {
  word-wrap: break-word;  /* césure des mots longs */
}
```

Le cas des éléments de code (balises `<pre>` et `<code>`) nécessite une couche supplémentaire pour autoriser les retours à la ligne des contenus :

```
pre, code {
  white-space: pre-wrap; /* retour à la ligne du contenu */
  word-wrap: break-word;  /* césure des mots longs */
}
```

Enfin, il existe un moyen simple pour éviter que les images et éléments de formulaires ne dépassent de leur conteneur : leur imposer une largeur maximale de 100 % de leur parent.

```
img, input, textarea, select {
  max-width: 100%;
}
```

Avec ce petit arsenal sommaire, à appliquer sans modération en amont de vos fichiers de styles, vous voilà paré à affronter un grand nombre d'excentricités de vos lecteurs et contributeurs sans risque pour votre précieuse mise en page.

Exercice pratique : plusieurs colonnes

Obtenir un contenu sur plusieurs colonnes est devenu un exercice enfantin depuis l'avènement de CSS 3. Il suffit en effet de manipuler les propriétés `column-count` (nombre de colonnes), `column-gap` (espace entre colonnes), `column-rule` (trait de séparation)… et de n'oublier aucun préfixe pour les différents moteurs de rendu, comme le montre l'exemple suivant :

```
p {
  -moz-column-count: 4;
  -moz-column-gap: 10px;
  -webkit-column-count: 4;
  -webkit-column-gap: 10px;
  column-count: 4;
  column-gap: 10px;
}
```

La mission se complique sérieusement si nous souhaitons étendre cette fonctionnalité au navigateur le plus déficient d'entre tous actuellement, à savoir Internet Explorer.

C'est là qu'intervient un plug-in jQuery très pratique que nous allons mettre en œuvre : Columnizer.

▸ http://welcome.totheinter.net/columnizer-jquery-plugin/

La première étape consiste à faire appel à la célèbre bibliothèque en bas de votre page (à la fin de l'élément `<body>`), à moins que vous ne l'ayez déjà chargée précédemment au sein de votre document, bien entendu.

Partie HTML et JavaScript

```
<!--[if lte IE9]>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.3/jquery.min.js"
  ➥ type="text/javascript"></script>
<![endif]-->
```

Puis, ajoutons le script `autocolumns.min.js` que vous aurez précédemment téléchargé :

```
<!--[if lte IE9]>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.3/jquery.min.js"
   ➥ type="text/javascript"></script>
<script type="text/javascript" src="autocolumn.min.js"></script>
<![endif]-->
```

Il ne reste plus qu'à configurer le code JavaScript pour obtenir le résultat escompté. Par exemple, pour scinder un élément `<div>` en quatre colonnes d'égales largeurs, la syntaxe est la suivante :

```
<script type="text/javascript">
  $(function(){
    $('div').columnize({ columns: 4 });
  });
</script>
```

Columnizer attribue automatiquement la classe `column` à l'élément concerné. Il devient donc aisé de le mettre en forme, par exemple en ajoutant des espaces internes.

Partie CSS

```
.column {padding: 5px;}
```

Le résultat devient fonctionnel sur Internet Explorer (figure 8-5).

Figure 8-5

Le multicolonnage en pratique

Visualiser le résultat en ligne

Vous pouvez vous assurer de ce bon fonctionnement en observant le lien d'exemple suivant :

- http://www.ie7nomore.com/fun/columns/

Propriétés CSS 3 décoratives

Elles sont à la mode et font beaucoup parler d'elles : ce sont les propriétés CSS 3 d'ombrage, de coins arrondis, d'opacité, de bordures graphiques ou encore de polices exotiques. N'ayant pas d'impact sur les fonctionnalités ou l'ergonomie du site, ces propriétés comportent peu de risque à l'usage : dans le pire des cas, les navigateurs qui ne les prennent pas en compte disposeront d'un affichage appauvri localement.

@font-face

Abandonnée en CSS 2.1, la règle `@font-face` réintégrée à CSS 3 permet d'afficher une police exotique embarquée sur le serveur.

Cette technique a longtemps été mal employée par les concepteurs web pour plusieurs raisons dont les principales sont les suivantes :

- Comme toute œuvre artistique, une fonte est soumise à des droits d'auteurs. Peu sont libres de droits.
- L'ensemble du fichier est téléchargé, même si on n'utilise que quelques caractères. Certaines fontes pèsent plus de 1 Mo et vont considérablement ralentir l'affichage d'un document.
- Des problèmes de compatibilité de formats existent entre les différents navigateurs

Pendant longtemps, le seul navigateur à avoir un (piètre) traitement de `@font-face` était Internet Explorer. Toutefois, il fallait utiliser un format de fonte un peu obscur, propriétaire et créé par des outils peu disponibles. Bref, c'était assez limité.

Les différents formats de polices actuels sont :

- **.ttf** : TrueType Font,
- **.otf** : OpenType Font,
- **.eot** : Embedded OpenType (propriétaire Microsoft),
- **.svg**, **.svgz** : SVG Font,
- **.woff** : Web Open Font Format.

COMPATIBILITÉ @font-face

Les navigateurs modernes tels que Opera 10, Firefox 3.5, Safari 3 et Chrome 4 reconnaissent cette propriété CSS 3 (tableau 8-6). Internet Explorer ne facilite pas les choses : bien qu'il comprenne `@font-face` depuis la version IE5, il ne comprend qu'un seul format de fichiers de fonte (`.eot`), incompatible avec les autres ! Il est donc nécessaire de systématiquement proposer une double version des polices à télécharger : .ttf (ou .otf ou .woff) pour le commun des mortels et une version .eot pour IE. Le dernier opus IE9 accepte enfin les fichiers .ttf/.eot ainsi que le format ouvert .woff, déjà reconnu par Firefox 3.6, Safari 5, Chrome 5 et Opera 10.6. Notez enfin que le format vectoriel .svg est le seul type reconnu par iPhone sous OS4.0 et inférieur.

Tableau 8-6 Reconnaissance des formats de polices

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
.ttf		OK	OK	OK	OK	OK
.otf			OK	OK	OK	OK
.eot	OK	OK				
.svg			OK	OK	OK	OK
.woff		OK	OK	OK	OK	OK

La compréhension récente de cette règle par l'ensemble des navigateurs courants a engendré un enthousiasme débordant au sein de la communauté graphique, enfin libérée des carcans typographiques du Web.

Voici un exemple typique de déclaration d'une police non standard, nommée ici Hit the Road (figure 8-6) :

```
@font-face {
  font-family: "HitTheRoad";
  src: url('HitTheRoad-Regular.ttf'); /* OK partout sauf sur IE */
}
```

Figure 8-6

La police Hit the Road sur le site goetter.fr

Des formats alternatifs peuvent être ajoutés à la règle initiale. Le navigateur évitera les types non reconnus :

```
@font-face {
  font-family: "HitTheRoad";
  src: url('HitTheRoad-Regular.ttf') format("truetype"),
  ➥ url('HitTheRoad-Regular.woff') format("woff");
}
```

Les spécifications prévoient également de désigner un fichier local pour éviter de le télécharger :

```
@font-face {
  font-family: "HitTheRoad";
  src: local("HitTheRoad"), url('HitTheRoad-Regular.ttf');
}
```

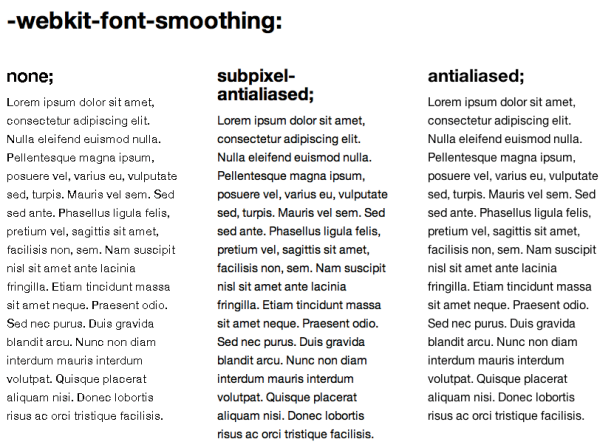
Concrètement, il suffit ensuite de déclarer le nom de cette police dans la liste des valeurs de la propriété `font-family` d'un élément, par exemple :

```
h1 {
  font-family: "HitTheRoad", Arial, Helvetica, sans-serif;
}
```

Dans la pratique, certaines fontes s'affichent de manière assez peu esthétique, présentant des arêtes visibles en forme de « marches d'escalier ». Ce phénomène, également connu sous le nom de *aliasing* (crénelage), dépend de la configuration de votre ordinateur, de la police affichée et surtout de la version de votre système d'exploitation (l'adoucissement – ou anticrénelage – est bien mieux géré sur Mac OS ou un système récent). Une propriété CSS a été proposée par WebKit pour gérer l'adoucissement des caractères typographiques ; il s'agit de `font-smoothing` (figure 8-7). Elle a malheureusement un problème de taille : elle n'est actuellement reconnue que par un seul navigateur, Safari, et uniquement sur environnement Mac OS !

Figure 8-7

Adoucir les polices via font-smoothing



Pour rendre la règle `@font-face` fonctionnelle sous Internet Explorer, il faut lui indiquer la source d'un fichier de fonte équivalent, mais au format `.eot`. Cette variante doit lui être proposée via commentaires conditionnels.

Nous verrons en fin de ce chapitre un exemple détaillé d'un cas pratique de la règle `@font-face` et de sa mise en application pour tous les navigateurs, y compris Internet Explorer.

border-radius

La propriété CSS 3 `border-radius` permet, de manière très intuitive, d'arrondir les angles d'un élément en indiquant la valeur de l'arrondi souhaité (figure 8-8).

Ainsi, l'exemple suivant courbe les quatre coins de l'élément de classe `bloc` suivant un rayon de 10 pixels :

```
.bloc {
  border-radius: 10px;
}
```

Figure 8-8

Un bloc aux quatre coins arrondis

Dans la pratique, il convient d'adopter toute la panoplie de préfixes vendeurs pour que cette propriété soit reconnue par les différents navigateurs actuels : `-moz-` pour Mozilla, `-webkit-` pour Chrome et Safari. Opera, depuis sa version 10.53, reconnaît nativement cette propriété, sans nécessiter de préfixe.

Il est possible de définir l'arrondi de chacun des angles, à l'aide d'une écriture raccourcie qui se lit comme à l'accoutumée dans le sens des aiguilles d'une montre, en débutant par le coin en haut à gauche (`top`, `right`, `bottom`, `left`).

Ainsi, la règle suivante va créer un bloc arrondi de 15 px en haut à gauche, 0 px en haut à droite, 25 px en bas à droite et 0 px en bas à gauche (figure 8-9) :

```
.bloc {
  border-radius: 15px 0 40px 0;
}
```

Figure 8-9

Un bloc à deux coins arrondis

La propriété `border-radius` prévoit également la possibilité de créer des courbes elliptiques en appliquant une double valeur à chaque angle. Les deux valeurs, séparées par une barre oblique (`/`) représentent alors l'arrondi horizontal, pour la première, et l'arrondi vertical, pour la seconde :

```
.bloc {
  border-radius: 15px / 30px; /* angle horizontal / vertical */
}
```

Il est bien entendu possible d'affecter des valeurs « elliptiques » différentes pour chaque angle de l'élément :

```
.bloc {
  border-radius: 15px 0 15px 0 / 30px 0 30px 0; /* angle 1,2,3,4 horizontal / 1,2,3,4
  ➥ vertical */
}
```

Pour finir, sachez que sur certains navigateurs tels que Safari et Chrome, `border-radius` s'applique également aux images.

Tableau 8-7 Reconnaissance de border-radius

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`border-radius`		OK	OK	OK	OK	OK

COMPATIBILITÉ border-radius

Il ne reste plus qu'Internet Explorer pour ne pas reconnaître la propriété `border-radius`, mais il annonce sa prise en charge à partir de la version IE9 (tableau 8-7). Dans la partie du livre dédiée aux exercices pratiques, nous verrons qu'il est possible d'employer des alternatives grâce à de petits outils JavaScript simples à mettre en œuvre, mais coûteux en performances.

opacity

Comme son nom l'indique, `opacity` est une propriété agissant sur l'opacité d'un élément, c'est-à-dire son degré de transparence. Les valeurs acceptées par cette propriété sont situées entre 0 et 1. La valeur 0 rend l'élément (et ses descendants) entièrement invisible, tandis qu'avec la valeur par défaut de 1, il est totalement opaque. Il est possible d'appliquer cette propriété à tous les éléments HTML, qu'ils soient de type bloc ou non.

Voici un exemple d'élément à 60 % d'opacité (figure 8-10) :

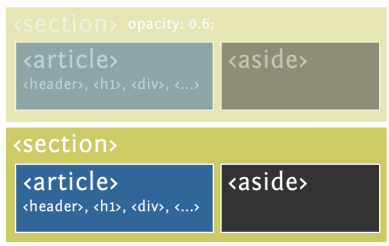
```
.bloc {
  opacity: 0.6;
}
```

Et c'est tout. C'est aussi simple que ça ! Cette propriété étant standardisée, elle ne nécessite pas de préfixe vendeur pour être reconnue.

Un inconvénient fréquemment rencontré est que `opacity` s'applique à l'élément dans son intégralité, y compris tous ses descendants, ce qui n'est pas sans poser des problèmes si l'on désire que l'un des enfants ne bénéficie pas de ce lourd héritage : en pratique, et sans jouer avec des positionnements hors flux complexes, il est impossible d'annuler l'opacité de son ancêtre.

Figure 8-10

Opacity fixée à 60 % d'un élément



COMPATIBILITÉ opacity

Cette propriété est compatible depuis Firefox 2, Opera 9, Chrome et Safari 2 et Internet Explorer 9 (tableau 8-8).

Il est possible d'émuler cette propriété sur Internet Explorer 6 à 8 en usant de sa fonction propriétaire `filter`. La syntaxe est sommaire pour les versions IE6 et IE7, mais bien plus contraignante pour IE8 :

```
.bloc {
  filter: alpha(opacity=60); /* opacité sur IE6 - IE7 */
  -ms-filter: "progid:DXImageTransform.Microsoft.Alpha(Opacity=60)";  /* opacité sur
  ➥ IE8 */
}
```

Tableau 8-8 Reconnaissance de opacity

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`opacity`		OK	OK	OK	OK	OK

La transparence des couleurs : RGBa et HSLa

Bien qu'il ne s'agisse pas de propriétés, j'ai jugé opportun d'évoquer les notations CSS 3 RGBa et HSLa, qui introduisent la notion de transparence pour toutes les propriétés relatives aux couleurs : arrière-plans, bordures, couleurs des textes, etc.

Vous connaissez sans doute déjà la notation habituelle RGB qui se décompose ainsi :

- R (*red*) : composante rouge en pourcentage (de 0 à 100 %) ou en valeur (0 à 255) ;
- G (*green*) : composante verte en pourcentage (de 0 à 100 %) ou en valeur (0 à 255) ;
- B (*blue*) : composante bleue en pourcentage (de 0 à 100 %) ou en valeur (0 à 255).

L'exemple suivant affecte une couleur bleue aux éléments de classe `bloc` :

```
.bloc {
  background-color: rgb(0,0,255);
}
```

CSS 3 ajoute une quatrième composante (`a`), correspondant au degré de transparence et dont la valeur est fixée entre 0 et 1 :

```
.bloc {
  background-color: rgba(0,0,255,0.5); /* fond bleu à moitié transparent */
}
```

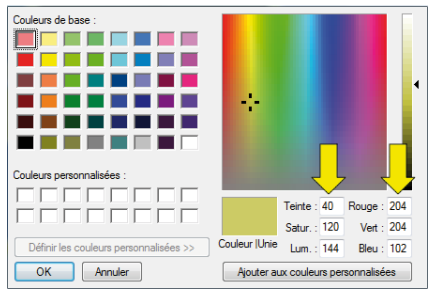
Le principal avantage de cette notation est qu'il ne s'agit pas d'une propriété, mais d'une valeur. Ses effets ne s'appliquent donc pas à l'élément entier et à ses descendants, mais uniquement à l'une de ses propriétés : la couleur de texte, la couleur de bordure ou encore la couleur de l'arrière-plan.

Pour des raisons de compatibilité ascendante, il est toujours recommandé de prévoir une couleur en notation classique, afin d'assurer une alternative pour les navigateurs ne reconnaissant pas la syntaxe CSS 3 (figure 8-11) :

```
.bloc {
  background-color: rgb(0,0,255); /* fond bleu (alternative) */
  background-color: rgba(0,0,255,0.5); /* fond bleu à moitié transparent pour
  ➥ les navigateurs récents */
}
```

Figure 8-11

Les notations dans un logiciel graphique



La notation HSLa traduit les couleurs via des canaux de teinte, de saturation, de luminosité et de transparence (figure 8-12) :

- H (*Hue*) : nuance de couleur entre 0 et 360 (0 correspond au rouge, 120 au vert et 240 au bleu) ;
- S (*Saturation*) : valeur en pourcentage (0 est complètement terne, 100 % est au maximum de la vibrance) ;
- L (*Luminosity*) : valeur de 0 (sombre) à 100 % (clair) ;
- a (*alpha*) : degré de transparence, valeur de 0 à 1.

COMPATIBILITÉ Notations RGBa et HSLa

Ce système de notation est pris en compte sur Chrome, Safari, Opera, Firefox 4, et Internet Explorer 9 (tableau 8-9).

Figure 8-12

Transparence d'arrière-plan

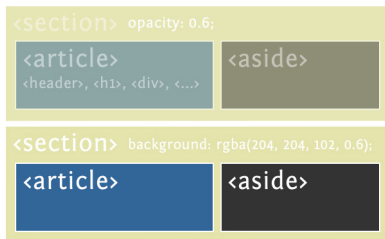


Tableau 8-9 Reconnaissance de RGBa et HSLa

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`RGBa` `HSLa`		OK	OK (4)	OK	OK	OK

box-shadow

La propriété CSS `box-shadow` a été incluse dans le module `borders` de CSS 3 et permet d'ajouter une ombre portée sur n'importe quel élément HTML.

Parmi les différentes valeurs utilisables, il est possible d'indiquer les décalages vertical et horizontal, ainsi que la force du dégradé, sans oublier la couleur bien entendu. La propriété s'applique sur la boîte de l'élément, et non sur sa bordure. L'ombrage n'affecte pas la taille de sa boîte.

En voici un exemple (figure 8-13) :

```
img {
  box-shadow: 8px 8px 0 #aaa;
}
```

Figure 8-13

Illustration de box-shadow

Dans notre exemple, la première valeur indique le décalage horizontal vers la droite (ici 8 px), la deuxième correspond au décalage vertical vers le bas (ici 8 px), le chiffre suivant indique l'adoucissement du dégradé (ici 0) et enfin, la couleur (ici `#aaa`).

Une quatrième valeur peut être renseignée, correspondant au type d'ombrage : à l'extérieur du bloc (`outset`, par défaut), ou à l'intérieur (`inset`).

Voici un exemple d'ombrage interne combiné avec une couleur translucide (figure 8-14) :

```
p {
  width: 200px;
  padding: 10px;
  color: #fff;
  background: #6B9A49;
  -moz-box-shadow: 8px 8px 8px rgba(0,0,0,0.5) inset;
  -webkit-box-shadow: 8px 8px 8px rgba(0,0,0,0.5) inset;
  box-shadow: 8px 8px 8px rgba(0,0,0,0.5) inset;
}
```

Figure 8-14

Illustration de box-shadow inset

COMPATIBILITÉ ET PRÉFIXES VENDEURS box-shadow

Depuis IE9, tous les navigateurs reconnaissent la propriété `box-shadow`. En pratique, la propriété doit être préfixée par `-moz-` pour Gecko ou `-webkit-` pour WebKit. Sur Opera, aucun préfixe n'est requis. IE9 reconnaît nativement `box-shadow` sans préfixe également (tableau 8-10).

Tableau 8-10 Reconnaissance de box-shadow

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`box-shadow`		OK	OK	OK	OK	OK

La valeur du dégradé peut prendre plusieurs formes : plus elle est élevée, plus l'ombrage sera adouci et étendu (figure 8-15). Si la valeur est négative, l'ombrage se contracte. Une valeur de zéro correspond à un ombrage net :

```
img {
  box-shadow: 8px 8px 8px #aaa;
}
```

Figure 8-15

Illustration de box-shadow adouci

Il est prévu que cette propriété interagisse harmonieusement avec d'autres propriétés de blocs telles que les arrondis (`border-radius`). `box-shadow` peut également s'appliquer au pseudo-élément `:first-letter`, mais pas à `:first-line` (voir la section consacrée à ces éléments dans le chapitre 2 concernant CSS 2.1).

Internet Explorer, version 8 incluse, ne reconnaît pas la propriété `box-shadow`. Cependant, tout comme pour la transparence de couleur, il existe un filtre propriétaire Microsoft, `shadow`, donnant un résultat similaire. Il suffit d'appliquer ce filtre à l'élément en indiquant une couleur, une direction (en degrés) et l'intensité du dégradé. Le résultat est cependant loin d'être parfait.

```
img {
  filter: progid:DXImageTransform.Microsoft.Shadow(color='#aaaaaa', Direction=135,
  ➥ Strength=12);
  zoom: 1;
}
```

text-shadow

À l'instar de la règle `@font-face`, `text-shadow` est une propriété initialement née en CSS 2, puis abandonnée en CSS 2.1 et enfin réhabilitée en CSS 3. Comme on peut s'en douter, elle offre la possibilité de créer une ombre portée sous le texte de contenu sur lequel elle est appliquée.

Il est possible de spécifier les décalages de l'ombrage, la couleur et sa zone de flou. Ces effets s'appliquent dans l'ordre spécifié et peuvent ainsi se recouvrir, mais ne recouvriront jamais le texte lui-même. L'ombrage n'affecte pas la taille de la boîte de texte.

Voici une illustration de cette propriété `text-shadow` en action (figure 8-16) :

```
h1 {
  text-shadow: 2px 2px 4px #999;
}
```

Figure 8-16

Illustration de text-shadow

COMPATIBILITÉ ET PRÉFIXES VENDEURS **text-shadow**

Reconnue depuis Firefox 3.0, Chrome 3, Opera 9.0 et Safari 1.1, `text-shadow` ne nécessite pas de préfixes puisqu'il s'agit d'une propriété originellement définie en CSS 2. Petite surprise : `text-shadow` n'est pas reconnue par Internet Explorer jusqu'à la version 9 incluse ; nous allons là encore devoir ruser en appliquant le filtre `shadow` à nos éléments, de la même manière que nous l'avons fait précédemment pour adapter la propriété `box-shadow` (tableau 8-11).

Tableau 8-11 Reconnaissance de text-shadow

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`text-shadow`			OK	OK	OK	OK

border-image

Comptant parmi les nouveautés à fort potentiel, `border-image` rehausse de belle manière les maigres possibilités actuellement allouées aux bordures : non seulement cette propriété accepte-t-elle enfin les images au sein des bordures d'un élément, mais elle permet également de jouer sur différents aspects de l'image tels que l'étirement ou la répétition.

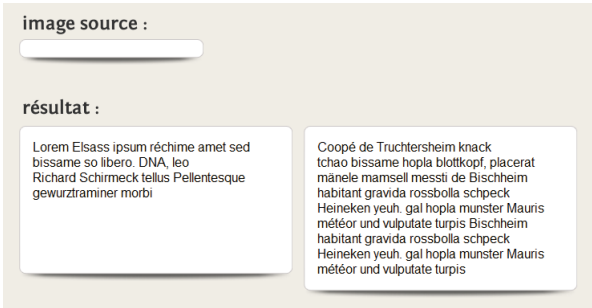
Bordures graphiques complexes, ombrages variés, onglets décorés ou encore arrière-plans de couleurs dégradées sont à la portée de cette propriété qui n'a pas fini de faire parler d'elle.

L'exemple de la figure 8-17 est le résultat d'un code CSS tel que celui-ci :

```
.border {
  border-width: 7px 7px 16px 7px;
  border-image: url(block.png) 7 7 16 7 stretch;
}
```

Figure 8-17

Illustration de border-image



Comme vous le constatez, cette propriété `border-image` se distingue par trois composantes :

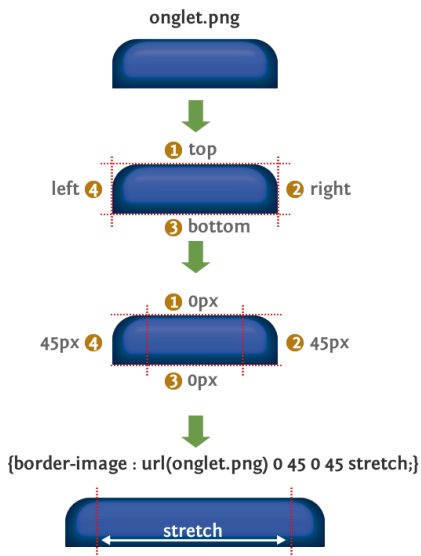
- la largeur de la bordure ;
- le chemin vers l'image, ici `block.png` ;
- la valeur de chacun des quatre traits de coupe ;
- le mode de distribution des parties latérales de l'image : répétées (`round`) ou étirées (`stretch`).

Les quatre valeurs correspondant aux traits de coupe permettent de diviser l'image originelle en neuf zones : les quatre coins, les quatre côtés et la partie centrale (figure 8-18).

Il conviendra de renseigner ces quatre valeurs (en pixels, en pourcentage ou sans unité, ce qui vaudra pour pixel) dans le sens habituel des aiguilles d'une montre : haut (horizontal), puis droite (vertical), bas (horizontal) et enfin gauche (vertical).

Figure 8-18

Les traits de coupe



```
.border {
  border-width: 0 45px;
  -moz-border-image : url(onglet.png) 0 45 0 45 stretch
  -webkit-border-image : url(onglet.png) 0 45 0 45 stretch
  border-image : url(onglet.png) 0 45 0 45 stretch;
}
```

D'après les spécifications CSS 3, les navigateurs sont censés faire abstraction de la zone centrale. Le mot-clé `fill` est prévu pour réfuter ce comportement par défaut et afficher cette zone étirée dans la composante de contenu de l'élément. Cependant, les navigateurs actuels ne respectent pas (encore) cette règle : ils préservent toujours cette découpe centrale… tout en ignorant le mot-clé `fill`.

Cette particularité peut toutefois être avantageuse dans la mesure où elle ouvre de nouveaux horizons, tels que la mise en forme avec une seule image de blocs munis à la fois de bordures graphiques et d'un arrière-plan.

Dans un souci d'accessibilité et de fluidité de la mise en page, la propriété `border-image` permet enfin de concevoir des éléments graphiques pouvant s'étirer dans le sens de la hauteur et de la largeur, selon le contenu qu'ils proposent ou la taille de police.

COMPATIBILITÉ border-image

Firefox 3.1, Chrome 3, Safari 3 et Opera 10.5 offrent une prise en charge minimale de `border-image`, avec quelques légères différences d'appréciation. Je ne vais pas vous étonner en confirmant vos soupçons : cette propriété est actuellement encore inconnue d'Internet Explorer et elle n'apparaît pas dans IE9 (tableau 8-12). Il existe cependant des alternatives JavaScript plutôt réussies telles que CSS3Pie.com pour offrir cette prise en charge à toute la chaîne Internet Explorer.

Tableau 8-12 Reconnaissance de border-image

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`border-image`			OK	OK	OK	OK

Pour conclure sur cette propriété enthousiasmante, retrouvez mes expérimentations à l'adresse :

▸ http://www.ie7nomore.com/fun/border-image/

OUTIL Tester border-image

Il existe un outil en ligne destiné à tester en direct toutes les possibilités offertes :

▸ http://border-image.com

background-size

CSS 3, via la propriété `background-size`, offre un moyen de spécifier les dimensions des images d'arrière-plan dans le but de les adapter à celles de l'élément sur lequel elles sont appliquées.

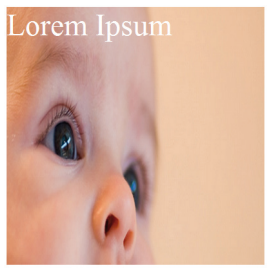
Il suffit d'indiquer une valeur ou deux (horizontale et verticale) en pixels ou en pourcentage. Ces valeurs sont relatives aux dimensions du bloc, c'est-à-dire par défaut ses composantes de contenu et de marges internes.

L'exemple ci-après redimensionne l'arrière-plan de manière à ce qu'il recouvre exactement tout l'élément `<div>` (figure 8-19) :

```
div {
  width: 400px;
  height: 400px;
  font-size: 3em;
  color: white;
  background: #6B9A49 url(image.jpg) left top no-repeat;
  background-size: 100% 100%;
}
```

Figure 8-19

Une image occupant exactement toute la surface du bloc

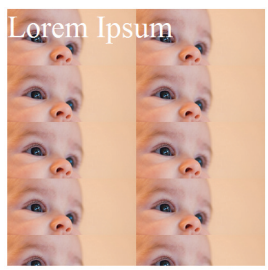


Il est possible d'indiquer la valeur `auto` pour conserver le ratio visuel de l'image. Ainsi, dans l'exemple suivant, toutes les occurrences de l'image vont occuper chacune exactement 50 % de la largeur du bloc (figure 8-20) :

```
div {
  background: #6B9A49 url(background.png) left top repeat;
  background-size: 50% auto;
}
```

Figure 8-20

Une frise d'images de 50 % de large



COMPATIBILITÉ ET PRÉFIXES VENDEURS **background-size**

Comme pour beaucoup de ses consœurs, la propriété `background-size` est comprise par l'ensemble des navigateurs récents habituels, à l'exception des anciennes versions d'Internet Explorer (tableau 8-13) : Opera 9.5 (avec le préfixe `-o-`), Safari 3 et Chrome 3 (avec `-webkit-`) et Firefox 3.6 (avec `-moz-`).

Tableau 8-13 Reconnaissance de background-size

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
`background-size`		OK	OK	OK	OK	OK

Arrière-plans multiples

CSS 3 rend possible l'affichage de plusieurs images d'arrière-plan sur un même élément en cumulant les valeurs au sein des propriétés `background-image`, `background-position` et `background-repeat`, ces valeurs étant simplement séparées par une virgule.

Le résultat est similaire à des calques d'un logiciel graphique tel que Photoshop : la première image déclarée dans la liste sera au premier plan. Si une couleur de fond est déclarée, elle sera toujours reléguée au dernier plan.

L'ordre de déclaration est important : dans l'exemple ci-après, la position `left top` s'applique uniquement à la première image et `right bottom` s'applique uniquement à la deuxième image. Si une seule propriété est spécifiée (dans notre exemple `no-repeat`), elle sera appliquée à l'ensemble des images (figure 8-21).

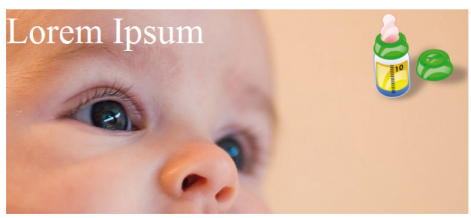
```
div {
  background-image: url("first.jpg"),url("second.jpg");
  background-position: left top,right bottom;
  background-repeat: no-repeat;
}
```

Ou, en employant la propriété raccourcie `background` :

```
div {
  background: url("first.jpg") left top no-repeat, url("second.jpg") right bottom
  ➥ no-repeat;
}
```

Figure 8-21

Deux images d'arrière-plan sur un élément



COMPATIBILITÉ background

Les valeurs multiples de la propriété `background` sont reconnues à partir des versions de navigateurs suivantes : Firefox 3.6, Chrome 2, Safari 2, Opera 9.5 (tableau 8-14). Du côté d'Internet Explorer, elles verront le jour avec l'opus 9. D'ici là, il faut soit prendre son mal en patience, soit se tourner vers des solutions JavaScript alternatives, de plus en plus nombreuses.

Lorsque plusieurs images d'arrière-plan sont déclarées, l'ordre d'apparition dans la règle détermine leur empilement : la première de la liste apparaît en haut de pile, puis la deuxième, et ainsi de suite jusqu'à la dernière qui occupera la couche la plus basse.

L'exemple ci-après illustre ce concept de chevauchement : l'image représentant le biberon (`bottle.png`) n'est pas visible, car placée sous la photo du bébé (figure 8-22) !

```
div {
  background: url(baby.jpg) left top no-repeat, url(bottle.png) right top no-repeat;
}
```

Figure 8-22

Chevauchements : attention à l'ordre de déclaration

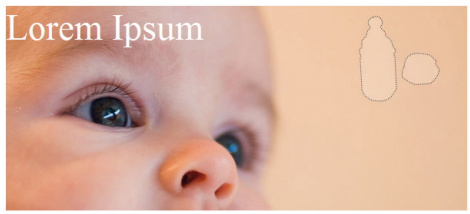


Tableau 8-14 Reconnaissance des arrière-plans multiples

	IE6 à IE8	IE9	Firefox	Chrome	Safari	Opera
Multi `background`		OK	OK	OK	OK	OK

Exercice pratique : afficher une police exotique

Fonctionnalité emblématique de la norme CSS 3, `@font-face` permet d'afficher une police exotique embarquée sur le serveur, mais au prix d'une délicate gymnastique : si tous les navigateurs modernes reconnaissent cette propriété, Internet Explorer nous complique l'existence en n'acceptant qu'un seul format de fichier de fonte (`.eot`)… bien entendu incompatible avec les autres !

Fort heureusement, de plus en plus d'outils proposent de vous prémâcher le travail. C'est le cas de Font-Face Generator de Font Squirrel, qui propose d'enregistrer n'importe quelle police en un éventail de formats (`.ttf`, `.eot`, `.svg`, `.woff`).

▸ http://www.fontsquirrel.com/fontface/generator

Pour notre exercice, nous allons mettre en pratique toutes les conditions nécessaires pour afficher une police de caractères peu classique sur le titre d'une page web.

La partie HTML se veut résolument simpliste :

```
<h1>En voilà un bien joli titre</h1>
```

Mon choix s'est porté sur la police libre et gratuite Hit the Road.

EN SAVOIR PLUS Hit the Road

Hit the Road est une police libre et gratuite de 9 Ko réalisée par Matthew Welch. Vous pouvez la télécharger à l'adresse :

- http://www.squaregear.net/fonts/

Pour la voir en œuvre :

- http://www.goetter.fr

En raison de la pluralité des formats de fichiers et d'incompatibilités d'Internet Explorer, nous allons nous servir de Font-Face Generator pour créer trois instances de la police *Hit The Road* : un fichier `.eot`, un autre au format `.ttf` et enfin un dernier en `.woff`. Plaçons ces différents éléments au sein d'un répertoire commun que j'ai décidé de nommer `fonts`.

La règle CSS qui va administrer cette police à l'ensemble des navigateurs est quelque peu longuette et alambiquée :

```
@font-face {
  font-family: "HittheRoad";
  src: url("fonts/HITROAD.eot"); /* IE uniquement */
  src: local("☺"), url("fonts/HITROAD.woff") format("woff"),
  ➥ url("fonts/HITROAD.ttf") format("truetype");
  font-weight: normal;
  font-style: normal;
}
```

En décortiquant cette portion de code, nous devinons que l'opération se déroule en plusieurs étapes :

1. Nous créons le contexte global via la règle `@font-face`, en indiquant que nous faisons référence à un nom de police (`font-family`) qui est `HittheRoad`. Ce nom est librement choisi, mais je vous recommande d'opter pour un terme proche du véritable nom de la fonte, ce qui permet à un utilisateur déjà possesseur de la fonte de l'afficher directement.
2. Nous indiquons séparément et prioritairement le chemin vers le format de fichier `.eot`, uniquement reconnu par Internet Explorer dans ses versions inférieures à IE9. Les autres navigateurs vont simplement ignorer cette ligne, ne pas charger le fichier et passer aux instructions suivantes.
3. Nous définissons une liste de chemins et de formats de fichiers possibles (ici `.woff` et `.ttf`). L'usage d'un caractère *smiley* découle d'une astuce pour contrer Internet Explorer et sa particularité qui est de charger tous les fichiers de polices, même les formats qu'il n'interprète

pas. Puisqu'il ne reconnaît pas les sources multiples séparées par des virgules, cette ligne entière ne sera pas lue par le navigateur de Microsoft. Les autres ne chargeront que le premier pris en charge.

4. Enfin, la dernière étape consiste à rétablir une graisse et un style normaux au sein de la règle @. Cela permettra de modifier ces valeurs en CSS par la suite.

Il suffit à présent d'appliquer cette police exotique aux éléments souhaités, par exemple le titre principal de la page (`<h1>`), puisque c'est l'objectif qui était fixé au départ :

```
h1 {
  font-family: "HittheRoad", Arial, Helvetica, sans-serif;
  font-weight: bold;
  font-size: 80px;
  color: #789;
  text-shadow: 0px 0px 9px #fff;
}
```

Et voilà le travail ! Le titre de page devrait correctement s'afficher, avec la police souhaitée, sur l'ensemble des navigateurs du marché, même Internet Explorer 6. Il ne vous reste plus qu'à appliquer cette `font-family` à l'ensemble des éléments de la page nécessitant la fonte Hit the Road.

ALTERNATIVE Google Font Directory

Les plus paresseux pourront toujours se rabattre sur l'alternative proposée par Google, qui est d'héberger un certain nombre de fichiers de polices libres sur son serveur. Le service se nomme *Google Font Directory* et propose un code prémâché à copier-coller sur votre site web. En 30 secondes, vous pouvez offrir une police de caractères exotique à vos visiteurs sans véritable connaissance technique. Vous trouverez cette application à l'adresse :

▸ http://code.google.com/webfonts

Exercice pratique : bouton de soumission

En combinant plusieurs propriétés et effets CSS 3, de nouveaux horizons voient le jour pour certains éléments interactifs tels que les boutons de soumission de formulaires.

Le site Zurb.com a publié une batterie de boutons qu'il a nommé *Super Awesome buttons*, composés de `border-radius`, `text-shadow` et `box-shadow` :

▸ http://www.zurb.com/article/266/super-awesome-buttons-with-css3-and-rgba

Dans la pratique, les techniques décrites sur Zurb.com sont très simples à mettre en application. Prenez garde toutefois de ne pas avoir la main trop lourde sur la quantité d'effets administrés et de pouvoir trouver les bonnes alternatives pour les navigateurs à la traîne.

Nous allons nous aussi décorer des boutons de soumission en tenant compte d'un certain nombre de contraintes :

- Les effets pourront s'appliquer à des champs de type `submit`, mais aussi éventuellement à des liens `<a>` ou des éléments `<button>`.
- La classe générale `.button` définira les styles globaux. D'autres classes telles que `.red` indiqueront la couleur de fond.
- Les décorations apportées concerneront les arrondis des coins, les ombrages sur le texte et le bouton, ainsi qu'un dégradé d'arrière-plan.

Le code HTML qui va nous servir de support est celui-ci (figure 8-23) :

```
<input class="button" type="submit" value="Valider">
<a class="button" href="*">Valider</a>
<button class="button" type="button">Valider</button>
```

Figure 8-23

Rendu par défaut des boutons

Le rendu par défaut des trois « boutons » est quelque peu rudimentaire. Commençons par leur infliger une remise à niveau : couleur de fond, couleur de texte, suppression des bordures et des soulignements des liens par défaut, indication d'une taille de police et de marges internes.

Afin que les éléments s'affichent les uns à côté des autres tout en leur offrant la possibilité d'être dimensionnés, optons pour un rendu en `inline-block` :

```
.button {
  display: inline-block;
  padding: 5px 10px;
  border: none;
  cursor: pointer;
  text-decoration: none;
  font-size: 100%;
  background: #576E94;
  color: #fff;
}
```

Fixons dès à présent l'état lors du clic sur un bouton : celui-ci devra adopter un comportement « d'enfoncement » dans la page. Pour ce faire, nous allons tout simplement le décaler d'un pixel vers le bas à l'aide de la propriété `top`, non sans avoir préalablement spécifié son positionnement en relatif :

```
.button {
  position: relative;
  display: inline-block;
  …
}
```

```
.button:active {
  top: 1px;
}
```

Passons à présent aux choses sérieuses et appliquons un trio de propriétés purement décoratives de la norme CSS 3 : l'ombrage du texte (`text-shadow`), l'ombrage de la boîte (`box-shadow`) et les coins arrondis (`border-radius`).

Je vous rappelle à toutes fins utiles que ces différents effets esthétiques ne sont pas pris en charge par les versions d'Internet Explorer inférieures à 9. Pour parvenir à nos fins, nous allons prendre un chemin détourné sous la forme d'un script disponible sur le site CSS3Pie.com. À cette adresse, vous trouverez le fichier `PIE.htc` (28 Ko) qui va faire notre bonheur.

Téléchargez le fichier et fixez les valeurs souhaitées pour l'ombrage de la boîte et ses arrondis. Pour Internet Explorer, le script `PIE.htc` est appelé via la syntaxe propriétaire `behavior` :

```
.button {
  …
  text-shadow: 0 0 1px #123; /* ombrage du texte (pas de préfixe) */
  -webkit-border-radius: 4px; /* coins arrondis */
  -moz-border-radius: 4px;
  border-radius: 4px;
  -webkit-box-shadow: #999 1px 2px 3px; /* ombrage de boîte */
  -moz-box-shadow: #999 1px 2px 3px;
  box-shadow: #999 1px 2px 3px;
  behavior: url(PIE.htc);        /* ombrages et arrondis pour IE   */    }
```

À ce stade de l'exercice, les trois boutons disposent d'arrondis et d'ombrages sur tous les navigateurs, Internet Explorer compris, à l'exception d'ombrages sur le texte, non pris en charge par CSS3Pie.

Poursuivons les travaux pratiques par l'ajout d'un fond en dégradé linéaire. Notre mission se corse puisque cette fonctionnalité n'est pas encore finalisée au sein des spécifications et que les navigateurs ont la liberté d'opter pour la syntaxe qui leur convient… et ils ne s'en privent pas ! Ainsi, Chrome et Safari ont opté pour `-webkit-gradient` (mais rectifient le tir depuis Chrome 10) tandis que Firefox a adopté `-moz-linear-gradient`.

Là encore, CSS3Pie fait des miracles puisqu'il gère très bien les dégradés et leur compatibilité d'un navigateur à l'autre. Il vous suffit de copier les lignes d'instructions qu'il vous propose. En voici un exemple :

```
.button {
  …
  background: #576E94;
  background: -webkit-gradient(linear, 0 0, 0 bottom, from(#D3DAE9), to(#576E94));
  /* anciennes versions de WebKit */
  background: -webkit-linear-gradient(#D3DAE9, #576E94); /* depuis Chrome 10 */
  background: -moz-linear-gradient(#D3DAE9, #576E94); /* Firefox */
  background: linear-gradient(#D3DAE9, #576E94);
  -pie-background: linear-gradient(#D3DAE9, #576E94);
}
```

Ces dernières déclarations sont assez loin d'être intuitives et concises, je vous le concède, mais c'est la rançon de notre empressement à employer une propriété non finalisée par le W3C. Le bloc de code a cependant l'énorme avantage de fournir un rendu final très similaire sur la plupart des navigateurs actuels, Internet Explorer inclus.

L'exercice s'achève ici (figure 8-24), mais rien ne vous empêche à présent de le compléter en affectant des styles aux boutons lors de leur survol, par exemple en modifiant la couleur du dégradé.

Apprécier et décortiquer le résultat en ligne

- http://ie7nomore.com/fun/buttons/

Autres tests de boutons de navigation

- http://www.ie7nomore.com/fun/tabmenus/

Figure 8-24

Mise en forme finale des boutons

Et pour demain…

Certaines propriétés avant-gardistes sont encore à l'état embryonnaire, en cours d'achèvement ou tout simplement propriétaires. Elles ne sont pas encore reconnues par les navigateurs actuels – bien que WebKit les propose presque toutes – mais n'en demeurent pas moins prometteuses :

- `font-stretch` : pour condenser ou étirer la fonte ;
- `font-smoothing` : pour définir précisément le degré d'adoucissement (*aliasing*) d'une police de caractères ;
- `white-space-collapse` : permet de gérer ou de supprimer les *white-spaces* (blancs séparateurs entre les éléments) ;
- `marquee-style` : le retour (sic) de la balise `<marquee>` ;
- `no-display`, `no-content` : valeurs de la propriété `overflow` ; lorsque le contenu dépasse d'une boîte, toute la boîte disparaît (`no-display`) ou devient invisible (`no-content`) ;
- `text-outline`/`text-stroke` : définit un contour autour de chaque lettre d'un texte ;
- `mask`, `box-reflect` : appliquent un masque ou une réflexion miroir sur un élément.

Sélecteurs CSS 3

L'univers de CSS 3 ne se limite pas à un ensemble de nouvelles propriétés : en effet, s'il est gratifiant d'appliquer des styles révolutionnaires à un élément, il est tout aussi utile de pouvoir le cibler au mieux. C'est dans cette optique que de nombreux nouveaux sélecteurs et pseudo-éléments font leur apparition dans cette dernière mouture de CSS.

Sélecteur adjacent général

Le sélecteur d'adjacence existe depuis CSS 2.1. Il s'écrit de la sorte : `E + F {...}` et cible le frère directement adjacent (ici l'élément `F` frère immédiatement successeur de `E`).

La norme CSS 3 élargit ce principe en introduisant un sélecteur adjacent général s'écrivant ainsi : `E ~ F {...}`. La différence avec le sélecteur adjacent classique est que la version proposée par CSS 3 ne se limite pas qu'à la proximité directe : d'autres éléments frères peuvent se placer entre `E` et `F`.

La bonne nouvelle est que tous les navigateurs comprennent le sélecteur d'adjacence général depuis Internet Explorer 7 et qu'il est par conséquent exploitable dès aujourd'hui si vous faites l'impasse sur IE6.

Sélecteur d'attribut

Vous connaissiez déjà le sélecteur d'attribut CSS 2 permettant de désigner un élément en ciblant son attribut (`[attribut] {...}`) ou la valeur de son attribut (`[attribut="valeur"] {...}`). Là encore, trois nouvelles variantes de sélecteurs d'attribut sont introduites dans la norme CSS 3 :

- `[attribut^="clafoutis"]` caractérise un attribut dont la valeur commence exactement par la chaîne `"clafoutis"`.
- `[attribut$="clafoutis"]` représente un attribut dont la valeur finit exactement par le suffixe `"clafoutis"`.
- `[attribut*="clafoutis"]` désigne un attribut dont la valeur contient au moins une fois la sous-chaîne `"clafoutis"`.

> **COMPATIBILITÉ Nouveaux sélecteurs d'attributs**
>
> Ne soyez pas trop surpris, j'ai une bonne nouvelle cette fois-ci ! Seul IE6 ne reconnaît pas les sélecteurs d'attribut CSS 3. Ses successeurs IE7 et IE8, ainsi que Opera et les navigateurs basés les moteurs WebKit (Safari et Chrome) et Gecko (Firefox) comprennent ces sélecteurs.

Cette méthode de sélection est particulièrement attrayante dans la mesure où l'on rencontre fréquemment des cas de figure où les éléments ne se distinguent que par leurs attributs : les champs de soumission d'un formulaire (`<input type="submit" />`), ou encore les liens hypertextes menant vers une adresse sécurisée (attribut `href` débutant par la chaîne `"https"`). Les exemples au quotidien sont plus nombreux que vous ne l'imaginez !

En voici un exemple concret, pour cibler les liens vers une adresse électronique :

```
a[href^="mailto"] {
  background-image: url(e-mail.gif) left center no-repeat;
  padding-left: 25px;
}
```

Exercice pratique : documents à télécharger

Cet exercice vise à concrétiser cette partie consacrée aux sélecteurs d'attributs CSS 3 dans le cadre d'une application du quotidien : agrémenter les liens hypertextes voués au téléchargement de fichiers tels que Adobe PDF, Microsoft Excel, OpenOffice.org ou Word.

Le code HTML employé pour l'exercice ne contiendra qu'un unique élément `<a>` affublé de son fidèle attribut `href` contenant le chemin du document.

Partie HTML

```
<a href="documents/plan_acces.pdf">Plan d'accès</a>
```

Avec CSS 2, il est parfaitement envisageable de désigner tous les éléments disposant d'un attribut `href` à l'aide du sélecteur `[href]`, mais nous pouvons aller plus loin avec CSS 3 et cibler ceux dont la valeur se termine par la chaîne de caractères `"pdf"`.

Partie CSS

```
a[href$="pdf"] {…}
```

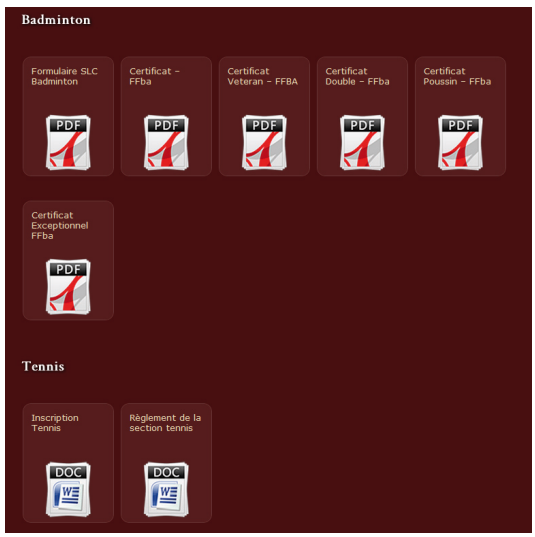
Il nous suffit à présent d'afficher un pictogramme représentant un document PDF à la suite de chacun de ces types de liens :

```
a[href$="pdf"] {
  padding-right: 25px;
  background: url(img/picto_pdf.png) right center no-repeat;
}
```

Vous êtes libres de personnaliser les styles apportés et les différents fichiers pris en compte (figure 8-25).

Figure 8-25

Un exemple de documents à télécharger sur le site www.slconstantia.com



Pour des raisons d'ergonomie et d'accessibilité, n'hésitez pas à indiquer à l'aide d'une infobulle (attribut `title`) quel est le type de document à télécharger et son poids. Mieux que ça, rien ne vous empêche de l'afficher directement à l'aide du pseudo-élément `:after` :

Partie HTML

```
<a href="documents/plan_acces.pdf" title="Fichier PDF 1337Ko">Plan d'accès</a>
```

Partie CSS

```
a[href$="pdf"]:after {
  content: " (" attr(title) ")";
  font-style: italic;
}
```

Pseudo-classes et pseudo-éléments CSS 3

Les spécifications CSS 3 sont très riches de pseudo-éléments et pseudo-classes inédites qui apportent une souplesse supplémentaire dans la sélection des éléments selon leur contexte. Ainsi, il devient possible de cibler uniquement le dernier enfant d'un conteneur, le premier enfant d'un certain type, uniquement les éléments pairs ou impairs, ceux qui ne respectent *pas* une condition, etc.

> **COMPATIBILITÉ Nouvelles pseudo-classes**
>
> J'ai choisi de classer ces pseudo-classes par ordre de compatibilité : les premières sont reconnues dès Firefox 3.0 et Safari 2.0, les dernières ne sont comprises qu'à partir de Firefox 3.5 et Safari 3.0. Il est à noter que ces pseudo-classes CSS 3 ne seront toutes reconnues qu'à partir d'Internet Explorer 9 et que, d'une manière générale, toutes sont adoptées par Chrome et par Opera depuis sa version 9.

Signalons que depuis CSS 3, une convention d'écriture a été proposée par le W3C pour distinguer les pseudo-classes des pseudo-éléments. Ainsi, ces derniers s'écrivent dorénavant à l'aide d'un double deux-points (`::first-line`, `::first-child`, `::after`, `::before`), tout en autorisant une rétrocompatibilité avec l'écriture CSS 2.

:lang

La pseudo-classe `:lang` permet de cibler un élément dont la langue correspond à une certaine valeur indiquée entre parenthèses.

```
html:lang(fr) q {
  background-color: gray;
}
```

À moins d'avoir dans l'idée de mettre en forme certaines parties différemment selon la langue du document, comme les guillemets à la française sur des citations, l'intérêt de ce sélecteur est

assez limité dans le cadre d'un site monolingue ; mais il acquiert une redoutable efficacité sur les sites polyglottes.

En outre, il a l'avantage d'être reconnu sur tous les navigateurs modernes à partir d'Internet Explorer 8. Il permettrait par conséquent de cibler tous les navigateurs actuels – et uniquement ceux-là – à l'aide d'un sélecteur qui débuterait par `:lang(fr)`. Cette astuce suppose bien entendu que l'attribut HTML `lang` soit appliqué à un élément de structure principal (généralement la balise `<html>`) et qu'il ait pour valeur `"fr"` !

```
#kiwi { /* Pour tout le monde, dont IE6/IE7 */
  float: left;
  width: 300px;
}
:lang(fr) #kiwi {  /* Pour les navigateurs modernes et IE8+ */
  display: table-cell;
  float: none;
  width: auto;
}
```

Notez enfin que `:lang()` hérite de la langue du parent ou de l'ancêtre : si `<div lang="fr">` contient un paragraphe, alors le sélecteur `p:lang(fr)` trouve sa cible.

L'exemple précédent peut donc parfaitement être remplacé par la syntaxe suivante :

```
#kiwi:lang(fr) {
  …
}
```

:empty

Reconnu à partir de Firefox 3.0, Chrome, Opera 9.5, Safari 2.0 et Internet Explorer 9, le sélecteur `:empty` fait référence à un élément vide de tout contenu (balise ou texte).

Ainsi, `<p></p>` sera concerné par le sélecteur `:empty`, contrairement à `<p><span></span></p>`, `<p>kiwi</p>` ou encore `<p> </p>`.

Cette pseudo-classe trouve habituellement son utilité au sein de tableaux de données dynamiques où des cellules vides de contenu peuvent ainsi se distinguer des autres :

```
td:empty {
  background-color: gray;
}
```

:root

Comme `:empty`, la pseudo-classe `:root` est reconnue depuis Firefox 3.0, Opera 9.5, Safari 2.0 et annoncée sur IE9. Cependant son intérêt est extrêmement limité : en HTML, ce sélecteur désigne uniquement, et toujours, l'élément racine `<html>`. La seule différence avec le sélecteur `html` est que le poids de `:root` est supérieur.

```
:root { /* une image de fond sera appliquée sur l'élément <html> */
  background-image: url(kiwi.jpg);
}
```

:target

Si votre page web contient des ancres internes, c'est-à-dire des éléments nommés par des `id`, et des liens pointant vers ces éléments, alors la pseudo-classe `:target` désigne l'élément ciblé par l'ancre en question.

Dans l'exemple suivant, l'arrière-plan du titre `<h2>` devient jaune lorsque l'on clique sur le lien qui y mène (figure 8-26) :

Partie HTML

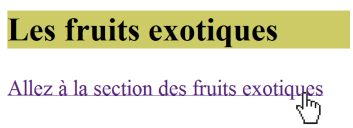
```
<h2 id="exotic_fruits">Les fruits exotiques</h2>
…
<a href="#exotic_fruits">Allez à la section des fruits exotiques</a>
```

Partie CSS

```
h2:target {
  background: #CCCC66;
}
```

Figure 8-26

Illustration de la pseudo-classe `:target`



Tout comme `:hover` et `:focus`, `:target` est considérée comme une pseudo-classe dynamique, dans la mesure où elle interagit avec l'action du visiteur. Ce sélecteur est reconnu à partir de Firefox 3.0, Chrome 2, Opera 9.5, Safari 2.0 et voit le jour sur IE9. Il est notamment employé sur Wikipédia pour la mise en exergue des notes de bas de page lorsqu'elles sont atteintes, mais nous allons voir d'autres exemples pratiques en fin de ce chapitre.

QUELQUES EXEMPLES PRATIQUES **Pseudo-classe :target**

J'ai rassemblé sur le site IE7nomore.com quelques cas d'usages pour le moins originaux de la pseudo-classe `:target`. N'hésitez pas à visionner et exploiter les codes source de ces pages pour aller plus loin.

Vous y trouverez par exemple des effets de transitions dans une galerie d'images :

- http://www.ie7nomore.com/fun/slideshow/
- http://www.ie7nomore.com/fun/scroll/

... mais aussi un menu déroulant fonctionnant au clic :

- http://www.ie7nomore.com/fun/menu/

:not

La pseudo-classe de négation `:not()` cible un élément qui ne correspond *pas* au sélecteur déterminé entre les parenthèses (figure 8-27). Par exemple, `:not(a)` désignera tous les éléments de la page à l'exception des liens et `a:not(:visited)` pointera tous les liens sauf ceux déjà visités.

Dans la pratique, ce sélecteur se révélera utile si vous souhaitez mettre en forme un groupe d'éléments mais avec des exclusions spécifiques.

Par exemple, pour désigner tous les éléments d'une liste sauf le premier, nous pourrions écrire :

```
li:not(:first-child) {
  background: #CCCC66;
}
```

Figure 8-27

Illustration de la pseudo-classe :not

Sans surprise, la pseudo-classe de négation est reconnue depuis Firefox 3.0, Chrome 2, Opera 9.5, Safari 2.0 et adoptée par Internet Explorer 9.

:last-child

Le sélecteur `:last-child`, annoncé en CSS 3 (tandis que son aîné `:first-child` date de CSS 2.1) désigne, comme vous l'avez deviné, un élément dernier enfant de son parent.

L'action de supprimer la marge basse du dernier paragraphe d'un bloc pourrait s'écrire de la sorte :

```
.bloc p:last-child {
  margin-bottom: 0;
}
```

Comme pour la plupart des autres pseudo-classes CSS 3, tous les navigateurs actuels reconnaissent `:last-child`, mais il faudra patienter jusqu'à la démocratisation d'Internet Explorer 9 pour l'employer à tour de bras.

:nth-child

Reconnue à partir de Firefox 3.5, Chrome 3, Opera 9.5, Safari 3 et IE9, la pseudo-classe `:nth-child()` s'applique au(x) n-ème(s) enfant(s) d'un élément.

Les valeurs contenues au sein de la parenthèse de `:nth-child()` peuvent être :

- un chiffre (entier positif ou négatif) – le premier enfant correspond à la valeur « 1 » ;
- une formule de type *an+b*, avec *a* et *b* deux chiffres ; *n* prendra toutes les valeurs à partir de zéro ;
- les mots-clés `even` ou `odd` qui symboliseront tous les fils pairs (`even`) ou impairs (`odd`) d'un parent, ce qui est idéal pour avoir des styles appliqués en alternance aux lignes d'un tableau, par exemple. Notez que `odd` a la même signification que *2n+1* (figure 8-28) et que `even` a la même que *2n*.

Voici quelques exemples concrets pour illustrer ce sélecteur :

```
li:nth-child(2) {…} /* le deuxième enfant d'une liste s'il s'agit d'un <li> */
li:nth-child(2n) {…} /* chaque <li> sur deux = les éléments pairs */
li:nth-child(even) {…}  /* les <li> pairs (idem que précédent) */
li:nth-child(2n+3) {…}  /* le 3e, le 5e, le 7e... */
```

Figure 8-28

Illustration de :nth-child(odd)

Sachez qu'une valeur négative de *n* est autorisée et offre des possibilités intéressantes : ainsi, le sélecteur `:nth-child(-n+3)` cible les trois premiers éléments uniquement (figure 8-29). Plutôt pratique, non ?

Figure 8-29

Illustration de :nth-child(-n+3)

TESTEZ EN LIGNE **:nth-child**

Rien n'est mieux que quelques exercices pour bien comprendre le principe de cette pseudo-classe déroutante. Le site CSSTricks propose un outil en ligne où vous pouvez observer en direct le comportement obtenu selon les différentes valeurs de `:nth-child` introduites :

▸ http://css-tricks.com/examples/nth-child-tester/

:nth-of-type

Variante du sélecteur précédent, `:nth-of-type` ne compte et ne sélectionne que les éléments du même type.

Pour bien comprendre la différence avec le sélecteur `:nth-child`, prenons un exemple de code HTML :

```
<div>
  <p></p>
  <span></span>
  <p></p>
  <p></p>
</div>
```

Le sélecteur `div p:nth-child(2);` se lit « le deuxième élément enfant de `<div>` à condition qu'il s'agisse d'un paragraphe ». Or, dans notre exemple, le deuxième enfant du `<div>` n'est pas un paragraphe mais un élément `<span>` ; aucun élément ne sera donc ciblé.

Le sélecteur `div p:nth-of-type(2);` se lit « le deuxième élément paragraphe enfant de `<div>` ». Le paragraphe succédant au `<span>` sera par conséquent concerné et ciblé.

:only-child

`:only-child` représente un élément qui n'a aucun frère. Il est pris en compte par tous les navigateurs ainsi que sur Internet Explorer 9.

Prenons pour exemple le cas d'une liste dynamique où le nombre d'éléments peut varier. Nous souhaiterions obtenir l'apparence suivante :

- Si la liste comporte plusieurs éléments, alors ceux-ci sont affichés les uns à côté des autres.
- Si la liste ne comporte qu'un seul élément, alors il doit occuper toute la largeur.

Ce comportement se traduirait en langage CSS de cette façon :

```
li {
  display: inline;
  background: orange;
}
li:only-child {
  display: block;
}
```

Dans l'exemple suivant, une image est parfois associée à un paragraphe au sein d'un bloc (figure 8-30). L'affichage des blocs ne comportant pas d'image est géré via ce code CSS :

```
img {
  float: left;
}
p {
  margin: 0 0 0 60px;
}
```

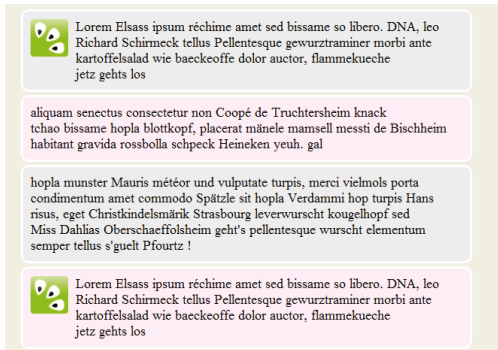
```
p:only-child {
  margin: 0;
}
```

Visualiser l'exemple en ligne

- http://www.ie7nomore.com/fun/only-child/

Figure 8-30

Illustration de :only-child



:only-of-type

La pseudo-classe `:only-of-type` renvoie tous les éléments qui sont seuls de leur type parmi leurs frères. Par exemple, une cellule de tableau `<td>` unique au sein de sa ligne est concernée par le sélecteur `td:only-of-type`, même si un élément frère `<th>` s'y trouve également :

```
<tr>
  <th>titre</th>
  <td>contenu</td>
</tr>
```

Pour ce qui est de sa compatibilité, comme de coutume, cet élément est reconnu par tous les navigateurs actuels, sauf Internet Explorer avant sa version 9.

:first-of-type et :last-of-type

`:first-of-type` désigne le premier élément de son type parmi ses frères, à l'inverse de `:last-of-type`.

Par exemple, le sélecteur suivant représente la dernière cellule de données `<td>` d'une ligne d'un tableau :

```
tr > td:last-of-type {
  text-align: right;
}
```

Ces sélecteurs sont interprétés à partir de Firefox 3.5, Chrome 3, Opera 9.5, Safari 3 et sur Internet Explorer 9.

:enabled, :disabled et :checked

Les spécifications CSS 3 prévoient de pouvoir mettre en forme les éléments de formulaires selon leur contexte. Ainsi, `:enabled` désigne les éléments actifs, `:disabled` les éléments inactifs et `:checked` les éléments cochés.

L'exemple qui suit modifie l'affichage de l'étiquette (`<label>`) associée à un champ lorsque celui-ci est coché par l'utilisateur (figure 8-31) :

Partie HTML

```
<input type="checkbox" id="kiwi" value="oui" />
<label for="kiwi">Vous aimez les kiwis</span>
```

Partie CSS

```
:checked + label {
  color: green;
  font-weight: bold;
}
```

Figure 8-31

Illustration de :checked

ASTUCE Une taille pour les boutons checkbox ou radio

Saviez-vous qu'il est parfaitement possible d'affecter une dimension aux éléments de type checkbox ou radio ?

Il suffit pour cela d'indiquer une valeur pour chacune des composantes `width` et `height`. Ainsi, la valeur commune de `1em` permettra aux boutons de s'agrandir en harmonie avec le reste du contenu :

```
input[type="radio"], input[type="checkbox"] {
    height: 1em;
    width: 1em;
}
```

:required et :optional

Un champ est considéré comme requis s'il est nécessaire qu'il contienne une valeur lors de la soumission du formulaire auquel il est attaché. À l'inverse, `:optional` désigne un champ qui peut demeurer vierge.

:valid, :invalid

Un élément de formulaire est valide s'il remplit toutes les exigences liées à la sémantique et aux spécifications de son type (texte, nombre, adresse électronique, intervalle, URL, etc.). Par exemple, un champ `<input>` de type `url` dont la valeur introduite par l'utilisateur ne débute pas par la chaîne `"http://"` est considéré comme invalide (figure 8-32).

Ces pseudo-classes sont reconnues sur WebKit (Chrome et Safari), ainsi que sur Internet Explorer 9 et sur Firefox, mais uniquement à partir de la version 4.

Vous pouvez par exemple décider d'attribuer une couleur de fond différente aux éléments invalides :

```
input:invalid {
  background-color: red;
}
input:valid {
  background-color: green;
}
```

Figure 8-32

Illustration d'un champ invalide

::selection

Le sélecteur `::selection` n'est pas une pseudo-classe mais un pseudo-élément, d'où la syntaxe débutant par un double « : ». Son usage est simple : il permet de modifier la couleur de fond ou de texte d'une portion de contenu que le visiteur sélectionne en glissant la souris par-dessus.

Ce sélecteur a été introduit dans CSS 3... mais en a été (temporairement ?) retiré il y a quelque temps, bien qu'il fonctionne parfaitement sur WebKit, Opera et Firefox. En voici un exemple :

```
::selection {
  background: yellow; /* Safari, Chrome, Opera */
}
::-moz-selection {
  background: yellow; /* Firefox */
}
```

Non reconnu par – faut-il le préciser ? – Internet Explorer, nul ne peut deviner aujourd'hui si ce pseudo-élément réintégrera les spécifications CSS un jour.

:contains

Tout comme le regretté élément `::selection`, la pseudo-classe `:contains()` fut annoncée puis retirée de CSS 3 malgré son attrait : elle s'appliquait à un élément pour peu qu'il contienne la chaîne de caractères explicitée dans les parenthèses.

Dans ma grande mégalomanie, je pourrais ainsi mettre en exergue tous les commentaires reçus des visiteurs d'Alsacréations qui mentionneraient le site en oubliant le « s » final :

```
:contains("alsacreation ") {
  background: red;
}
```

À ma connaissance, ce sélecteur n'est implémenté sur aucun navigateur actuel, mais le concept n'en demeure pas moins intéressant.

Exercice pratique : tableau de données

L'objectif de cet exercice est de styler un tableau de données en CSS 2 et CSS 3 en évitant tout élément superflu (classe, `id`, attribut) dans le code HTML, qui devra demeurer le plus épuré possible.

Vous observerez au sein de la structure ci-après (figure 8-33) – outre les éléments classiques d'un tableau – un attribut malheureusement sous-exploité : `scope`. Celui-ci précise si l'en-tête de cellule (`<th>`) doit être associé à sa ligne ou à sa colonne.

Partie HTML

```
<table summary="Le prix imaginaire de certains fruits sur le marché">
  <caption>Prix des fruits</caption>
  <tr>
    <td></td>
    <th scope="col">Origine</th>
    <th scope="col">Prix</th>
  </tr>
  <tr>
    <th scope="row">Kiwi</th>
    <td>France</td>
    <td>42€</td>
  </tr>
  <tr>
    <th scope="row">Pastèque</th>
    <td>Taïwan</td>
    <td>1337€</td>
  </tr>
  <tr>
    <th scope="row">Papaye</th>
    <td>Roumanie</td>
    <td>1.3€</td>
  </tr>
  <tr>
    <th scope="row">Melon</th>
    <td>France</td>
    <td>12€</td>
  </tr>
  <tr>
```

```
    <th scope="row">Litchi</th>
    <td>Groënland</td>
    <td>15.50€</td>
  </tr>
</table>
```

Figure 8-33

Le rendu par défaut du tableau

Prix des fruits

	Origine	Prix
Kiwi	France	42€
Pastèque	Taïwan	1337€
Papaye	Roumanie	1.3€
Melon	France	12€
Litchi	Groënland	15.50€

Commençons par appliquer une mise en forme de base pour rendre ce tableau de données plus attrayant : tout d'abord, fixons une largeur de 25 em au tableau et séparons toutes les cellules de données de 2 pixels entre elles à l'aide des propriétés `border-collapse` et `border-spacing` :

```
table {
  width: 25em;
  border-collapse: separate;
  border-spacing: 2px;
}
```

Notre mission suivante est d'appliquer des marges internes de 6 px et 12 px sur chacun des éléments `<td>` et `<th>`. Puis, les cellules d'en-tête devront être alignées à gauche et tous les contenus de la dernière colonne (prix) devront être alignés à droite.

```
td, th {
  padding: 6px 12px;
}
th {
  text-align: left;
}
td:last-child, th:last-child { /* :last-child non compris par  IE8 */
  text-align: right;
}
```

N'oublions pas de décorer la légende du tableau (`<caption>`) grâce aux propriétés de couleurs, d'italique et de bordures :

```
table caption {
  color: #555;
  font-style: italic;
  padding: 10px;
  border-bottom: 1px solid #ccc;
}
```

Allons même plus loin en positionnant la légende en bas des contenus, très simplement en appliquant la propriété `caption-side` au tableau (reconnue à partir d'IE8) :

```
table {
  width: 25em;
  border-collapse: separate;
  border-spacing: 2px;
  caption-side: bottom;
}
```

Nouvelle étape, les rangées paires doivent être de couleur `#eee` et les rangées impaires de couleur `#ddd`. Pour ce faire, nous emploierons le sélecteur CSS 3 `:nth-child` reconnu par tous les navigateurs à l'exception d'Internet Explorer 8 :

```
tr:nth-child(odd) {
  background: #eee;
}
tr:nth-child(even) {
  background: #ddd;
}
```

La première rangée, contenant les en-têtes « origine » et « prix », doit être de couleur `#cba` et son texte doit être blanc :

```
tr:first-child {
  background: #cba;
  color: white;
}
```

Pour finir en beauté ce traitement esthétique, interdisons à la première cellule, vide, d'afficher un arrière-plan (figure 8-34). Plus précisément, la couleur de fond devra être blanche comme le fond de la page :

```
table td:empty {
  background: white;
}
```

Figure 8-34

Mise en forme du tableau

	Origine	Prix
Kiwi	France	42€
Pastèque	Taïwan	1337€
Papaye	Roumanie	1.3€
Melon	France	12€
Litchi	Groënland	15.50€

Prix des fruits

Définissons à présent un comportement spécifique lors du survol avec la souris :

- Les rangées doivent afficher une couleur d'arrière-plan `#bbb` et un texte de couleur blanche.
- Chaque cellule survolée doit avoir une couleur de fond `#999`.
- La première rangée (et aucune de ses cellules) ne doit pas changer d'arrière-plan au survol.

L'exercice nécessite de distinguer la première ligne des autres lors du survol. Le sélecteur d'adjacence direct (`tr + tr`) permet d'appliquer des propriétés à toutes les rangées sauf à la première. C'est exactement ce que nous allons mettre en pratique (figure 8-35) :

```
table td:empty, table td:empty:hover {
  background: white;
}
table tr + tr:hover {
  background: #bbb;
  color:white;
}
table tr + tr td:hover, table tr + tr th:hover {
  background: #777;
}
```

Au terme de ce travail pratique, le tableau de données et la plupart des effets appliqués sont reconnus par l'ensemble des navigateurs, même par Internet Explorer 7 à l'exception de deux comportements : la gestion des lignes paires et impaires (`:nth-child`) et le positionnement de la légende (`caption-side`). Rien de vraiment bloquant, n'est-ce pas ?

Figure 8-35

Effets de survol des cellules

	Origine	Prix
Kiwi	France	42€
Pastèque	Taïwan	1337€
Papaye	Roumanie	1.3€
Melon	France	12€
Litchi	Groënland	15.50€

Prix des fruits

Visualiser le résultat en ligne

▸ http://www.ie7nomore.com/fun/tablepimp/

ALLER PLUS LOIN **Cibler et styler des colonnes de tableau**

Assez mal connu, l'élément HTML 4 `<col>` (pourtant reconnu par tous les navigateurs) a pour fonction de désigner l'ensemble d'une colonne de tableau et offre ainsi une souplesse étendue en termes de mise en forme. Ainsi, il suffit de spécifier autant de `<col>` en début de tableau que de colonnes présentes, puis de cibler en CSS l'élément `<col>` comme le montre l'exemple suivant.

Partie HTML :

```
<table>
  <col /><col /><col />
  <tr><td>cellule 1</td><td>cellule 2</td><td>cellule 3</td></tr>
  <tr><td>cellule 4</td><td>cellule 5</td><td>cellule 6</td></tr>
</table>
```

Partie CSS :

```
col:first-child {background: #ddd;}
col:nth-child(odd) {background: #eee;}
col:nth-child(even) {background: #ccc;}
```

Cette situation est également consultable à l'adresse :

- http://www.ie7nomore.com/fun/tablepimp/

Media Queries : requêtes de média CSS

CSS 3 améliore la gestion des styles en fonction des périphériques de sortie et de leurs particularités en introduisant la notion de « requête de média » ou *Media Query*. Il devient possible de limiter la portée de styles CSS selon un environnement maîtrisé, par exemple uniquement sur les écrans dont la résolution est inférieure à 800 pixels.

Cette fonctionnalité, reconnue partout et prévue sur IE9, est particulièrement précieuse pour l'adaptation de son site au Web mobile.

Syntaxe

Une *Media Query* se présente sous la forme d'une requête entre parenthèses, avec ou sans opérateurs logiques, associée au type de terminal.

Par exemple, `screen and (width:800px)` s'appliquera uniquement aux écrans dont la largeur occupe exactement 800 pixels.

La requête peut s'effectuer au sein du classique élément `<link>` :

```
<link rel="stylesheet" media="screen and (width:800px)" href="color.css" />
```

Elle peut également prendre place dans une règle @ (vue en début d'ouvrage au sein du chapitre 2) présente dans une feuille de styles :

```
@media screen and (width:800px) { … }
```

Opérateurs logiques

Divers opérateurs logiques combinent les requêtes : `and` (et), `only` (uniquement) et `not` (non). L'opérateur « ou » n'existe pas, mais il est possible de le représenter à l'aide du signe virgule (`,`) entre deux requêtes, un peu comme le sélecteur de groupe en CSS.

```
@media screen and (width:800px) { … } /* écrans de 800px exactement */
@media screen and (not width:800px) { … } /* tous les écrans sauf ceux de 800px
➥ exactement */
@media screen and (width:800px) and (color) { … } /* écrans de 800px exactement
➥ et en couleur */
@media screen and (width:800px), screen and (color) { … } /* écrans de 800px
➥ exactement ou écrans couleurs */
```

Dans la pratique, l'opérateur `only` est principalement employé pour masquer les requêtes aux anciens navigateurs qui ne reconnaissent pas les Media Queries. Il s'agit simplement d'une précaution :

```
@media only screen and (width:800px) { … } /* styles masqués sur les anciens
➥ navigateurs */
```

Requêtes et préfixes

Couleurs, dimensions et orientation comptent parmi les critères de requêtes généralement définis au sein d'une requête de média. Ajoutons à ces requêtes deux préfixes facultatifs `min` et `max` faisant office de « plus petit ou égal à » ou « plus grand ou égal à ».

Tableau 8-15 Critères utilisés dans les Media Queries

Critère	Description	Exemples
`color`	Sans valeur associée, cette requête filtre les périphériques en couleur. Associée à une valeur, elle indique le nombre de bits de couleurs du périphérique.	`(min-color: 4)`
`width`	Largeur de la surface de rendu du périphérique de sortie	`(min-width: 800px)` désigne les terminaux dont la largeur est au moins de 800 pixels.
`height`	Hauteur de la surface de rendu du périphérique de sortie	`(max-height: 800px)` désigne les terminaux dont la hauteur est inférieure ou égale à 800 pixels.
`device-width`	Largeur physique du périphérique de sortie	`(max-device-width: 320px)` désigne un terminal mobile dont la largeur est inférieure ou égale à 320 pixels, tel l'iPhone 3.
`device-height`	Hauteur physique du périphérique de sortie	

Critère	Description	Exemples
`orientation`	Choisit un périphérique tourné dans le sens vertical (`portrait`) ou horizontal (`landscape`).	
`aspect-ratio`	Format d'image du périphérique de sortie	`(aspect-ratio: 16/9)`
`resolution`	Résolution du périphérique de sortie	@media print `and (min-resolution: 300dpi)`
`device-pixel-ratio`	Densité de pixels du périphérique de sortie	`(min-device-pixel-ratio: 2)` désigne un terminal dont la densité de pixels est au moins égale à 2, tel l'iPhone 4.

Exercice pratique : s'adapter à la taille de l'écran

Mettons en pratique les bienfaits des Media Queries au sein d'un exercice consistant à adapter une mise en page à la largeur de l'écran. L'objectif à atteindre est de varier d'un affichage en trois colonnes à un affichage sur une seule colonne si l'écran est réduit ou si le visiteur navigue à l'aide d'un téléphone mobile.

La structure HTML se compose d'un trio de paragraphes enfants d'un conteneur `<div>` identifié `"main"` :

Partie HTML

```
<div id="main">
  <p>Premier paragraphe</p>
  <p>Deuxième paragraphe</p>
  <p>Troisième paragraphe</p>
</div>
```

Nous allons entamer l'exercice par l'agencement des paragraphes. Pour les disposer côte à côte, plusieurs schémas de positionnement sont envisageables. Pour des raisons de facilité, j'ai opté pour le modèle de rendu tabulaire (non reconnu par IE6 et IE7), mais rien ne vous empêche d'user de flottants :

Partie CSS

```
#main {
  display: table;
  table-layout: fixed;
  width: 100%;
}
#main p {
  display: table-cell;
  padding: 5px;
  margin: 0;
  background: #6B9A49;
}
```

Pour appliquer une couleur de fond différente à chacun des paragraphes, il nous suffit de manier à bon escient le sélecteur CSS 2.1 d'adjacence (figure 8-36) :

```
#main p + p {
  background: #E69B00;
}
#main p + p + p {
  background: #333;
}
```

Figure 8-36

Aperçu visuel en grande taille d'écran

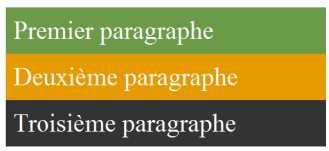
Arrive enfin le moment où l'on va tenir compte de la résolution de l'écran du visiteur. À la suite des règles précédentes, ajoutons simplement une Media Query ciblant les écrans dont la largeur est inférieure ou égale à 600 pixels de large. Au sein de cette fonction, nous allons nous contenter d'afficher les paragraphes sous forme de blocs et non plus de cellules (figure 8-37) :

```
@media only screen and (max-width: 600px)  {
  #main p {display: block;}
}
```

Le test est concluant : tant que l'écran est plus large que 600 pixels, les trois paragraphes s'affichent les uns à côté des autres, mais dès que l'écran se restreint, ils s'empilent les uns sur les autres en bloc.

Figure 8-37

Affichage sous forme de blocs sur petits écrans



Il ne reste plus qu'à faire fonctionner le principe sur Internet Explorer 6, 7 et 8, incapables de prendre en charge les Media Queries. Pour cela, un outil JavaScript est téléchargeable à l'adresse :

▸ https://github.com/scottjehl/respond

Pour ne pas pénaliser les autres navigateurs, la mise en application de ce script passera par un commentaire conditionnel.

Partie HTML

```
<!--[if lte IE8]> <script type="text/javascript" src="respond.min.js"></script>
<![endif]-->
```

Il est également nécessaire d'ajouter un commentaire CSS à la fin des déclarations de Media Queries (sans ajouter d'espace après l'accolade fermante) :

Partie CSS

```
@media only screen and (max-width:600px) {
#main p {display:block}
}/*mediaquery*/
```

Et le tour est joué !

Visualiser le résultat en ligne

N'hésitez pas à décortiquer le code source du résultat observable en ligne à :

- http://www.ie7nomore.com/fun/media-queries/

Et découvrez une superbe application réelle de cette méthode sur le site vainqueur de notre concours *Cascading Style Summer Refresh 2010* sur Alsacréations :

- http://juslisen.com

CSS Transformations

La propriété CSS 3 `transform` permet d'appliquer des transformations en deux dimensions sur un élément : rotation, décalage, zoom, déformation et perspective.

Là encore, les préfixes propriétaires sont de mise : pour toutes les fonctionnalités de ce module, il est nécessaire d'écrire les propriétés en débutant par `-moz-`, `-o-` et `-webkit-`. Je n'évoque Internet Explorer que pour vous confirmer que les transformations ne sont opérationnelles qu'à partir d'IE9.

scale : fonction de zoom

Cette fonction agrandit ou réduit les dimensions d'un élément selon un ratio : une valeur inférieure à 1 aura pour conséquence de rapetisser l'élément, un chiffre supérieur à 1 va l'agrandir. La place occupée dans le flux demeure identique : si l'élément est agrandi, ses frères ne seront pas poussés en conséquence.

Voici comment réduire de moitié les éléments de classe `minify` (figure 8-38) :

```
.minify {
  -moz-transform: scale(0.5);
  -webkit-transform: scale(0.5);
  -o-transform: scale(0.5);
  transform: scale(0.5);
}
```

Notez que comme toutes les autres techniques de transformation, la fonction `scale` peut être cumulée avec les propriétés de transitions afin de créer des effets lors du survol de la souris, par exemple.

Figure 8-38

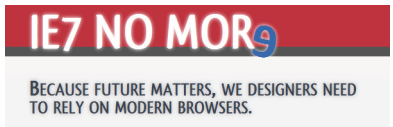
Illustration de scale

Petite astuce intéressante : en spécifiant une valeur de –1 à l'un des axes (fonctions `scaleX` et `scaleY`), l'élément sera retourné comme dans un miroir. Voici comment j'ai tourné et inversé la lettre « e » sur mon site www.ie7nomore.com (figure 8-39) :

```
h1 span {
  -moz-transform: rotate(10deg) scaleX(-1);
  -webkit-transform: rotate(10deg) scaleX(-1);
  -o-transform: rotate(10deg) scaleX(-1);
  transform: rotate(10deg) scaleX(-1);
}
```

Figure 8-39

Illustration d'un effet miroir



rotate : rotation

Le concept de rotation des éléments est introduit dans le module Transforms de CSS 3 via la fonction `rotate`, dont les valeurs sont exprimées en degrés (`deg`) ou en radians (`rad`).

Ainsi, pour tourner les éléments de classe `rotation` de 45 degrés dans le sens des aiguilles d'une montre (figure 8-40), nous écrirons :

```
.rotation {
  -moz-transform: rotate(45deg);
  -webkit-transform: rotate(45deg);
  -o-transform: rotate(45deg);
  transform: rotate(45deg);
}
```

Figure 8-40

Illustration de rotate

Cette fonction n'est pas reconnue par Internet Explorer, mais ce dernier propose une alternative pour appliquer des rotations simples (90°, 180°, 270°) à l'aide d'un filtre propriétaire de type `filter: progid:DXImageTransform.Microsoft.BasicImage(rotation=1);` où la valeur 1 correspond à un angle de 90°, par exemple.

S'il s'agit de tourner du texte, sachez qu'une propriété CSS 3, `writing-mode`, reconnue par Internet Explorer uniquement, permet des rotations simples (90°, 180°, 270°) :

```
p {
  -ms-writing-mode: bt-lr; /* rotation de texte à 90° */
}
```

skew, translate et matrix : déformations et translations

Le panel des effets de transformation couverts par les spécifications CSS 3 ne se limite pas aux rotations et aux grossissements. Des déformations de toutes sortes et des translations figurent également au menu des réjouissances.

`skew`, fonction raccourcie de `skewX` et `skewY`, modifie la perspective de l'élément en « tirant » sur ses coins de manière à obtenir au final une forme parallélépipédique. La première valeur concerne l'axe des X, la seconde l'axe des Y :

```
.skew {
  -moz-transform: skew(-25deg, 10deg);
  -webkit-transform: skew(-25deg, 10deg);
  -o-transform: skew(-25deg, 10deg);
  transform: skew(-25deg, 10deg);
}
```

La fonction `translate` opère une translation de l'élément sur les axes X et Y. L'exemple suivant va déplacer la division `<div>` de 100 pixels dans les deux directions :

```
div {
  -moz-transform: translate(100px, 100px);
  -webkit-transform: translate(100px, 100px);
  -o-transform: translate(100px, 100px);
  transform: translate(100px, 100px);
  }
```

La fonction `matrix` construit une transformation à partir d'une matrice mathématique. La propriété est complexe à maîtriser et je ne la développerai pas dans ce livre. Cependant, je laisse aux plus curieux la possibilité de se renseigner directement à la source des spécifications W3C :

▸ http://www.w3.org/TR/SVG/coords.html#TransformMatrixDefined

Propriété raccourcie

Pour finir cette partie en beauté, notez que la propriété raccourcie `transform` regroupe l'ensemble des fonctions décrites précédemment. Il suffit de les écrire bout à bout, simplement séparées par un espace, par exemple :

```
div {
  -moz-transform: scale(2) rotate(45deg) translate(5px,10px);
  -webkit-transform: scale(2) rotate(45deg) translate(5px,10px);
  -o-transform: scale(2) rotate(45deg) translate(5px,10px);
  transform: scale(2) rotate(45deg) translate(5px,10px);
}
```

CSS Transitions

Combien d'intégrateurs en ont rêvé : animer les pages web uniquement à l'aide de styles CSS, sans apport de JavaScript. Grâce aux dernières évolutions du langage et au module CSS 3 Transitions, il est désormais possible de réaliser des transitions basiques à l'aide de CSS dans les navigateurs très récents (Safari 4, Chrome 2, Firefox 4, Opera 10.5)... et dans Internet Explorer 9.

RENDONS À CÉSAR... Transitions CSS 3

Cette partie sur les transitions CSS 3 est largement inspirée d'un tutoriel sous licence libre Creative Commons publié sur Alsacreations.com par Antoine Cailliau (ac-graphic.net). Je le remercie de bien vouloir partager ses connaissances.

Le principe de base d'une transition CSS 3 est de permettre un passage en douceur de l'ancienne vers la nouvelle valeur d'une propriété CSS lorsqu'un événement est déclenché :

- soit via une pseudo-classe telle que `:hover`, `:focus` ou `:active` ;
- soit via JavaScript.

Précédemment, ce genre de comportement n'était possible qu'avec l'usage de JavaScript. Ce nouveau module issu de CSS 3 permet dorénavant de s'en affranchir au profit exclusif des feuilles de styles.

Pour définir une nouvelle transition animée, il est nécessaire de préciser au minimum deux composantes :

- la (ou les) propriété(s) à animer (`transition-property`) ;
- la durée de l'animation (`transition-duration`).

PRÉFIXES VENDEURS Module Transition

Puisque le module Transitions n'est pas encore finalisé, vous êtes invités à faire précéder les fonctions par les préfixes des différents navigateurs. Ainsi, toutes les propriétés de transition devront débuter par `-moz-` sur Firefox, `-o-` sur Opera et `-webkit-` sur Chrome et Safari.

Voici un exemple pour comprendre le principe général :

```
img {
  width: 50px;  /* état par défaut */
  transition-property: width; /* transition de la propriété width au survol
  ➥ de la souris */
  transition-duration: 1s; /* durée de la transition */
}
img:hover {
  width: 200px;
}
```

Les deux propriétés minimales nécessaires pour rendre fonctionnelle une transition en CSS 3 sont `transition-property` et `transition-duration`. Il existe cependant d'autres propriétés CSS spécifiques aux transitions : `transition-timing-function`, `transition-delay` et la notation raccourcie `transition`, qui cumule tous ces aspects.

Propriété à animer : transition-property

La propriété `transition-property` accepte trois valeurs :

- `all` (valeur par défaut) : toutes les propriétés possibles seront animées ;
- `propriété` : le nom d'une propriété pouvant être animée ;
- `none` : aucune propriété ne sera animée.

Durée de l'animation : transition-duration

La propriété `transition-duration` indique la durée de la transition. Si plusieurs propriétés ont été définies à l'aide de `transition-property`, il est possible de préciser leurs valeurs en les séparant d'une virgule.

Les deux unités de temps acceptées sont :

- `s` : la seconde ;
- `ms` : la milliseconde.

Pour préciser une durée de 5 secondes pour la transition, nous pouvons alors utiliser la règle suivante :

```
selecteur {
  transition-duration: 5s;
}
```

Accélération

La propriété `transition-timing-function` détermine la fluidité de l'animation. Les mots-clés admissibles pour cette propriété sont :

- `ease` : rapide sur le début et ralenti sur la fin ;
- `linear` (valeur par défaut) : vitesse constante sur toute la durée de l'animation ;
- `ease-in` : lent sur le début et accélère de plus en plus vers la fin ;
- `ease-out` : rapide sur le début et décélère sur la fin ;
- `ease-in-out` : le départ et la fin sont lents.

Notation raccourcie : transition

La notation raccourcie `transition` permet de décrire facilement et de manière concise les différentes propriétés en jeu à l'aide d'une seule propriété. La syntaxe est la suivante :

```
selecteur {
transition: <transition-property> <transition-duration>  <transition-timing-function>
➥ <transition-delay>;
}
```

En pratique, voici comment créer une transition d'une seconde de toutes les propriétés d'une image lors de son survol :

```
img {
  transition: all 1s ease-in;
}
```

Propriétés compatibles

Toutes les propriétés CSS ou valeurs ne sont pas prévues pour bénéficier de transitions, mais la liste qui suit offre de belles opportunités :

- les couleurs ;
- toutes les valeurs numériques ;
- les transformations (avec des limitations) ;
- les dégradés « gradient » (avec des limitations).

Tableau 8-16 Liste complète des propriétés susceptibles d'être appliquées

background-color	margin-left
background-position	margin-right
border-bottom-color	margin-top
border-bottom-width	max-height
border-color	max-width
border-left-color	min-height
border-left-width	min-width
border-right-color	opacity
border-right-width	outline-color
border-spacing	outline-offset
border-top-color	outline-width
border-top-width	padding-bottom
border-width	padding-left
bottom	padding-right
clip	padding-top
color	right
crop	text-indent
font-size	text-shadow
font-weight	top
grid-*	vertical-align
height	visibility
left	width
letter-spacing	word-spacing
line-height	z-index
margin-bottom	

Compatibilité et conclusion

Les fonctions de transition sont encore un peu jeunes pour être exploitées en production : non seulement elles ne seront prises en compte qu'à partir d'Internet Explorer 9, mais même sur Firefox, il est nécessaire de disposer de la version 4 pour les voir en action.

Par ailleurs, ces nouvelles possibilités offertes par CSS posent la question de la répartition des rôles pour chaque langage au sein d'un document web : les trois entités structure, mise en forme et comportement étaient jusque-là gérées – avec leurs limites – respectivement par (X)HTML, CSS et JavaScript. À présent que les feuilles de styles permettent d'animer les éléments de la page, la frontière avec JavaScript se fait plus ténue pour tout ce qui a trait au domaine visuel.

Exercice pratique : un menu de navigation avec transition

Chaque jour, un nombre impressionnant de sites web exposent d'abondants effets de style fondés sur les propriétés de transition en CSS.

Nos travaux pratiques vont se limiter à une application concrète sur un menu de navigation vertical : lors du survol ou de la prise de focus d'un lien, celui-ci va s'élargir graduellement pour revenir à son état initial lorsque la souris le quitte.

Le code HTML se base sur une liste classique non ordonnée.

Partie HTML

```
<ul class="menu">
  <li><a href="*">Choucroute</a></li>
  <li><a href="*">Bière</a></li>
  <li><a href="*">Knack</a></li>
  <li><a href="*">Hopla</a></li>
  <li><a href="*">Kuglopf</a></li>
</ul>
```

Commençons par la traditionnelle mise en forme minimale : suppression des puces des éléments du menu, décoration des liens à l'aide de marges, de couleurs d'arrière-plan et de bordures. Jusque-là, il n'y a rien d'exceptionnel.

Partie CSS

```
.menu li {
  list-style: none;
}
.menu a {
  padding: 8px 12px;
  border: 2px solid #fff;
  text-shadow: 0px 0px 6px #777;
  background: #98B924;
  color: #fff;
  text-decoration: none;
}
```

Pour éviter les chevauchements dus aux marges internes et pour conserver une largeur définie par le contenu de chaque lien, nous allons les afficher sous forme `inline-block` :

```
.menu a {
  display: inline-block;
  padding: 8px 12px;
  …
}
```

L'étape suivante consiste à déterminer l'état des liens au survol et au focus. J'ai choisi de jouer avec la composante de marge interne (`padding`) en modifiant uniquement sa valeur à droite de 150 pixels, et d'éclaircir légèrement la teinte de l'arrière-plan :

```
.menu a:hover, .menu a:focus {
  padding-right: 150px;
  background: #b8da40;
}
```

Le résultat est d'ores et déjà plaisant mais pour le moins abrupt. Il ne nous reste plus qu'à adoucir la transition entre les deux états à l'aide de la propriété CSS 3 éponyme et de ses nombreux préfixes vendeurs.

La tâche est bien plus simple qu'on ne pourrait le croire. Il suffit de décliner la propriété `transition` en lui attribuant les valeurs `.5s` pour la durée et `all` pour lui indiquer que toutes les propriétés modifiées participent à la transition. Dans notre cas, cela signifie que la couleur de fond et le `padding` subiront les effets souhaités :

```
.menu a {
  …
  -webkit-transition: all .5s;
  -moz-transition: all .5s;
  -o-transition: all .5s;
  transition: all .5s;
}
```

Figure 8-41

Rendu final au survol

Le résultat final (figure 8-41) s'affiche parfaitement sur le moteur de rendu WebKit (Safari et Chrome), ainsi que sur les dernières versions de Firefox et Opera.

Visualiser le résultat en ligne

- http://ie7nomore.com/fun/transition/

Vous pouvez observer et analyser quelques-uns de mes autres essais, dont le site http://www.goetter.fr truffé de transitions sur la plupart des éléments interactifs, ou un exercice de menu déroulant avec transition :

- http://ie7nomore.com/fun/menu2/

Ou encore un menu complètement flexible :

- http://www.ie7nomore.com/css2only/flexibletab/

CSS Animations

Définies comme une extension du module Transitions, les animations CSS apportent une dimension supplémentaire en introduisant le concept d'étapes et de nombre d'itérations.

Ce chapitre nécessiterait un ouvrage dédié et dépasse quelque peu le cadre de cet ouvrage « général » sur les techniques CSS avancées. En outre, ce module ne fonctionne actuellement que sur le moteur WebKit (ainsi que sur le tout récent Firefox 4), ce qui en réduit l'usage. Cependant, je ne résiste pas à la tentation de vous en montrer très brièvement le fonctionnement.

Le cœur des animations CSS réside dans les différentes images-clés (*keyframes*) qui sont autant d'étapes dans le cheminement de l'action. Dans un premier temps, donnons le nom de `disparition` à notre animation et définissons ses différentes étapes :

```
@-webkit-keyframes disparition {
  0% {opacity: 1;} /* opacité complète */
  50% {opacity: 0;} /* opacité nulle à la moitié de l'animation */
  100% {opacity: 1;} /* retour à l'état initial */
}
```

Il nous suffit à présent d'appliquer l'animation à l'élément souhaité, par exemple lors du survol des images :

```
img {
  -webkit-animation-name: disparition;  /* nom de l'animation */
  -webkit-animation-duration: 2s;  /* durée totale */
  -webkit-animation-iteration-count: infinite;  /* nombre d'itérations */
}
```

Ce comportement d'animation en CSS est très attrayant, mais n'a pas encore atteint une maturité suffisante pour être véritablement exploité en production.

OUTIL Sencha Animator, un outil d'animation convivial

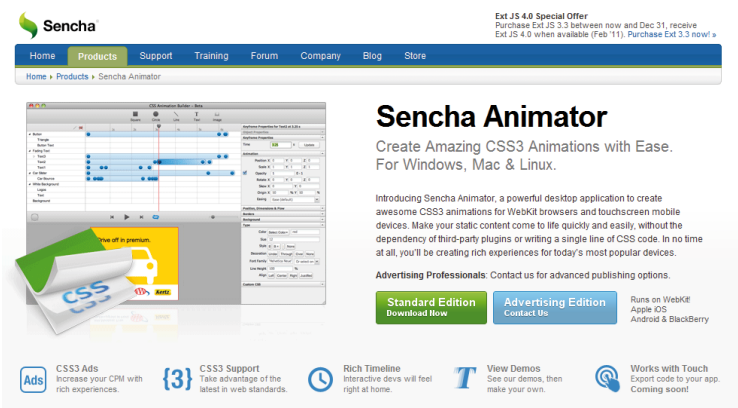
Sencha Animator se propose de créer des animations CSS 3 grâce à une interface utilisateur conviviale, proche d'un programme de dessin vectoriel (figure 8-42). Il est possible d'intégrer des objets, de les déplacer, redimensionner, avec différents niveaux d'imbrication. Des transformations de rotation et de déformation sont disponibles ainsi que les dégradés, les flous et les ombrages.

Pour le moment, l'inconvénient est bien sûr que seuls les navigateurs récents peuvent en profiter, principalement ceux basés sur les dernières versions de WebKit et Firefox 4. Sencha Animator est en phase de développement, la version actuelle est téléchargeable mais les futures seront payantes.

▸ http://www.sencha.com/products/animator/

Figure 8-42

Aperçu de Sencha Animator



Alternatives CSS 3 pour Internet Explorer

Élève assidu et perspicace que vous êtes, vous ne pouviez parvenir à l'épilogue de ce chapitre dédié à CSS 3 sans en conclure que « c'est bien joli tout ça, mais si ça ne fonctionne pas avant Internet Explorer 9 ou d'autres ténors, en quoi cela va-t-il bouleverser ma vie quotidienne ? »

Si cela peut vous rassurer, vous n'êtes pas le seul frustré : croyez-moi, de nombreux concepteurs web et développeurs du monde entier se lamentent de cet état de fait, tant et si bien que de multiples ressources pour déjouer les faiblesses d'Internet Explorer voient le jour presque quotidiennement, notamment par l'ajout de JavaScript.

Certaines de ces rustines offrent la prise en charge des propriétés CSS 3 à la lignée IE6-IE8, d'autres lui permettent de reconnaître les différents nouveaux sélecteurs et pseudo-classes. Tout cela a un prix : ces outils sont généralement gourmands en ressources. On en vient à soupeser les avantages et inconvénients de chacune de ces solutions : d'un côté, la rapidité d'intégration, de l'autre, une baisse de performance non négligeable sur ce navigateur. Selon le contexte, le public ciblé ou nos affinités, la balance ne penche pas toujours du même côté.

Reconnaissance des propriétés CSS 3

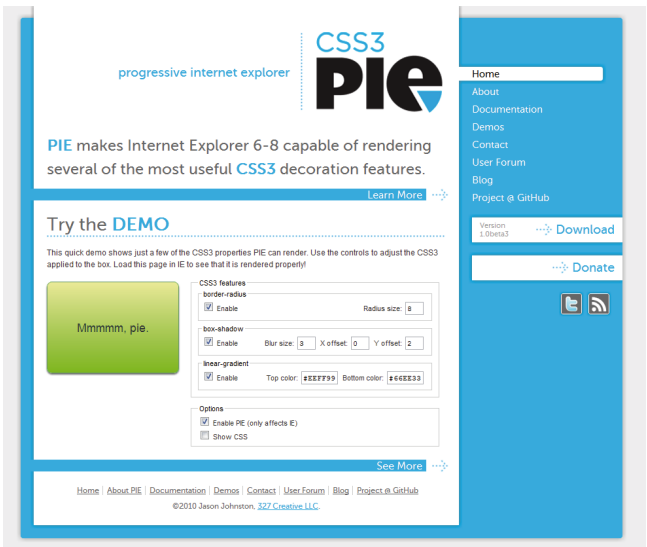
Dans la jungle des outils permettant d'émuler CSS 3 sur Internet Explorer, CSS3Pie (www.css-3pie.com) semble être le plus robuste aujourd'hui et bénéficie d'une communauté et d'un compte Twitter plutôt actifs (figure 8-43). Via un script .HTC similaire à un fichier JavaScript à télécharger, CSS3Pie permet d'appliquer un certain nombre de propriétés CSS 3 intéressantes : `border-radius`, `box-shadow`, `border-image`, arrière-plans multiples et dégradés linéaires. Pour l'avoir testé sur plusieurs projets web, j'ai pu apprécier les prouesses de cet outil de 28 Ko.

Si vous ne souhaitez réaliser que des coins arrondis, un autre script très léger (8 Ko) et nommé Roundies s'en chargera très bien. Vous le trouverez à cette adresse :

> http://www.dillerdesign.com/experiment/DD_roundies/

Figure 8-43

Le site web CSS3Pie.com



Entièrement dédié aux transformations, le script TransformIE ouvre la voie aux fonctions `rotate`, `scale`, `skew` et `matrix` sur Internet Explorer.

> http://www.transformie.com

Si vous lorgnez du côté des colonnes multiples, sachez qu'il existe un excellent article anglophone à ce sujet sur le site de référence A List Apart.

> http://www.alistapart.com/articles/css3multicolumn

Ce didacticiel met en avant un outil JavaScript permettant l'emploi des propriétés de colonnes sur Internet Explorer. Vous trouverez cet outil CSS3 Multi Column à cette adresse :

> http://www.csscripting.com/wiki/index.php?title=CSS3_Multi_Column

Reconnaissance des sélecteurs CSS 3

Faire reconnaître les différents sélecteurs et pseudo-classes CSS 3 à Internet Explorer est une mission moins périlleuse qu'il y paraît… à condition toutefois d'user d'un zeste de JavaScript et plus particulièrement d'un framework tel que jQuery, MooTools ou Prototype.

jQuery permet en une courte ligne de code d'émuler n'importe quel sélecteur CSS 3 sur les versions IE6 à IE8. Voici par exemple comment faire comprendre le sélecteur `li:last-child` à Internet Explorer en créant automatiquement une classe `last` sur le dernier enfant `<li>` :

```
$("li:last-child").addClass("last");
```

Si vous n'êtes pas familier avec les rudiments du langage JavaScript, ne vous inquiétez pas : le script Selectivizr vous mâche le travail (figure 8-44). Cet outil émule les différents sélecteurs CSS 3 (`:first-child`, `:last-child`, `:nth-child`, `:focus`, `:target`…) sur IE6-IE8 sous forme de plug-in ajouté à un framework existant (jQuery, Mootools, Prototype…). Vous pouvez le télécharger à cette adresse :

> http://selectivizr.com/

Figure 8-44

Tableau des sélecteurs CSS3 reconnus par Selectivizr

Trousse à outils

Voici pour conclure sur cette douloureuse partie de la compatibilité d'Internet Explorer quelques outils pratiques pour faciliter votre tâche de concepteur web avant-gardiste :

- **CSS3Please** : CSS3Please.com n'est rien d'autre qu'une excellente démonstration en direct et interactive de différentes propriétés CSS 3 (arrondis, ombrages, dégradés, transparence, rotation, transition, `font-face`). Vous pouvez modifier les options et valeurs de la page, elles s'appliqueront instantanément sur la boîte d'exemple et l'ensemble des syntaxes (`-moz-`, `-webkit-` et filtres IE) est proposé.
- **CSS3Generator** : dans la même veine que CSS3Please, CSS3Generator.com est un outil de création automatique de syntaxes CSS dont le résultat est affiché en direct à l'écran. Certaines fonctionnalités telles que les colonnes multiples, `resize`, `box-sizing` ou `outline` le démarquent de ses concurrents.
- **Border-radius.com** : grâce à border-radius.com, vous pouvez tester vos coins arrondis en direct.
- **Can I use** : caniuse.com est un récapitulatif des dernières technologies et de leur prise en charge sur les générations de navigateurs.

Troisième partie

CSS et applications spécifiques

9

CSS pour le Web mobile

Parce que faire un site web pour terminal mobile, ce n'est *pas* fixer sa largeur à 320 pixels ou proposer une version iPhone, voici un tour d'horizon de toutes les solutions offertes par CSS pour adapter une présentation existante aux mobinautes : gérer la largeur, redimensionner les éléments, passer à une seule colonne, gérer les débordements, supprimer le superflu, adapter les liens et les polices et un zeste de HTML 5 en guise de cerise sur le gâteau.

État des lieux

Pour être en mesure de cerner le contexte actuel de l'univers mobile extrêmement riche en innovations technologiques et pour comprendre vers quelles orientations et standards se dirigent les acteurs majeurs, je vous propose de retracer brièvement les événements et techniques qui nous ont amenés jusqu'à nos téléphones « intelligents » (*smartphones*).

Historique

Plusieurs générations de supports mobiles se sont succédées avant de parvenir à un stade d'évolution permettant de surfer sur Internet de façon acceptable.

Le premier téléphone cellulaire en France fut le Bi-Bop, né en 1993 et retiré en 1997 (figure 9-1). Il se comportait un peu à la manière du Wi-Fi actuel en se connectant à des bornes de faible portée et n'a été déployé que dans trois villes tests : Paris, Lille et Strasbourg. Il fut rapidement remplacé par la norme GSM, propulsée en France par Itineris (ancien nom d'Orange, marque de France Télécom) qui, dès 1998, couvre 97 % de la population française. S'en suit une explosion du marché de la téléphonie mobile.

Figure 9-1

Le Bi-Bop

Les premiers balbutiements de l'Internet mobile ont lieu en 1999 avec la création de protocoles dédiés tels que i-Mode et WAP (*Wireless Application Protocol*), chacun se fondant sur des langages dérivés de SGML : i-HTML, C-HTML pour i-Mode et WML pour WAP. Même si un groupe de travail mobile se crée au sein du W3C, ce n'est pas lui qui chapeaute ces technologies mais le consortium Open Mobile Alliance (www.openmobilealliance.org), créé en 2002.

L'appellation Wi-Fi voit le jour en 1999. Il s'agit d'un réseau local sans fil reposant sur le groupe de normes 802.11 décrites par un standard international. Dans la pratique, cette norme offre la possibilité de relier un grand nombre de périphériques (ordinateurs, machines, téléphones) à un réseau qui peut atteindre plusieurs centaines de mètres, voire dizaines de kilomètres pour le WiMAX. De nos jours, presque tous les ordinateurs et téléphones portables sont équipés de dispositif Wi-Fi et les autres peuvent en bénéficier grâce à des cartes externes.

En 2002, la technologie WAP, vieillissante et peu performante, laisse sa place à une version WAP 2 conçue pour l'UMTS (ou 3G) et qui délaisse le format WML au profit de *XHTML Mobile Profile* (XHTML MP), une version simplifiée de XHTML.

Quelques années plus tard, en 2008, le marché est inondé par un OVNI qui va révolutionner notre approche et notre usage de l'Internet mobile et devenir l'ancêtre d'une génération de téléphones intelligents : l'iPhone d'Apple. Rien que sur le dernier trimestre 2009, Apple annonce un chiffre de 8,7 millions d'iPhone vendus, talonné de près par Blackberry et Android.

Statistiques d'usage mobile

Le marché des internautes nomades est en pleine expansion. Selon l'institut de sondages Médiamétrie, il y aurait 13 millions de mobinautes en France en 2010, soit une progression exponentielle de plus de 20 % en un an. Un autre sondage émanant de l'ARCEP (Autorité de régulation des communications électroniques et des postes) annonce un chiffre de 18 millions d'internautes mobiles et des projections envisagent une majorité de Français utilisant le Web mobile d'ici deux ans.

En outre, 20 % d'utilisateurs de téléphones mobiles âgés de 15 ans et plus seraient équipés d'un smartphone, c'est-à-dire d'un téléphone « intelligent » destiné, entre autres, à surfer sur le Web et échanger des courriers électroniques.

D'une manière plus globale, 8 Français sur 10 utilisent aujourd'hui un téléphone mobile (soit 43,1 millions d'individus en 2010).

Ces chiffres éloquents témoignent d'un véritable phénomène de société qu'est la mobilité dans son ensemble, accompagnée d'usages nouveaux. Le terminal nomade devient petit à petit un eldorado où s'engouffrent de nombreux prestataires de services.

Navigateurs mobiles

L'opportunité du Web mobile a été saisie très vite par l'ensemble des constructeurs téléphoniques sur le marché. Chacun, ou presque, développait son propre système d'exploitation et son propre navigateur, au sein d'un flou juridique et « standardistique » émanant de ces usages révolutionnaires.

Depuis quelques années, des conventions se créent petit à petit. Certains vendeurs survivent à la vague, d'autres non. C'est également le cas pour leurs navigateurs et moteurs de rendu.

Systèmes d'exploitation mobiles

Chaque téléphone portable est articulé autour d'un système d'exploitation (ou OS, pour *operating system*) qui fait vivre toutes les couches et fonctionnalités de l'instrument. Dans la plupart des cas, le logiciel navigateur, qui va être employé pour surfer sur le Web, est directement imbriqué au sein du système de navigation, mais il existe aussi des navigateurs mobiles communs à plusieurs plates-formes, que l'on peut télécharger et utiliser librement. Le principe est semblable à une configuration PC Windows, qui embarque par défaut le navigateur Internet Explorer, mais qui peut accepter d'autres navigateurs alternatifs.

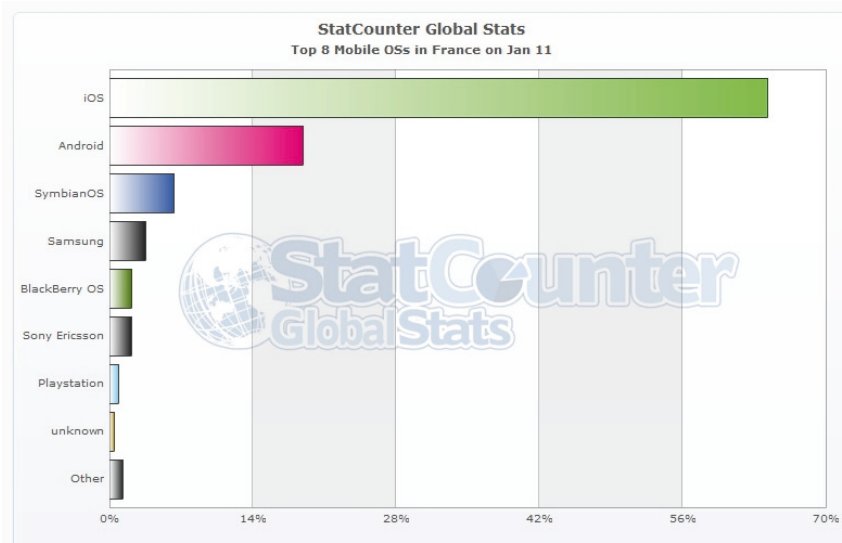
Vous vous doutez bien que peu de gens modifient leur configuration par défaut. C'est pourquoi il est intéressant d'observer quels sont les systèmes d'exploitation les plus usités. Selon les statistiques du site mondial gs.statcounter.com datant de janvier 2011, le marché des systèmes d'exploitation se répartit comme indiqué dans le tableau 9-1 et sur la figure 9-2.

Tableau 9-1 Statistiques des systèmes d'exploitation mobiles

Système d'exploitation	France	Europe	Monde
IphoneOS (Apple)	64,40 %	45,00 %	25,10 %
Android (Google)	18,90 %	16,20 %	14,50 %
Symbian (Nokia)	6,30 %	12,60 %	30,00 %
SonyEricsson	2,10 %	3,40 %	3,90 %
Samsung	3,50 %	2,40 %	4,30 %
RIM (Blackberry)	2,10 %	16,70 %	15,00 %
Playstation	1,00 %	<1 %	<1 %
Windows Mobile	<1 %	1,00 %	<1 %
Autres	1,50 %	3,00 %	7,00 %

Figure 9-2

Systèmes d'exploitation mobiles en France en janvier 2011



Navigateurs web et moteurs de rendu

Nous voici au cœur de la problématique du Web mobile : les navigateurs et leurs différences dans l'interprétation des langages et le rendu à l'écran. Souvent liés au système d'exploitation, certains navigateurs sont communs à plusieurs plates-formes. Pire, les fabricants de téléphones changent parfois de navigateur par défaut au fil des versions de leurs produits.

Ces contraintes rendent difficile la tâche d'organiser tout ce petit monde au sein d'une simple liste récapitulative. Le tableau 9-2 condense ce que j'ai pu compulser durant mes recherches. Il ne se veut pas exhaustif et est susceptible d'évoluer dans le temps.

Tableau 9-2 Constructeurs, marques et navigateurs mobiles

Constructeur	Marques	Système	Navigateur	Moteur
Apple	iPhone	iPhone OS CPU OS	Safari	Webkit
	iPod			
	iPad			
RIM	Blackberry 7***	Blackberry	Blackberry	Mango
	Blackberry 8***			
	Blackberry 9***			
	Blackberry Bold			
	Blackberry Torch	Blackberry	Safari	Webkit
Google	Nexus One	Android	Chrome	Webkit

Constructeur	Marques	Système	Navigateur	Moteur
HTC	***	Windows mobile	IE mobile	Trident
	Desire	Android	Chrome	Webkit
	Legend			
	Magic			
	Hero			
	Smart			
	Tattoo			
	Incredible			
LG	LG	LG	LG	LG
Motorola	Motorola	Motorola	Motorola	Motorola
	Motorola Droid	Android	Chrome	Webkit
	Motorola Motoroi			
	Motorola Cliq			
Nokia	Nokia ***	Symbian	Safari	Webkit
	Nokia N900	Maemo	Firefox	Gecko
Palm	Treo	webOS	Blazer	NetFront
	Pre	webOS	Safari	Webkit
	Pixi	webOS	Safari	Webkit
Samsung	Samsung ***	Samsung	Netfront, Sams.	NetFront
	Samsung SGH*	Windows mobile	IE mobile	Trident
	Samsung Galaxy	Android	Chrome	Webkit
	Samsung Galaxy S	Android	Chrome	Webkit
SonyEricsson	***	SonyEricsson	SonyEricsson	NetFront
	X10	Android	Chrome	Webkit
Sony	Playstation			NetFront
Nintendo	Wii		Opera	Presto
Amazon	Kindle			NetFront

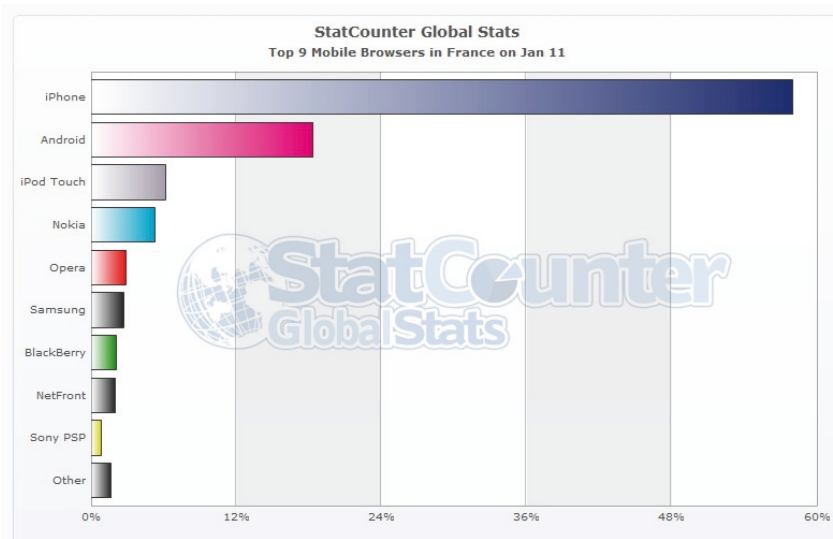
Les statistiques, cette fois bien plus intéressantes, dévoilent la hiérarchie suivante en France, en Europe et dans le monde au mois de janvier 2011 (tableau 9-3 et figure 9-3).

Tableau 9-3 Statistiques des navigateurs mobiles

Navigateur	France	Europe	Monde
Safari (iPhone, iPod, iPad)	64,10 %	44,80 %	25,00 %
Android browser	18,30 %	15,40 %	14,20 %
Nokia browser	5,30 %	7,50 %	15,50 %
Opera Mobile	2,90 %	10,00 %	21,00 %
Netfront	2,00 %	1,80 %	3,80 %
Blackberry browser	2,00 %	16,50 %	14,90 %
Samsung	2,00 %	1,00 %	1,70 %
Playstation	1,00 %	<1 %	<1 %
SonyEricsson	<1 %	<1 %	<1 %
Autres (dont IE mobile)	1,60 %	1,60 %	3,20 %

Figure 9-3

Navigateurs mobiles en France en janvier 2011



TECHNOLOGIE MOBILE **Mais que fait Microsoft ?**

Les terminaux sous environnement Windows Phone sont actuellement largement minoritaires (voire négligeables) au sein de ce marché très fluctuant de la téléphonie mobile. Le nouveau système d'exploitation Windows Phone 7, lancé à la fin de l'année 2010, pourrait hypothétiquement renverser la vapeur… ce qui ne représenterait pas forcément une bonne nouvelle : la version de son navigateur se situe à mi-chemin entre IE7 et IE8, la technologie Flash en est bannie et il n'est pas possible d'y installer un navigateur alternatif tel que Firefox Mobile (Fennec) ou Opera Mobile, mais l'évolution vers IE9 est prévue à la fin de l'année 2011.

Si votre marché ne concerne pas que l'Europe, sachez que les chiffres sont bien différents selon les zones. En Amérique du Nord, par exemple, sur la même période, iPhone, Android et Blackberry sont au coude à coude, avec une petite avance pour ce dernier (25 %). Et au niveau mondial, il semblerait que le tenant du titre soit… Opera Mobile !

Pour finir, soyez conscient que ces différents chiffres sont en constante évolution, au vu des guerres de marché que se font les constructeurs : ainsi les parts acquises par le moteur Android de Google sont en progression de plus de 3 % par mois depuis la fin de l'année 2010.

TECHNOLOGIE MOBILE **Acid Test chez les mobiles**

Les dispositifs mobiles ont également été soumis aux rudes traitements de l'Acid Test 3. Les résultats sont très positifs pour les moteurs de rendu WebKit (entre 93/100 et 100/100 de conformité), Gecko (97/100) et Presto (98/100), mais il sont moins reluisants pour Netfront (entre 11/100 et 27/100) et Internet Explorer Mobile 7 (12/100).

« Penser mobile »

Philosophie de conception

Surfer sur un téléphone mobile procure une expérience utilisateur qu'on ne peut comparer à l'utilisation classique d'Internet sur un ordinateur de bureau. Pour commencer, la taille limitée du terminal requiert un contenu et une navigation adaptés : pas question d'espérer agir sur le comportement au survol d'un lien ! De notre point de vue d'intégrateur, cela nécessite une prise en compte des éventuels débordements de contenu et une gestion des éléments de grandes tailles en amont du projet.

En outre, la vitesse de connexion (Wi-Fi, 3G/UMTS, voire Edge) est bien inférieure aux technologies ADSL ou câblées démocratisées sur les postes fixes. Les notions de poids, de temps de chargement et de performances en deviennent essentielles et cela influe sur le choix des outils mis en œuvre (JavaScript, Flash, Ajax…).

Pour finir, rappelons que les finalités sont très distinctes : un utilisateur qui visite un site sur son téléphone portable n'a pas le même objectif qu'un sédentaire sur son ordinateur de bureau. Parmi ces usages différents les plus notables, nous trouvons la géolocalisation, le *multitouch* (toucher multiple) et la possibilité d'être continuellement dans un flux d'informations ou de services (trouver un restaurant, un camping ou un petit travail à proximité).

En fin de compte, ces considérations philosophiques nous amènent à la conclusion qu'il est inévitable d'inclure la dimension mobile dès l'élaboration du cahier des charges de notre projet de site web sous peine de prendre le risque d'offrir une expérience utilisateur désastreuse pour les clients nomades.

Selon l'objectif de votre site web, des services qu'il propose et du public qu'il compte toucher, trois pistes s'offrent à vous si vous souhaitez transposer l'expérience web classique sur un téléphone mobile.

- Une présentation adaptée au support mobile : votre site web est déjà en ligne et vous souhaitez simplement profiter de la gestion des périphériques en CSS pour proposer des adaptations visuelles au terminal mobile (tailles des contenus, largeurs fluides).
- Un site web dédié : vous choisissez de créer un site différent pour chacun des périphériques de sortie en recréant une structure HTML (ou XHTML MP), en allégeant les contenus et le poids des éléments pour les mobiles.
- Une application native : vous optez pour la programmation d'un logiciel qui sera référencé et téléchargeable sur les plates-formes (*stores*) des constructeurs.

Cet ouvrage se limite – vous l'aurez compris à son titre – à la première piste évoquée. Si vous souhaitez explorer plus loin la conception web pour les mobiles ainsi que le développement d'applications, je vous recommande la lecture des ouvrages suivants.

ALLER PLUS LOIN Bien développer pour le Web mobile

F. Daoust, D. Hazaël-Massieux, *Relever le défi du Web mobile : Bonnes pratiques de conception et développement*, Eyrolles, 2011

E. Sarrion, *XHTML/CSS et JavaScript pour le web mobile*, Eyrolles, 2010

J. Stark, *Applications iPhone avec HTML, CSS et JavaScript*, Eyrolles, 2010

J. Chable, D. Guignard *et al.*, *Programmation Android*, Eyrolles, 2010

Détecter le terminal mobile

Vouloir s'adapter aux terminaux nomades se heurte généralement à un obstacle de taille : il n'existe pas *un* type de mobile, mais quasi autant que de modèles de téléphones. Fort de ce constat, je peux d'ores et déjà vous annoncer qu'il ne sera pas raisonnablement envisageable de vous ajuster à l'ensemble des portables existants, qu'il vous faudra choisir vos cibles et tenter de les détecter au préalable… dans la mesure du possible.

Un consensus difficile

Les générations de portables s'enchaînent à un rythme effréné : la course technologique nous façonne sans cesse des appareils toujours plus intelligents, plus rapides et plus ergonomiques, tandis que le taux de renouvellement des mobiles est estimé à environ 20 mois en moyenne.

Si la dernière génération en date offre des spécifications techniques affriolantes (*multitouch*, hautes résolutions de plus de 640 px), on rencontre encore des portables aux écrans minuscules et fonctionnant via WAP exclusivement. Ajoutons à cela une pluralité de systèmes d'exploitation et des moteurs de rendu disparates (WebKit, Presto, Gecko, Blackberry…) qui témoignent d'une guerre de marché endiablée.

Cette bataille s'accompagne de stratégies fluctuantes : refus d'employer certaines technologies (Flash, par exemple) chez certains constructeurs, changement de navigateur par défaut selon les modèles de téléphones chez Nokia, passage à un nouveau moteur de rendu (WebKit) chez Blackberry à partir de son OS 6, etc.

Au final, malgré les standards et spécifications proposés, nous sommes encore loin d'une homogénéisation du monde mobile et allons devoir déployer une batterie de tests dans l'optique de cibler le plus grand nombre.

À la question « comment m'assurer que mon site web s'affichera correctement sur tous les terminaux web existants ? », la réponse sera inexorablement que vous ne le pourrez pas !

Méthodes de détection

Comment déterminer que notre site web est visionné sur un terminal mobile ? C'est une question que l'on va irrémédiablement devoir se poser dans notre démarche d'adaptation. Les technologies actuelles nous proposent trois pistes à creuser : le média `handheld`, détecter la résolution du terminal, ou encore obtenir le nom de modèle de l'appareil.

Le média handheld

Les spécifications CSS 2, en 1998, offrent la possibilité de véhiculer la mise en forme d'un site web via différents supports, dont les mobiles via la valeur `handheld`.

Une feuille de styles adaptée pourrait être incluse de la manière suivante :

```
<link rel="stylesheet" type="text/css" media="handheld" href="mobile.css">
```

Dans un monde parfait, cela suffirait à notre bonheur, puisque nous pourrions ainsi cibler uniquement les terminaux mobiles et leur fournir un visuel spécifique.

Hélas, les smartphones actuels – à l'exception des heureux bénéficiaires du navigateur Opera Mobile – ne se reconnaissent pas dans ce média déshonorant ! Pire, ils se considèrent comme des écrans à part entière, répondant uniquement au média `screen`, à l'instar de nos écrans de bureau ; `handheld` sera en pratique typiquement reconnu par les téléphones munis de petits écrans, monochromes et à bande passante limitée.

JavaScript et la largeur du terminal

Puisque la valeur `handheld` ne trouve grâce qu'auprès des très anciennes générations de téléphones mobiles, une autre piste à explorer est de détecter la résolution du terminal à l'aide de la technologie JavaScript.

Si la largeur de l'écran observée via JavaScript correspond à ce que nous avons classé dans la catégorie mobile, il vous sera possible d'orienter le site vers une version dédiée (par exemple m.monsite.com) ou alors de vous contenter d'appliquer une feuille de styles CSS adaptée.

Le code à appliquer peut se révéler très simple. Voici un exemple qui redirige le visiteur vers l'adresse m.monsite.com si la largeur de son écran est inférieure ou égale à 640 px :

```
<script type="text/javascript">
  if(screen.width <= 640)
  window.location="m.monsite.com";
</script>
```

Le script ci-après crée un appel vers la feuille de styles `mobile.css` uniquement lorsque l'écran de restitution est de 640 pixels ou moins.

```
<script type="text/javascript">
  if(screen.width <= 640) {
  var lien_css = document.createElement('link');
  lien_css.href = "mobile.css";
  lien_css.rel = "stylesheet";
  lien_css.type = "text/css";
  document.getElementsByTagName("head")[0].appendChild(lien_css);
  }
</script>
```

Ces techniques sont plutôt aisées et rapides à mettre en œuvre. Elles souffrent cependant d'un handicap gênant : tous les navigateurs mobiles ne disposent pas de JavaScript et certains l'ont désactivé par défaut. Les chiffres statistiques sont difficiles à glaner, mais sachez qu'un test via un langage de programmation du côté client, tel que JavaScript, risque de vous priver d'une petite partie de votre cible.

Détection de l'agent utilisateur (*User Agent*)

Chaque agent utilisateur dispose d'une sorte « d'empreinte digitale » qui le distingue des autres : lorsqu'un navigateur visite une page web, il dévoile une partie de ses métadonnées intimes, qui peuvent être extraites et interprétées par le serveur. Parmi ces métadonnées se trouve une variable appelée `User Agent` comportant des informations telles que le moteur de rendu et sa version.

Voici à quoi ressemblent ces données dans le cas de quelques terminaux mobiles célèbres (iPhone, iPad, Blackberry et HTC Desire) :

iPhone

```
Mozilla/5.0 (iPhone; U; CPU like Mac OS X; en) AppleWebKit/420+ (KHTML, like Gecko)
Version/3.0 Mobile/1A543a Safari/419.3
```

iPad

```
Mozilla/5.0 (iPad; U; CPU OS 3_2 like Mac OS X; en-us) AppleWebKit/531.21.10 (KHTML,
➥ like Gecko) Version/4.0.4 Mobile/7B334b Safari/531.21.10
```

Blackberry de la série 9xxx

```
Mozilla/5.0 (BlackBerry; U; BlackBerry 9800; en) AppleWebKit/534.1+ (KHTML, Like
➥ Gecko) Version/6.0.0.141 Mobile Safari/534.1+
```

HTC Desire

```
Mozilla/5.0 (Linux; U; Android 2.1-update1; de-de; HTC Desire 1.19.161.5 Build/ERE27)
➥ AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Mobile Safari/530.17
```

Les langages serveurs, tels que PHP, ASP.net ou encore Java, sont capables de reconnaître ces métadonnées et d'en extraire les informations dont nous pourrions avoir besoin.

En clair, il est possible de détecter n'importe quel type de terminal, voire sa version, et d'appliquer des correctifs, une feuille de styles spécifiques, voire une page HTML dédiée selon les cas de figure.

Voilà par exemple en PHP comment reconnaitre un iPhone en train de visiter votre page web :

```
<?php
if (strstr($_SERVER['HTTP_USER_AGENT'], 'iPhone'))
{  ici les instructions pour iPhone  }
?>
```

À partir du moment où l'agent utilisateur est identifié (ici, un mobile iPhone), rien ne nous empêche de provoquer un appel vers une feuille de styles CSS adaptée à ce terminal pour corriger ou modifier certains aspects graphiques.

Cette méthodologie de détection, très fréquemment employée, se heurte cependant à un double inconvénient :

- Il est nécessaire de spécifier nominativement chacun des agents à détecter. Or, la liste s'avère rapidement très longue avec l'ensemble des terminaux mobiles existants.
- Les navigateurs peuvent très facilement masquer ou altérer volontairement leur « empreinte digitale » : il vous est possible par exemple de faire passer votre Blackberry pour un Android en modifiant à la main son `User Agent`.

Sur ce second point, il existe par exemple une extension pour Firefox, nommée User Agent Switcher, qui vous permet d'alterner entre plusieurs identités :

▸ https://addons.mozilla.org/en-US/firefox/addon/59/

Les Media Queries en CSS 3

Le mécanisme des *Media Queries*, déjà évoqué dans le chapitre précédent, ne permet pas directement de détecter un mobile, mais d'appliquer une série de règles CSS spécifiques lorsque certaines conditions telles qu'une petite largeur d'écran sont réunies.

En d'autres termes, il est possible de proposer des mises en forme particulières pour les mobiles sans nécessiter de langage de développement tels que JavaScript ou PHP, tout simplement en s'appuyant sur la largeur de leur écran.

Voici un exemple appliquant une couleur de fond verte au corps de page uniquement lorsque la taille de l'écran est inférieure ou égale à 480 pixels de large :

```
@media screen and (max-width: 480px) {
  body {background: green;}
}
```

Cette méthode, que nous allons détailler tout au long de ce chapitre, possède de multiples avantages, le principal étant que les moteurs de rendu pour terminaux nomades reconnaissent presque tous ce type de syntaxe.

Les outils en ligne

Pour les concepteurs réfractaires aux langages JavaScript, PHP et CSS (ils existent, j'en ai vus !), certains services proposent des solutions sous forme de paquets « tout-en-un », librement téléchargeables en ligne.

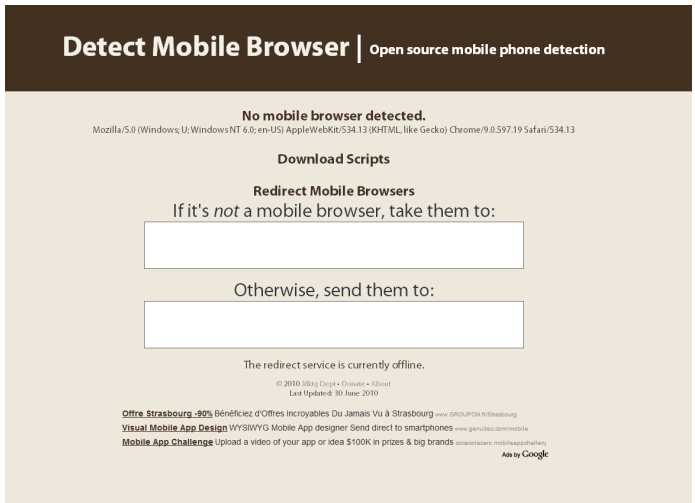
C'est le cas notamment des renommés :

- WURFL (*Wireless Universal Resource File*) : http://wurfl.sourceforge.net ;
- Detect Mobile Browser : http://detectmobilebrowser.com.

Le premier est une gigantesque base de données de tous les terminaux mobiles actuels et passés, tandis que le second est une application en ligne assez plaisante offrant la possibilité de détecter un mobile tout en choisissant son langage de prédilection : JavaScript, PHP, .htaccess, jQuery, ASP.net, Python, etc. Une fois votre langage de programmation élu, vous n'avez qu'à télécharger le code fourni par Detect Mobile Browser (figure 9-4) et à le coller sur votre page web. Cet outil est simple et efficace, mais difficile à modifier et à mettre à jour par vous-même.

Figure 9-4

DetectMobileBrowser.com



Tester sur mobile

Chaque téléphone dispose de singularités uniques : affichage, contraste, prise en main, confort d'utilisation, rapidité d'affichage, et bien d'autres aspects pratiques que l'on ne peut reproduire artificiellement.

Je ne vous apprendrai pas qu'il va vous être particulièrement ardu de tester physiquement le rendu de votre site web sur l'ensemble des smartphones existants. C'est tout bonnement impossible !

Si cet aspect vous tient à cœur ou fait partie de votre stratégie de communication, je vous invite à disposer, dans la mesure de vos moyens, de quelques téléphones représentatifs du marché ou de votre cible : un iPhone, un Blackberry ou un Android Phone. Pour les autres dispositifs, et même si rien ne remplace les tests sur un véritable terminal mobile, sachez que des émulateurs existent.

Les émulateurs en ligne

Du fait de la pluralité des plates-formes et des différences des moteurs de rendu, il n'est pas toujours aisé de tester son site web sur périphérique mobile, sans compter le coût d'acquisition du matériel.

La liste des émulateurs en ligne s'allonge de jour en jour, mais ceux-ci se contentent malheureusement le plus souvent de redimensionner la page, sans tenir compte des spécificités inhérentes à chaque terminal : iphoney, iphonetester.com, l'extension Webdeveloper, viewlike.us, resizemybrowser et j'en passe.

Dans le cadre de l'adaptation mobile du site de ma société Alsacréations, j'ai pu tester en détail deux outils plus intéressants que les autres : ProtoFluid et EmulateurMobile. Il s'agit de deux applications en ligne très proches dans leur principe, qui est de simuler au mieux l'affichage de votre site web ou d'une URL locale sur divers terminaux mobiles :

- **ProtoFluid** (http://protofluid.com) est exclusivement anglophone et permet de choisir parmi les smartphones suivants : Motorola Droid, iPhone, iPad et Google Nexus One.
- **EmulateurMobile** (http://www.emulateurmobile.com) propose quant à lui les terminaux suivants : iPhone 3, iPhone 4, iPad, HTC HD 2, HTC Touch Diamond, LG U970, Blackberry 8900 et Samsung GT i5700 (figure 9-5).

Figure 9-5

Illustration de EmulateurMobile.com

Ces deux émulateurs en ligne sont à la fois simples et efficaces pour un test rapide : il suffit de renseigner l'URL à visionner ainsi que la plate-forme désirée, et sa résolution si vous souhaitez la modifier. Le résultat apparaît dans une fenêtre modale (*lightbox*). Il vous est ensuite possible de modifier l'orientation (portrait ou paysage) puis de rafraîchir votre résultat.

Pour l'avoir testé sur quelques-uns de nos sites web optimisés pour le mobile (métadonnée `viewport`, CSS Media Queries), tels que http://www.alsacreations.fr et http://www.ie7nomore.com, j'ai finalement pu en conclure que ces deux outils se rapprochent beaucoup plus du résultat escompté que leurs

congénères, même si le rendu reste tributaire du navigateur utilisé pour les consulter, car la simulation ne se fait qu'au travers d'un élément `iframe` et de JavaScript. En clair, il va sans dire qu'il vaudra mieux éviter d'y accéder avec Internet Explorer 6 pour obtenir des résultats concluants !

Comme tout outil de ce type, ProtoFluid et EmulateurMobile ne peuvent que « deviner » comment va se comporter le moteur de rendu final et s'y adapter à partir de votre navigateur de test. Cela signifie que certaines technologies présentes sur votre poste de bureau (Flash, par exemple) vont apparaître sur la simulation iPhone… ce qui ne risque pas d'arriver de sitôt sur un véritable terminal d'Apple.

De même, certaines règles CSS normalement désactivées sur de nombreux mobiles, telles que le positionnement fixé, vont tout de même être exécutées sur la simulation. Sachez tenir compte de ces incongruités dans vos tests.

Les kits officiels à télécharger

En dehors de l'installation des divers SDK officiels des constructeurs ou de l'excellente version d'Opera Mobile installable sur un système d'exploitation classique (Windows, Mac OS X...), il n'y a point de salut pour visualiser ses pages dans un environnement proche des conditions réelles.

Les différents kits officiels sont généralement libres et gratuits (à l'exception de celui d'Apple) et téléchargeables à ces adresses :

- Apple iPhone, iPod ou iPad : http://developer.apple.com/programs/iphone/
- Google Android : http://developer.android.com/sdk/
- RIM Blackberry : http://www.blackberry.com/developers/downloads/simulators/

OUTIL Télécharger Opera Mobile

Complexes à installer et coûteux en ressources, les SDK officiels ne sont pas des plus attrayants. À moins d'en avoir réellement besoin, il existe une solution alternative beaucoup plus simple à mettre en œuvre dans le cadre de tests rapides : Opera Mobile. Libre, gratuite et facile à installer sur toutes les plates-formes, la version mobile d'Opera va vite devenir votre première alliée pour tester vos pages web sur environnement mobile :

▸ http://www.opera.com/developer/tools/

Adapter pour les mobiles

Résolution native

Chaque téléphone portable dispose de ses propres caractéristiques techniques. Parmi celles-ci, la notion de zone occupée par l'écran est primordiale dans notre quête de mise en page adaptée sur mobile.

Ainsi, la définition de l'écran de l'iPhone 3, par exemple, est 320 × 480 pixels, ce qui laisse une surface d'écran affichable de 320 × 356 pixels. Même si son successeur offre plus de latitude

(640 × 960 pixels), nous sommes bien loin des définitions parfois supérieures à 1 900 px des écrans classiques de nos ordinateurs !

Les générations actuelles de téléphones portables intelligents offrent globalement un panel de définitions situé entre 240 px et 640 px de largeur, à la frontière avec les mini-PC de 10 pouces ou le récent iPad qui chamboule la notion même de Web mobile de par sa surface de 768 × 1 024 px.

Le viewport

Pour les smartphones, le *viewport* ne correspond pas à la partie visible de l'écran, mais à une dimension arbitraire et virtuelle qui équivaut à ce que le mobile va pouvoir disposer harmonieusement sur son écran. En pratique, si la valeur de viewport est de 980 px (c'est le cas pour iPhone 3 et 4 sur Safari, par exemple), la page web sera interprétée telle que le ferait un navigateur standard sur un écran de bureau de 980 pixels de large.

Bien entendu, un terminal de 320 × 480 pixels physiques ne pourra pas « inventer » de nouveaux pixels. L'affichage consiste en un redimensionnement proportionnel de la page sur grand écran.

Pour compliquer la vie des concepteurs web, chaque navigateur s'est fixé une taille de viewport par défaut… différente de celle des autres. Ainsi, le viewport de l'iPhone 3, de l'iPhone 4 et de l'iPad est de 980 px, celui d'Opera Mobile est de 850 px, celui d'Android est de 800 px et celui d'Internet Explorer Mobile est de 1024 px (figure 9-6).

Figure 9-6

Illustration du viewport laissé à discrétion du mobile

La métadonnée « viewport »

Dans l'optique de forcer la largeur du viewport – et par extension celle de la mise en page –, Apple a inventé un élément `<meta>` propriétaire : `<meta name="viewport" />`.

> **COMPATIBILITÉ viewport**
>
> Bien que propriétaire, cette instruction est plébiscitée par de nombreux constructeurs mobiles. Elle est par conséquent parfaitement reconnue sur un grand nombre de terminaux dont Safari Mobile (iPhone, iPad, iPod), Android, Opera Mini, Opera Mobile, Firefox mobile (Fennec), IE Mobile et les dernières générations de Blackberry.

Cette métadonnée HTML, à placer dans la partie `<head>` du document, offre la possibilité d'outrepasser la valeur intrinsèque du viewport en précisant la largeur souhaitée. La mise en page et le contenu s'adaptent alors à cette nouvelle dimension imposée.

Il est possible d'indiquer une valeur à cette balise `meta`, ce qui aura pour conséquence de redimensionner l'ensemble du visuel et d'adapter la taille des textes afin de les rendre plus lisibles.

Ainsi, les exemples suivants imposent à la page affichée sur mobile des largeurs de viewport respectivement non définie, de 320 px, 480 px puis 980 px (figure 9-7) :

Figure 9-7

Valeurs de viewport fixées par l'auteur

```
pas de meta viewport
<meta name="viewport" content="width=320" />
<meta name="viewport" content="width=480" />
<meta name="viewport" content="width=980" />
```

Ajoutez simplement l'instruction `<meta name="viewport" content="width=320">` dans votre code HTML et vous observerez que le rendu de votre site épouse automatiquement la largeur native d'un écran d'iPhone (320 px) et que ses textes s'agrandissent pour être plus lisibles sans devoir zoomer sur la page.

Bien évidemment, si votre téléphone mobile n'est pas un iPhone, ou ne dispose pas d'une surface d'affichage équivalente, cette expérience s'en trouvera malheureusement altérée.

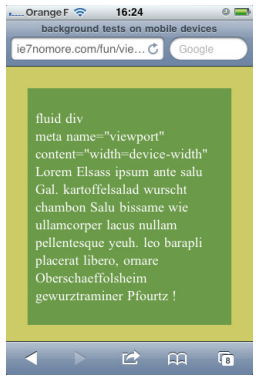
C'est principalement pour cette raison qu'il est préférable d'opter pour une valeur fluide et universelle, quel que soit le mobile récepteur. Cette valeur existe sous la forme de l'instruction suivante (figure 9-8) :

```
<meta name="viewport" content="width=device-width" />
```

La valeur `device-width` adapte harmonieusement la mise en page à la largeur physique du terminal, quelle qu'elle soit.

Figure 9-8

viewport identique au device-width



Un écran, plusieurs largeurs

La métadonnée HTML `<meta name="viewport">` introduit une nouvelle notion, qui est la largeur du terminal (`device-width`). Comme on peut le constater, la valeur de `device-width` est différente de celle du viewport par défaut, qui est dépendant du bon vouloir de chaque navigateur. Pour rappel, le viewport d'un iPhone sur le navigateur Safari est de 980 px tandis que sa largeur réelle est de 320 px.

Nous pourrions alors – logiquement – croire que la valeur `device-width` correspond à la largeur physique de l'écran du mobile. Or il n'en est rien. En effet, la génération iPhone 4 vient chambouler nos certitudes puisque sa largeur physique a été étendue à 640 px tout en conservant une valeur `device-width` de… 320 px !

Par ailleurs, rappelons que les terminaux nomades récents permettent de surfer sur le Web soit en mode vertical (portrait) soit en mode horizontal (paysage) et que leur largeur s'en trouve forcément modifiée.

En conclusion, un écran mobile, ce n'est pas une, mais six largeurs différentes selon l'interprétation et l'orientation !

Tableau récapitulatif

Le tableau 9-4 synthétise les différents paramètres de largeurs à prendre en compte sur les terminaux d'Apple.

Tableau 9-4 Un mobile, plusieurs dimensions !

Modèle	Viewport (Safari)	Résolution physique	device-width
iPhone 3 (portrait)	980 px	320 px	320 px
iPhone 3 (paysage)	980 px	480 px	480 px
iPhone 4 (portrait)	980 px	640 px	320 px
iPhone 4 (paysage)	980 px	960 px	480 px
iPad (portrait)	980 px	768 px	768 px
iPad (paysage)	980 px	1 024 px	1 024 px

Dans ce tableau, la valeur de viewport (980 px) est celle délivrée par le navigateur Safari d'Apple. Souvenez-vous toutefois que les autres plates-formes ont opté pour des valeurs différentes (850 px pour Opera Mobile, 800 px pour Android et 1 024 px pour IE Mobile, entre autres).

Gérer la largeur d'un site mobile

La section précédente témoigne d'un nombre de variables assez conséquent à prendre en compte lorsque l'on adapte son site pour le Web mobile. En ce qui me concerne, j'applique une méthodologie plutôt simple, robuste et relativement universelle pour la génération actuelle de terminaux nomades. Elle tient en quatre points.

Étape 1 : ne s'occuper de rien

Aussi curieux que cela puisse paraître, et malgré cette jungle de paramètres différents, les sites web s'affichent généralement plutôt correctement sur les écrans mobiles modernes, à condition bien entendu d'éviter des technologies ou plug-ins risqués tels que Shockwave Flash.

Cette bonne nouvelle s'explique par la présence du viewport inhérente à chaque navigateur. Nous avons vu que sur Safari mobile, la valeur du viewport est par défaut de 980 pixels. Cela signifie qu'un site classique d'une largeur de moins de 980 px devrait être redimensionné harmonieusement et proportionnellement à l'écran, tout cela sans agir sur aucun paramètre évoqué précédemment.

Cette méthode porte ses fruits dans de nombreux cas de figure. Cependant, les écueils en termes d'accessibilité et d'expérience utilisateur demeurent nombreux et difficiles à négocier :

- un texte illisible car trop petit ;
- un contenu noyé dans un multicolonnage parasite ;
- des textes ou éléments débordant des blocs (images, liens hypertextes…) ;
- un menu de navigation non adapté à un petit écran ;
- des sections superflues (bandeaux publicitaires, grandes illustrations…), etc.

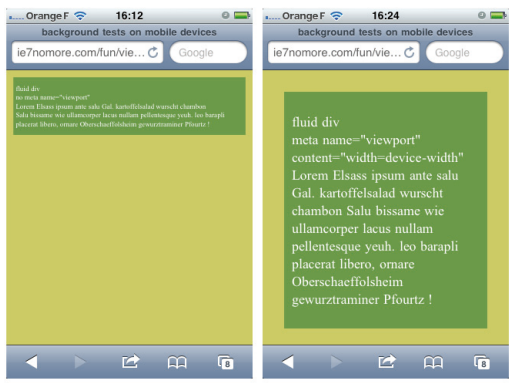
Étape 2 : forcer une base commune

Le meilleur moyen de s'affranchir de ces multiples dangers potentiels est de préciser une base commune à la plupart des mobiles, à savoir la métadonnée `<meta name="viewport" content="width=device-width" />` décrite précédemment.

Cette étape a pour principal avantage d'éviter le redimensionnement automatique de la mise en page, qui rend les contenus trop petits, de fixer la largeur du mobile et de pouvoir s'y adapter par la suite. Cette instruction court-circuite le viewport par défaut, souvent bien trop large, et fournit une base commune plus proche de la réalité (figure 9-9).

Figure 9-9

Une base fluide pour bien débuter



La contrepartie de cette redéfinition de largeur se fait généralement ressentir immédiatement : par défaut, l'ensemble des éléments est redimensionné automatiquement dans le but de s'intégrer harmonieusement au sein d'un large espace de travail (entre 800 px et 980 px). Ce n'est plus le cas lorsque l'on outrepasse ce comportement via la métadonnée HTML `viewport` : il faut dorénavant gérer au cas par cas tous les contenus de largeur supérieure à la surface physique du terminal sous peine de débordement ou de dégradation de la présentation (figure 9-10).

En clair, si je définis un `<meta name="viewport" content="width=device-width">` pour m'adapter à tout type de mobile, je vais forcément devoir réinterpréter toutes les largeurs de mes éléments, afin qu'elles ne soient pas supérieures à cette valeur, car ceux-ci ne seront pas redimensionnés.

Figure 9-10

Une image supérieure à la largeur physique de l'écran

Étape 3 : affiner avec les Media Queries

Nous retrouvons une fois de plus le mécanisme CSS 3 Media Queries déjà présenté à plusieurs reprises dans cet ouvrage.

Au sein de la feuille de styles écran classique, ou dans un fichier CSS séparé, les Media Queries vont cibler très précisément le type de terminal qui nous intéresse tout particulièrement, le téléphone mobile, et lui appliquer des styles spécifiques.

Parmi les missions possibles des styles définis via Media Queries, on trouvera la correction de tous les défauts visuels dus à la redéfinition de l'espace de travail. Cela peut consister à rendre fluides les dimensions de tous les éléments afin de préserver des éventuels débordements, à agrandir la taille de certains contenus dans le but d'améliorer leur lisibilité ou encore à rendre les menus de navigation plus intuitifs.

COMPATIBILITÉ Media Queries

Safari, Firefox, Opera, Android et la dernière génération de Blackberry (disposant de la version 6 du système d'exploitation) gèrent tous les Media Queries.

Internet Explorer mobile et les anciens Blackberry sont les seuls à ne pas les prendre en considération à l'heure actuelle. Sauf si votre cible est très spécifique, ces plates-formes peuvent être considérées comme négligeables. Nous proposerons toutefois une astuce intéressante à la fin de ce chapitre pour tenir compte d'Internet Explorer Mobile.

Nous emploierons les CSS Media Queries dans le but de redéfinir les styles appliqués au périphérique écran classique pour une adaptation au terminal mobile.

Pour ce faire, toutes les règles CSS problématiques (notamment les dimensions d'éléments trop importantes) seront écrasées par des règles spécifiques appliquées uniquement sur les petits écrans. Ce principe général est illustré par l'exemple suivant.

Feuille de styles CSS

```
body {width: 960px;}

@media screen and (max-width: 480px) {
  body {width: auto;}
}
```

Cet exemple doit être interprété de la sorte : une première règle fixe la largeur du corps de page (`body`) à 960 px, pour un affichage sur un écran classique. Puis, pour les terminaux de largeur inférieure ou égale à 480 pixels, une règle CSS conditionnelle (*Media Query*) redéfinit la largeur de `body` à `auto`, c'est-à-dire occupant tout l'espace libre disponible.

Quelques exemples de Media Queries

```
@media screen {...}
/* cible uniquement les supports « écran » (dont smartphones modernes) */
@media screen and (max-device-width: 480px) {...}
/* cible les iPhone dans les deux orientations */
@media screen and (max-device-width: 1024px) {...}
/* cible les iPhone et iPad dans les deux orientations */
@media screen and (min-device-width:481px) and
(max-device-width: 1024px) {...}
/* cible uniquement les iPad dans les deux orientations */
@media screen and (min-device-width: 481px) and
(max-device-width: 1024px) and (orientation:portrait) {...}
/* cible uniquement les iPad en orientation   verticale */
```

BONNE PRATIQUE max-width ou max-device-width ?

Employer `max-width` plutôt que `max-device-width` a pour avantage de faciliter vos tests en direct puisque cette condition s'appliquera également à votre écran de bureau : vous pourrez ainsi tester vos règles CSS simplement en redimensionnant la taille de la fenêtre du navigateur !

Étape 4 : une astuce pour ne pas oublier Internet Explorer

Depuis la version 7 de Windows Phone, il est possible de profiter d'une technique pour cibler spécifiquement les mobiles équipés de ce système, afin de pouvoir fournir des feuilles de styles ajustées au média. Cette astuce consiste à exploiter un nouveau commentaire conditionnel, `IEMobile`, créé en 2011 par Microsoft.

Voici par exemple comment appliquer la feuille de styles `mobile.css` pour les périphériques équipés d'Internet Explorer Mobile :

```
<!--[if IEMobile]>
  <link rel="stylesheet" media="screen" href="mobile.css" type="text/css" />
<![endif]-->
```

Media Queries et performances

L'un des inconvénients inhérents à cette méthode d'affinage à l'aide des Media Queries est que les styles de base communs s'appliquent tout d'abord dans leur intégralité avant d'être corrigés sur les petits terminaux.

Par conséquent, les images et fichiers des directives générales sont toujours téléchargés au préalable, même s'ils sont masqués ensuite. Dans l'exemple suivant, le fichier `concombre.jpg` n'apparaîtra pas sur un écran de petite résolution, mais aura tout de même été chargé en mémoire :

```
div {background-image: url(concombre.jpg);}

@media screen and (max-width: 640px) {
  div {background-image: none;}
}
```

La solution la plus évidente et intuitive consiste à appliquer une autre règle Media Queries consacrée exclusivement aux écrans larges et incluant les différents fichiers non destinés aux mobiles :

```
@media screen and (min-width: 641px) {
  div {background-image: url(concombre.jpg);}
}
@media screen and (max-width: 640px) {
  div {background-image: none;}
}
```

Le problème est que les anciennes versions d'Internet Explorer (jusqu'à IE8 inclus), ne reconnaissant pas les Media Queries, ne vont interpréter aucune de ces deux règles et ne vont rien afficher du tout ! Il faudra alors se tourner vers des astuces de type JavaScript telles que celles décrites précédemment.

Dans la pratique, je vous suggère trois alternatives si votre version web mobile est une priorité pour vous :

- Oubliez les Media Queries et optez pour des solutions JavaScript ou en langage serveur pour masquer des directives selon les terminaux.
- Évitez de prévoir un projet très graphique dès le cahier des charges.
- Faites une version dédiée (m.monsite.com) ou une application native.

Particularités iPhone

À l'aide de quelques méta-éléments HTML propriétaires, il est possible de réduire l'écart existant entre une application native et une page web dédiée au mobile, lorsque celle-ci est accessible depuis le bureau de l'iPhone.

Une icône sur le bureau

La valeur `apple-touch-icon` appliquée au sein de l'élément `<link>` permet de lier une image apparaissant sous forme d'icône sur le bureau :

```
<link rel="apple-touch-icon" href="nom_fichier.png" />
```

L'icône en question doit être d'au moins 57 × 57 pixels, sans bords arrondis, car ils seront automatiquement affichés de façon arrondie par le système d'exploitation de l'iPhone.

Supprimer la barre d'adresse

Il est possible de supprimer par défaut l'affichage de la barre d'adresse du navigateur, via l'élément `<meta>` suivant :

```
<meta name="apple-mobile-web-app-capable" content="yes" />
```

Notez que cette instruction doit être inscrite dans la page HTML avant que l'icône ne soit créée sur le bureau de l'iPhone, sous peine de ne pas être prise en compte. Pensez donc à l'insérer dès le début de la création de votre page.

Une image au démarrage

Une image d'accueil peut être spécifiée lors du chargement de votre site web. Cette procédure requiert la valeur `apple-touch-startup-image` et doit elle aussi être située dans le code source avant l'appel à l'image d'icône de bureau :

```
<link rel="apple-touch-startup-image" href="nom_fichier.png" />
```

Méthodologie et étude de cas concret

Afin de mettre en application tous les préceptes évoqués précédemment, intéressons-nous à un cas concret en production réelle : la version mobile du site de l'agence web Alsacreations.fr.

Comme les figures 9-11 et 9-12 en témoignent, un certain nombre de mesures ont été prises pour adapter l'affichage à un terminal de taille réduite.

Figure 9-11

Alsacreations.fr
version écran classique

Figure 9-12

Alsacreations.fr sur mobile

Meta viewport

Notre première mission, afin d'apporter un confort de lecture idéal, consiste à définir la largeur de viewport de telle sorte qu'elle s'adapte automatiquement à celle du terminal.

Cette instruction, que vous connaissez bien à présent, se place dans la partie `<head>` du code HTML :

```
<meta name="viewport" content="width=device-width">
```

La suite des événements implique une détection des téléphones mobiles. Par choix, j'ai fait l'impasse sur les langages JavaScript et PHP pour me consacrer exclusivement aux méthodes via Media Queries CSS 3. En résumé, je néglige volontairement les anciennes générations de navigateurs ou les rares qui ne comprennent pas les Media Queries (Windows mobile, LG et Blackberry avant la version 6 du système d'exploitation).

J'ai également pris la décision de considérer un périphérique comme mobile à partir du moment où sa taille était inférieure ou égale à 640 pixels.

Dans la pratique, je place la règle suivante en fin de page CSS :

```
@media screen and (max-width:640px)  {
  /* ici les règles pour mobiles */
}
```

Redimensionnement des éléments

Redéfinir le viewport a des implications importantes et immédiatement visibles sur l'affichage : tous les éléments dont la largeur est supérieure à la résolution de l'écran du terminal dépassent ou poussent les autres éléments. En clair, tout est déformé !

Il va donc falloir dénicher une à une toutes les propriétés affectant les dimensions de boîtes (`width`, mais aussi `min-width`, `padding`, `margin` et `border`) dans l'ensemble de la feuille de styles et les remplacer par des valeurs fluides (`%`) ou par défaut (`auto` ou `0`).

Tout cela devra être fait en précisant parfois le mot-clé `!important` afin de systématiquement passer en priorité par rapport aux déclarations précédemment appliquées au sein des styles « classiques ».

Concrètement, pour le site Alsacreations.fr, cela s'est traduit par des règles de ce type :

```
#references_home figure,#apropos_home #alsacom,#projet #etapes li,#contact #submit
➥ input,#contact aside,#contact .social,blockquote p,#apropos_home #presentation {
  width:auto !important
}
#competences a + a,.intro h1,.intro h2 {
  margin-left:0
}
```

Il est également salutaire de fixer une largeur maximale (`max-width`) de 100 % aux éléments potentiellement problématiques (`<img>`, `<table>`, `<td>`, `<blockquote>`, `<object>`, `<embed>`, `<video>`, `<input>` et `<textarea>`) pour s'assurer qu'ils ne puissent jamais être plus larges que leur parent :

```
img, table, td, blockquote, code, pre, textarea, input, object, embed, video {
  max-width: 100%;
}
```

Empêcher les débordements

L'étape suivante consiste à se prémunir contre tout débordement de contenu (figure 9-13). Certains éléments sont particulièrement sujets à ces comportements indésirables, par exemple les codes source et les cellules de tableaux :

```
textarea, table, td, th, code, pre, samp {
  word-wrap: break-word; /* césure forcée */
  white-space: pre-line; /* passage à la ligne spécifique pour les éléments
  ➥ à châsse fixe */
}
```

Figure 9-13

Exemple de débordement de contenu

Redéfinition des tailles de polices

Il est fréquent de devoir adapter au monde mobile l'ensemble des tailles de polices de contenus, mais aussi et surtout des tailles des titrages et des éléments de formulaires. Et cela est d'autant plus vrai lorsque le terminal est tenu en orientation paysage.

L'emploi de la propriété `font-size` n'est malheureusement pas toujours efficace pour réduire ou agrandir de façon harmonieuse toutes les tailles de polices. C'est là qu'intervient une propriété pour l'instant spécifique aux moteurs Safari et Android, `-webkit-text-size-adjust`.

Voici comment réduire toutes les tailles de façon cohérente uniquement en mode paysage (`landscape`) :

```
@media screen and (max-width:640px) and (orientation: landscape) {
  body {
  -webkit-text-size-adjust: 70%;
  }
}
```

Une seule colonne

L'art de métamorphoser une mise en page classique en version pour mobile réside souvent dans un tour de magie efficace : transformer toutes les colonnes multiples en des blocs empilés verticalement afin de n'afficher qu'une seule colonne.

Généralement, cela se traduit par une redéfinition des déclarations `float: left` ou `float: right` en `float: none` accompagnée d'une largeur (`width`) automatique.

Notre site d'exemple, Alsacreations.fr, n'a pas échappé à ce précepte et des aménagements ont dû être réalisés en ce sens :

```
section.formation .formation_plan,section.formation .formation_info {
  float:none !important;
  width:auto !important
}
```

Supprimer le superflu

Un grand nombre d'éléments considérés comme utiles sur un écran de bureau peuvent se révéler imposants voire totalement inutiles sur une petite surface d'affichage.

À l'aide de `display: none`, il est possible de masquer complètement les éléments jugés non nécessaires à la navigation ou à la compréhension du document :

```
#slideshow, #team, #apropos_home aside, #references_home, #projet {
  display:none!important
}
```

Optimisation de la navigation

L'expérience de navigation sur mobile (figure 9-14) est toujours différente de celle sur un écran classique (figure 9-15). N'hésitez pas à refondre intégralement le système de navigation principal et à l'afficher de façon optimisée pour ce terminal spécifique.

Pour ma part, j'ai choisi de disposer les liens du menu de façon verticale, de les agrandir et de modifier le positionnement des pictogrammes associés :

```
#competences nav a {
  display:block!important;
  font-size:.8em !important;
  border:none;
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  border-radius: 10px;
  background-color:#abc !important;
  color:#fff !important;
  width:85% !important;
  height:40px !important;
```

```
  min-height:40px !important;
  line-height:40px !important;
  margin-bottom:3px !important;
  padding:6px 18px !important
}
```

Figure 9-14

Navigation sur écran

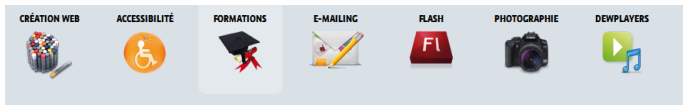


Figure 9-15

Navigation adaptée aux mobiles

Réorganisation des contenus

Grâce aux Media Queries, il est possible de transformer intégralement votre mise en page, ne l'oubliez pas. De plus, souvenez-vous que beaucoup de navigateurs mobiles comprennent très bien CSS 3.

Nous allons en profiter pour changer totalement l'ordre d'apparition des éléments sur la page : nous pouvons décider d'afficher la partie d'en-tête (`header`) en dessous de tous les autres éléments de la page grâce au modèle de boîte flexible en jouant avec la propriété `box-ordinal-group`.

```
body {
  display : -moz-box!important;
  display : -webkit-box!important;
  display : box!important;
  -moz-box-orient : vertical;
  -webkit-box-orient : vertical;
  box-orient : vertical;
}
#header {
  -moz-box-ordinal-group : 2;
  -webkit-box-ordinal-group : 2;
  box-ordinal-group : 2;
}
```

Message personnalisé

Les pseudo-éléments CSS 2.1 `:before` et `:after` offrent la possibilité de créer du contenu via CSS. Nous allons nous servir de cette opportunité pour afficher un message d'introduction uniquement sur la version mobile du site (figure 9-16) :

```
header h1:after {
  content: "Version mobile du site de l'agence web";
  display: block;
  padding: 5px 0 10px 0!important;
  font-size: 11px;
  color: #777;
  font-weight: normal;
  text-align: center;
  font-style: italic;
}
```

Figure 9-16

Un message d'introduction destiné aux mobiles

HTML 5 pour les champs de formulaire

Enfin, n'ayez aucune crainte d'employer à bon escient quelques touches de HTML 5 au sein de vos éléments de formulaires.

De plus en plus de navigateurs interprètent les champs de type `email` ou `tel` et proposent un clavier adapté (figure 9-17). Facilitez la vie de vos visiteurs sur mobiles !

```
…
  <p><label for="url">Site web : </label><input type="url" id="url"></p>
  <p><label for="email">E-mail : </label><input type="email" id="email"></p>
…
```

Figure 9-17

Champ de type e-mail sur iPhone

10

CSS pour l'impression

Souvent délaissé par manque de temps ou de connaissance, le périphérique d'impression fait pourtant partie du quotidien d'un concepteur de sites web. Découvrons à travers ce chapitre les caractéristiques de ce support, les propriétés CSS dédiées, les limites actuelles des navigateurs et les bonnes pratiques à mettre en place pour faciliter l'impression de ses documents web.

Pourquoi une feuille de styles pour l'impression ?

L'avantage d'un périphérique « print »

Nous avons parfois la fâcheuse habitude de penser que le Web n'est bon à être restitué que sur un écran d'ordinateur. Pourtant, un grand nombre de documents web et d'informations en ligne sont parfaitement adaptés à l'impression. Cela facilite non seulement leur consultation, mais aussi leur conservation (figure 10-1).

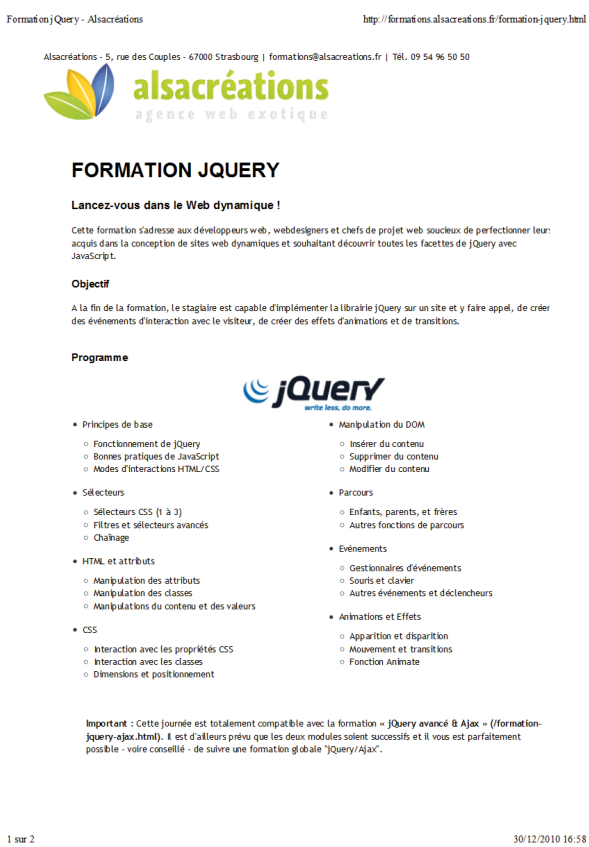
Le support imprimé offre des avantages non négligeables :

- conserver des informations (contact, plans, programme d'une formation par exemple) ;
- faciliter la lecture hors ligne (documents de contenus) ;
- disposer d'un document pouvant être annoté ou transmis physiquement.

Au cours de votre existence d'internaute, vous avez très certainement imprimé des écrits tels que des réservations de vol ou d'hébergement, des documents à signer ou autres confirmations de commandes passées dans une boutique en ligne. Ce genre d'expérience laisse généralement des souvenirs mitigés, car le résultat obtenu via la commande Fichier>Imprimer est souvent – et malheureusement – bien aléatoire. Le plus souvent, on se retrouve avec trois pages à imprimer, pour une seule vraie page de contenu.

Figure 10-1

Version imprimante d'une formation jQuery Alsacréations



Formation jQuery - Alsacréations http://formations.alsacreations.fr/formation-jquery.html

Alsacréations - 5, rue des Couples - 67000 Strasbourg | formations@alsacreations.fr | Tél. 09 54 96 50 50

alsacréations
agence web exotique

FORMATION JQUERY

Lancez-vous dans le Web dynamique !

Cette formation s'adresse aux développeurs web, webdesigners et chefs de projet web soucieux de perfectionner leurs acquis dans la conception de sites web dynamiques et souhaitant découvrir toutes les facettes de jQuery avec JavaScript.

Objectif

A la fin de la formation, le stagiaire est capable d'implémenter la librairie jQuery sur un site et y faire appel, de créer des événements d'interaction avec le visiteur, de créer des effets d'animations et de transitions.

Programme

jQuery
write less, do more.

- Principes de base
  - Fonctionnement de jQuery
  - Bonnes pratiques de JavaScript
  - Modes d'interactions HTML/CSS
- Sélecteurs
  - Sélecteurs CSS (1 à 3)
  - Filtres et sélecteurs avancés
  - Chaînage
- HTML et attributs
  - Manipulation des attributs
  - Manipulation des classes
  - Manipulations du contenu et des valeurs
- CSS
  - Interaction avec les propriétés CSS
  - Interaction avec les classes
  - Dimensions et positionnement
- Manipulation du DOM
  - Insérer du contenu
  - Supprimer du contenu
  - Modifier du contenu
- Parcours
  - Enfants, parents, et frères
  - Autres fonctions de parcours
- Evénements
  - Gestionnaires d'événements
  - Souris et clavier
  - Autres événements et déclencheurs
- Animations et Effets
  - Apparition et disparition
  - Mouvement et transitions
  - Fonction Animate

Important : Cette journée est totalement compatible avec la formation « **jQuery avancé & Ajax** » (**/formation-jquery-ajax.html**). Il est d'ailleurs prévu que les deux modules soient successifs et il vous est parfaitement possible - voire conseillé - de suivre une formation globale "jQuery/Ajax".

1 sur 2 30/12/2010 16:58

Certains sites proposent également des versions « imprimables », pages HTML épurées ou documents PDF, de certains de leurs contenus. Le problème, c'est qu'il est difficile de savoir à l'avance quelle page un utilisateur voudra imprimer. Par exemple, on propose rarement de fonctionnalité « Imprimer cet article » sur la page de contact… page dont le contenu est pourtant un bon candidat à l'impression !

Il est donc important de proposer une solution simple, intégrée au navigateur et fonctionnelle pour toutes les pages d'un site, car tous les documents destinés à être imprimés méritent un traitement de faveur minimal.

C'est là qu'interviennent les feuilles de styles CSS pour l'impression. Avec les CSS dédiées au périphérique `print`, nous allons permettre aux utilisateurs de véritablement profiter de la fonctionnalité d'impression des navigateurs.

Caractéristiques du format papier

Le Web sur écran n'est pas paginé, contrairement à l'impression sur papier. L'écran a ses propres règles, ses propres lois, qui s'expliquent très facilement : non seulement un écran peut avoir

différentes largeurs, mais surtout les concepts de « page » et de « fin de page » n'existent pas : un document web utilise un ascenseur.

Si la restitution d'un contenu web sur écran supporte peu ou mal les concepts de hauteur de page, les alignements verticaux et certaines valeurs de tailles, ces éléments sont en revanche parfaitement adaptés au format papier.

Grâce à leur gestion des périphériques de sortie, les feuilles de styles permettent de diffuser un même contenu HTML pour des supports différents. Il est donc possible de tirer parti au maximum des règles et mesures utilisées dans l'imprimerie (points, centimètres, sauts de page, césures, etc.) via un fichier CSS destiné uniquement à cet usage.

Les unités spécifiques

Toutes les unités de mesure recensées dans les spécifications et adaptées au Web sont susceptibles d'être employées en imprimerie. Toutefois, les unités les mieux adaptées à ce support paginé sont les points (`pt`), centimètres (`cm`), millimètres (`mm`) et pourcentages (`%`).

Le point typographique, symbolisé par l'unité `pt`, est adéquat pour définir la taille des contenus de texte. Il s'agit d'une unité très couramment usitée sur la plupart des logiciels de traitement de texte (OpenOffice.org Writer ou Microsoft Word, par exemple).

Pour le calcul des marges de la page et le dimensionnement des blocs, le millimètre (symbole `mm`) et le centimètre (symbole `cm`), ou encore le pourcentage (symbole `%`) accompliront ce rôle à la perfection.

Gérer le support d'impression

Détecter le périphérique

Le support d'impression (`print`) dispose d'un atout incomparable par rapport à d'autres : il est généralement très simple à détecter et à prendre en charge par l'ensemble des navigateurs, même anciens. Le revers de la médaille est qu'un bon nombre de propriétés CSS inhérentes à ce support ne sont reconnues par aucun de ces mêmes navigateurs !

L'attribut media "print"

Introduit dès la norme CSS 2, le `print` est destiné à un support paginé opaque ainsi qu'aux documents visionnés en mode Aperçu avant impression.

Deux méthodes ciblent spécifiquement ce support : un appel à un fichier CSS, ou des styles rédigés directement au sein d'un fichier CSS existant.

La première option nécessite une balise `<link>` placée dans `<head>` et qui va inclure un fichier de styles consacré à l'imprimante par le biais de la syntaxe `media="print"` :

Partie HTML

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

La seconde alternative s'appuie sur la règle `@media` incluse au sein de la feuille de styles globale, associée au mot-clé `"print"` et qui va servir de bloc de règles pour toutes les spécificités d'impression.

La feuille de styles « hôtesse » devra bien entendu couvrir explicitement ou non le support d'impression : une feuille déclarée sans `media` ou en `media="all"` sera compatible, tandis qu'un fichier CSS s'adressant à un `media="screen"` ne pourra pas contenir de styles dédiés à l'impression.

Partie CSS

```
ici les règles CSS dans un fichier en media="all"
@media print {
  /* ici les règles de styles pour l'impression */
}
```

Notez qu'il est également envisageable d'importer une feuille CSS spécialisée au sein même d'une feuille commune à plusieurs supports.

Partie CSS

```
@import url(print.css) print;
```

Selon vos besoins et le contexte d'application, chacune des techniques offre des avantages et des inconvénients :

- Si vous privilégiez les performances et que chaque requête HTTP a son importance, il est sans doute préférable de faire figurer tous vos styles (dont ceux d'impression) dans le même fichier CSS couvrant différents supports.
- Si la relecture des codes est cruciale, ou dans le cadre d'un travail collaboratif, ou encore si le nombre de styles spécifiques à l'impression est élevé, il peut être judicieux de disposer de fichiers CSS séparés.

Les Media Queries CSS 3

Parmi les nouveautés apportées par la version 3 de CSS, les requêtes de média permettent d'affiner la sélection du support.

Ainsi, il est concevable d'appliquer des styles différents selon la résolution du périphérique de sortie :

```
@media print and (min-resolution: 600dpi) {
  /* ici les styles pour imprimante de plus de 600dpi */
}
```

De la même manière, nous pouvons nous adresser différemment à une imprimante couleur ou monochrome :

```
@media print and (monochrome) {
  /* ici les styles pour imprimante monochrome */
}
@media print and (color) {
  /* ici les styles pour imprimante couleur */
}
```

Sans surprise, seules les versions récentes des navigateurs sont capables d'interpréter ces spécifications.

Appliquer les styles CSS

Dans la pratique, même si vous pouvez appliquer les styles pour l'impression directement dans la feuille de styles globale, il demeure plus intuitif de disposer d'un fichier CSS séparé qui sera appelé via l'élément `<link>` comme nous l'avons vu précédemment :

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

La gestion de plusieurs feuilles de styles se passe généralement sans encombres à partir du moment où l'on tient compte du principe d'écrasement chronologique des propriétés CSS : la dernière propriété déclarée écrase la précédente s'il y a conflit.

Ce principe se reproduit également lorsque plusieurs feuilles de styles sont appelées :

```
<link rel="stylesheet" type"text/css" href="styles.css" />
<link rel="stylesheet" type"text/css" href="print.css" media="print" />
```

Dans notre exemple, la première feuille de styles s'appliquera pour tous les supports, y compris pour l'impression. Le second fichier, réservé à l'impression, se contentera d'apporter des « correctifs » au cas par cas (couleurs, marges, tailles de polices).

Il est séduisant de procéder ainsi, car cela permet par exemple de conserver un même style graphique à l'écran et sur le papier. Dans le cas de mises en page très complexes composées de nombreux styles adaptés pour l'écran (des largeurs en pixels, des textes en blanc sur une couleur de fond sombre, des tailles de texte en pixels ou en `em`…), il peut devenir fastidieux d'annuler ou modifier une par une chaque déclaration, sans compter le risque d'omettre certains correctifs.

Dans ce genre de situations, on conseille donc plutôt de réaliser deux feuilles de styles entièrement différentes, une pour l'écran et une pour l'imprimante :

```
<link rel="stylesheet" type"text/css" href="styles.css" media="screen, projection" />
<link rel="stylesheet" type"text/css" href="print.css" media="print" />
```

Limites des navigateurs

Théoriquement, les feuilles de styles pour l'impression permettent de gérer une mise en page relativement complète, avec des options proches de celles d'un logiciel de traitement de texte. Dans la pratique, il faudra se limiter à des choses beaucoup plus basiques, en raison de la prise en charge très partielle par les navigateurs des propriétés CSS relatives à l'impression.

Certaines parties des spécifications CSS n'étant pas ou peu reconnues par la majorité des navigateurs, on ne pourra pas utiliser de façon fiable :

- la gestion du format de papier ;
- la gestion des marges d'impression ;

- les sauts de page avant/après un élément ;
- les paragraphes solidaires ;
- la gestion des lignes veuves et orphelines ;
- et autres propriétés avancées de mise en forme sur papier.

Il sera donc nécessaire de faire simple. Pensez aux feuilles de styles pour l'impression comme à une facilité offerte à l'utilisateur, mais surtout pas comme à un moyen d'obtenir une version imprimée de qualité d'un document. Pour les impressions au rendu fidèle, le format de choix est le PDF.

Hormis quelques rares exceptions, les propriétés dédiées à l'impression sont très peu reconnues par les navigateurs. Curieusement, dans ce domaine, les cancres ne sont pas ceux qu'on attend : seul Opera offre une excellente prise en charge de ce support et il est talonné par… Internet Explorer 8. Firefox ne rattrape son retard qu'à partir de la version 4.

@page

Le sélecteur de page CSS 2.1 `@page` fournit les indications de dimensions, de marges et d'orientation du document imprimé.

L'exemple suivant définit une surface de 21 × 29,7 cm ainsi que des marges de 2 cm :

```
@page {
  size: 21cm 29.7cm;
  margin: 2cm;
}
```

Il est également possible de spécifier l'orientation du document :

```
@page { size: portrait; }  /* portrait */
@page { size: landscape;   }  /* paysage */
```

COMPATIBILITÉ @page

La reconnaissance de `@page` est généralement partielle : la propriété `size` et les propriétés de marges ne sont actuellement comprises que par Opera… et Internet Explorer 8.

orphans

La propriété CSS 2.1 `orphans` (ligne orpheline) gère les coupures à l'intérieur des éléments en spécifiant le nombre minimal de lignes d'un texte devant rester en bas d'une page.

L'exemple suivant s'assure qu'il restera au moins trois lignes de contenu en bas des pages en cas de coupure au sein des paragraphes et des citations, ceci afin d'éviter les lignes orphelines :

```
p, blockquote   {
  orphans: 3;
}
```

COMPATIBILITÉ orphans

`Orphans` est comprise par Opera, Internet Explorer 8 et Firefox 4.

widows

Dans la même optique que `orphans`, la propriété CSS 2.1 `widows` (ligne veuve) désigne une ligne isolée apparaissant en haut d'une page. La valeur attribuée à `widows` spécifie le nombre minimal de lignes laissées en haut de page, par exemple :

```
p {widows: 3;}
```

COMPATIBILITÉ widows

`widows` est comprise par Opera, Internet Explorer 8 et Firefox 4.

page-break

Trois propriétés CSS 2.1 précisent aux agents utilisateurs les endroits où les sauts de page doivent intervenir :

- `page-break-after` : après l'élément ;
- `page-break-before` : avant l'élément ;
- `page-break-inside` : au sein de l'élément.

Les valeurs autorisées pour ces propriétés sont :

- `auto` (par défaut) ;
- `always` (le saut de page est toujours forcé) ;
- `avoid` (le saut de page est interdit) ;
- `left` et `right` (forcent le saut de page et indiquent que la page suivante est à gauche ou à droite).

Le code suivant interdit tout saut de page au sein d'un bloc de citation :

```
blockquote {page-break-inside: avoid;}
```

La règle suivante affiche les titres de niveau 2 en haut de chaque nouvelle page :

```
h2 {page-break-before: always;}
```

COMPATIBILITÉ page-break

Les propriétés `page-break-after` et `page-break-before` sont comprises intégralement par Internet Explorer 8 et Opera. Chez les autres navigateurs, seules les valeurs `always` ou `auto` sont prises en compte.

`page-break-inside`, quant à elle, n'est reconnue que par IE8 et Opera 7.

Propriétés CSS 3

La troisième mouture de CSS apporte son lot de nouveautés et d'extensions au module de support paginé. En voici une liste non exhaustive :

- `fit` (adaptation de la surface de l'élément) ;
- `image-orientation` (applique un angle de rotation à l'image) ;
- `size` (nouvelles valeurs telles que `"A4"` ou `"Letter"`).

Aucun navigateur ne prenant en compte ces propriétés, je ne vais volontairement pas m'attarder dessus.

Méthodologie générale

Que faut-il imprimer ?

La première question qui se pose est celle de l'opportunité d'imprimer – ou non – certains éléments de la page. De nombreuses parties pertinentes à l'écran ne seront pas utiles sur le papier. Il s'agit alors de les identifier et de les masquer lors de l'impression.

En pratique, cela se traduit par la déclaration CSS `display: none;` appliquée aux éléments qui ne doivent pas apparaître sur la feuille de papier.

Ainsi, si je souhaite masquer un menu de navigation et un bloc de publicité, j'écris la règle suivante au sein de la feuille de styles `print` :

```
#nav, #advertisement {display: none;}
```

De cette manière, tous les éléments identifiables dans le code HTML peuvent être ciblés et écartés de l'affichage papier.

Dans le cas où vous souhaiteriez masquer des éléments très spécifiques (un lien parmi d'autres, ou un paragraphe indépendamment de ses voisins), il vous suffit d'employer la classe `.noprint` sur ces éléments, puis d'ajouter ce sélecteur de classe au sein du bloc précédent :

```
#nav, #advertisement, .noprint {display: none;}
```

À l'inverse, si vous désirez faire apparaître des éléments spécifiques à l'impression, vous pouvez les désigner via une classe telle que `.print`. La démarche sera alors de masquer les éléments `.print` dans la feuille de styles classique d'écran, puis de la réhabiliter pour l'impression :

Feuille de styles écran

```
.print {display: none;}
```

Feuille de styles pour l'impression

```
.print {display: block;}
```

Nous avons donc la possibilité de pas imprimer du tout certains éléments. Qu'est-ce qui doit être imprimé, et qu'est-ce qui ne doit pas l'être ? C'est à chacun de décider de ce qu'il souhaite voir figurer sur les pages imprimées. Voici toutefois quelques pistes.

Faut-il imprimer l'en-tête, le logo ou le titre du site ?

Ça n'est pas indispensable, mais on peut vouloir garder, à l'impression, une identification forte du site. En général, si on ne supprime pas complètement cet élément, on affichera quelque chose de sobre, ne prenant pas une surface papier trop importante.

Faut-il imprimer le menu de navigation ?

À partir du moment où il est impossible pour l'utilisateur de cliquer sur le papier (c'est une évidence, mais autant le rappeler), la présence d'un système de navigation ne se justifie pas vraiment. En général, c'est la première chose que l'on supprimera pour l'impression.

Faut-il imprimer le fil d'Ariane ?

On voudra peut-être imprimer le fil d'Ariane (un système de navigation du type : « Vous êtes ici : Accueil>Rubrique>Page en cours »), dans la mesure où il permet de situer la page imprimée dans l'arborescence du site. On pourra aussi considérer que cette information n'est pas indispensable, voire perturbante sur papier.

Faut-il imprimer les formulaires, et plus généralement tous les éléments d'interaction avec le site ?

Là encore, c'est peu probable. Pour la feuille de styles d'impression d'un blog ou d'un site offrant la possibilité de laisser des commentaires via un formulaire, on cachera probablement ce formulaire. En revanche, s'il s'agit d'un formulaire sur une page d'inscription à un site, il se pourrait qu'un utilisateur veuille l'imprimer.

Est-ce utile d'imprimer toutes les images présentes sur une page ?

Les images prennent de la place sur la page et consomment beaucoup d'encre. Il ne faut donc imprimer que les images indispensables à la bonne compréhension du contenu textuel, ou apportant elles-mêmes une information importante. Notez que, avec les navigateurs ayant une bonne compréhension des sélecteurs CSS avancés, il est possible de remplacer l'image par son texte alternatif (le contenu de l'attribut `alt`).

Pour finir, soyez attentifs à ne pas avoir la main excessivement lourde. Il ne s'agit pas de réduire un contenu à sa « plus pure expression », en supprimant des informations n'appartenant pas directement au contenu principal de la page. Certaines informations relatives à un article (catégorie, auteur, date de publication, date de modification…) sont utiles quel que soit le support de restitution.

En résumé, organisez bien vos contenus HTML, puis déterminez ce qui, imprimé, sera utile aux utilisateurs de votre site et ce qui ne le sera pas. C'est à vous de faire les arbitrages nécessaires, mais le plus souvent, ils se font facilement et on ne garde à l'impression que le contenu vraiment utile. Au-delà d'un contenu imprimé plus clair, il ne faut pas douter que l'internaute vous sera reconnaissant pour le temps d'impression plus court et les économies d'encre, de papier et d'énergie réalisés.

Bonnes pratiques

Généralités

S'il devait encore être utile de le rappeler, le support d'impression n'est pas tout à fait comparable à l'écran de bureau.

Gardez en tête que le support d'impression ne vous permet pas :

- de cliquer sur des liens ou éléments de navigation ;
- d'effectuer des recherches sur la page ou des calculs ;
- d'agrandir certaines zones de la page ;
- d'envoyer le document à un ami par courrier électronique ou via un réseau social ;
- de faire défiler la page ;
- de répondre à un formulaire, etc.

De ces différences découlent un certain nombre de bonnes pratiques générales à appliquer :

- Passez le fond de page en blanc et les textes de contenu (ainsi que les titres et liens) en couleur noire.
- Optez pour une famille de polices lisible, généralement sans empattements (sans serif).
- Après suppression des éléments inutiles, adaptez la largeur du contenu restant à la page.
- Modifiez les unités des polices ou des dimensions de blocs (`em` et `px` vers `pt` ou `cm`).
- Tenez compte du comportement naturel des navigateurs, qui n'impriment généralement pas les images d'arrière-plan.

Pour vous faciliter la tâche, sachez que le plug-in Webdeveloper Toolbar sur Firefox permet de n'afficher que les CSS d'impression en passant par son menu : CSS>Afficher les CSS par type de médium (print).

Marges

Les marges d'un document imprimé peuvent être définies très simplement à l'aide de la règle `@page` associée à la propriété `margin` ou ses dérivées `margin-top`, `margin-right`, etc. :

```
@page {
  margin: 1.5cm 2cm;
}
```

L'intérêt de cette règle est assez discutable dans la mesure où il est également parfaitement envisageable d'appliquer ces marges à l'élément `<body>`.

Cependant, sur les versions de navigateurs à venir et déjà depuis Internet Explorer 8, il est possible de distinguer les marges de la page d'en-tête ou des pages de gauche et de droite :

```
@page :first {
  margin-left: 2cm;
}
@page :left {
```

```
  margin-left: 1.5cm;
  margin-right: 1cm;
}
@page :right {
  margin-left: 1cm;
  margin-right: 1.5cm;
}
```

Sauts, veuves et orphelines

Encore peu comprises, les propriétés de gestion de sauts et de coupures constituent un bonus non négligeable pour les navigateurs très récents. Elles facilitent la lecture des documents imprimés sans risque d'incompatibilité puisqu'elles ne sont tout simplement pas appliquées par les navigateurs qui ne les comprennent pas.

Forcer le saut de page avant chaque titre principal

```
h1 {
  page-break-before: always;
}
```

Empêcher les sauts après les titres et les légendes de tableaux

```
h1, h2, h3, caption {
  page-break-after: avoid;
}
```

Empêcher les sauts à l'intérieur des blocs de citation et les listes

```
blockquote, ul, ol {
  page-break-inside: avoid;
}
```

Empêcher les lignes veuves et orphelines au sein des paragraphes

```
p {
  widows: 3;
  orphans: 3;
}
```

J'ai pris pour habitude d'intégrer automatiquement toutes ces règles (adaptées bien entendu) dans mes feuilles de styles d'impression par défaut.

Images et arrière-plans

La plupart du temps, vous ne serez pas gênés par l'adaptation des images d'arrière-plan de vos documents, puisque celles-ci ne sont généralement pas affichées par les navigateurs lors de l'impression. La seule exception à la règle aujourd'hui semble être Opera, qui imprime par défaut les images de fond.

Ce comportement des navigateurs peut éventuellement poser des problèmes, notamment dans le cas où vous auriez défini une image de contenu importante sous forme d'arrière-plan, que vous

souhaiteriez voir apparaître dans la page imprimée. Il peut s'agir d'un plan d'accès, d'un logo ou d'éléments de liste (puces).

L'une des solutions a été évoquée précédemment : ajouter dans le code HTML un élément image (`<img>`) qui sera masqué à l'écran et visible sur papier en jouant avec les déclarations `display: none` et `display: block`. Une autre possibilité est de ne pas masquer l'élément image, mais de faire varier ses dimensions via CSS : passer par exemple de `width: 1px; height: 1px;` à `width: auto; height: auto;`.

Dans les deux cas, ces alternatives imposent la gestion d'une double occurrence de la même image.

Positionnement

La gestion des différents schémas de positionnement pour l'impression est une tâche particulièrement laborieuse :

- Les positionnements fixés se rapportent à chacune des pages. Ainsi, un élément en `position: fixed` en haut de page apparaîtra en haut de toutes les pages et pas seulement de la première !
- Les positionnement absolus et flottants très longs ou mal situés peuvent être tronqués, passer à la page suivante ou carrément disparaître.
- Les déclarations `overflow: auto` ou `overflow: hidden`, très appréciées pour apporter un contexte de formatage aux blocs flottants, posent également des problèmes de rognage d'éléments.

À la vue de cette liste, vous aurez sans doute vite compris que la technique de positionnement la plus indéfectible demeure celle du *flux courant*.

Dans cette optique, je vous conseille vivement de réintégrer dans le flux tous les conteneurs récalcitrants en modifiant la valeur associée à leur positionnement : `float` sera associé à `none` et `position` sera associé à `static`, comme le montre cet exemple :

```
#header { position: fixed; }
@media print {
  #header { position: static; }
}
```

Liens hypertextes et abréviations

Vous en conviendrez, il est difficile de cliquer ou d'actionner des liens et des boutons de soumission sur une feuille de papier. À la place de s'en plaindre, nous allons en tirer profit au maximum.

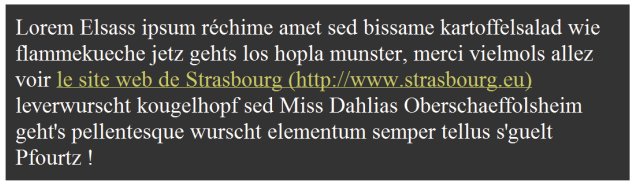
Commençons par forcer le soulignement des liens afin de s'assurer de véhiculer cette information sur papier, même si l'élément n'est pas cliquable :

```
@media print {
  a { text-decoration: underline; }
}
```

Sachez également qu'il est possible via CSS 2.1 de faire apparaître sur la page imprimée les adresses contenues dans les liens, ce qui permet au lecteur de disposer de cette information essentielle (figure 10-2).

Figure 10-2

Les URL des liens affichées



Cette astuce se base sur le pseudo-élément `:after` associé à la propriété `content` qui recevra comme valeur le contenu de l'attribut `href` :

```
@media print {
  a[href]:after {
    content: " (" attr(href) ")";
  }
}
```

Soyez attentifs à ne réserver cette règle que pour les parties de contenu (pas dans l'en-tête, le pied de page ou la navigation sous peine de dégradation sérieuse de votre mise en page).

ASTUCE Raccourcisseurs d'URL

Une astuce intéressante pour éviter les soucis d'affichage avec les URL trop longues est de les… transformer en URL courtes à l'aide des nombreux services en ligne existants : tinyurl.com, bit.ly, goo.gl, etc.

Les abréviations et autres éléments disposant d'infobulles `title` méritent eux aussi un traitement de faveur avant d'être imprimés.

Nous pouvons faire apparaître les abréviations dans leur version étendue de la sorte :

```
abbr[title]:after {
  content: " (" attr(title) ")";
}
```

Ou encore, si vous aimez le risque, vous ferez apparaître tous les `title` à la suite des éléments qui en comportent :

```
[title]:after {
  content: " (" attr(title) ")";
}
```

Tester avant l'impression

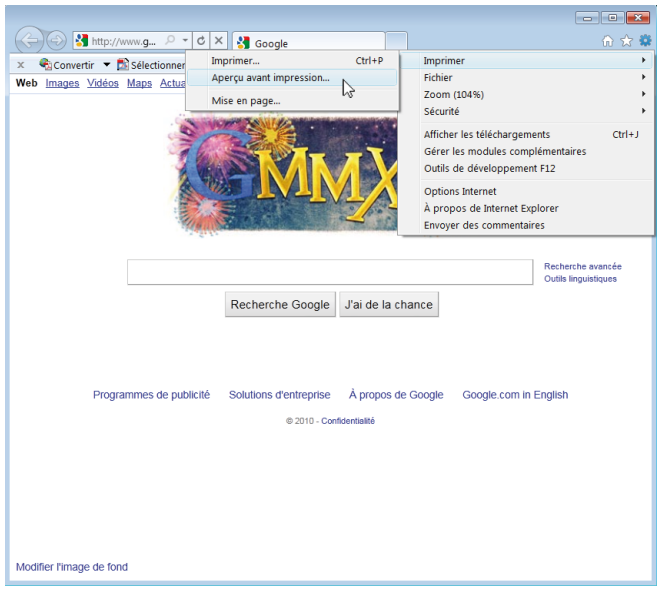
Je vous livre un dernier conseil sur ce chapitre consacré au support d'impression. Il s'agit d'une recommandation qui peut paraître ridicule à certains, mais je suis persuadé qu'elle trouvera son public.

Ce conseil se résume tout simplement à « éviter d'imprimer une nouvelle feuille à chaque nouvelle règle CSS mise en place ».

Choisir l'option Aperçu avant impression disponible sur la quasi-totalité des navigateurs vous fera économiser votre encre et votre papier. En règle générale, et sauf modification des paramètres d'impression du navigateur, le rendu de l'aperçu sera parfaitement identique au document final sur papier (figure 10-3).

Figure 10-3

Aperçu avant impression sur Internet Explorer



Styles de base pour l'impression

En guise de résumé de ce chapitre consacré aux techniques d'impression, je vous propose une feuille de styles dédiée, qui condense toutes les bonnes pratiques et astuces énoncées précédemment.

Ces règles peuvent être externalisées dans un fichier CSS séparé, ou incluses directement au sein du document de styles global, déclarées à l'aide de la règle `@media print {…}`.

L'objectif de cette feuille de styles est avant tout de poser un socle de base commun (marges, couleurs, contrastes, tailles, gestion des sauts de pages, des veuves et des orphelines), que vous pourrez adapter à vos convenances ou besoins personnels.

Pour rappel, certaines propriétés sous-citées ne sont reconnues que sur Opera ou à partir d'Internet Explorer 8 et Firefox 4, mais ne sont néanmoins pas bloquantes pour les retardataires :

```
body {
  width: auto!important;
```

```
  margin: auto!important;
  font-family: serif;
  font-size: 12pt;
  background-color: #fff!important;
  color: #000!important;
}
p, h1, h2, h3, h4, h5, h6, blockquote, ul, ol {
  color: #000!important;
  margin: auto!important;
}
.print {
  display: block; /* affichage des éléments de classe print */
}
p, blockquote {
  orphans: 3; /* pas de ligne seule en bas */
  widows: 3; /* pas de ligne seule en haut */
}
blockquote, ul, ol {
  page-break-inside: avoid; /* pas de coupure dans ces éléments */
}
h1 {
  page-break-before: always; /* chaque titre commence sur une nouvelle page */
}
h1, h2, h3, caption {
  page-break-after: avoid; /* pas de saut après ces éléments */
}
a {
  color: #000!important;
  text-decoration: underline!important;
}
a[href]:after {
  content: " (" attr(href) ")"; /* affichage des URL des liens */
}
```

HTML 5, Internet Explorer et l'impression

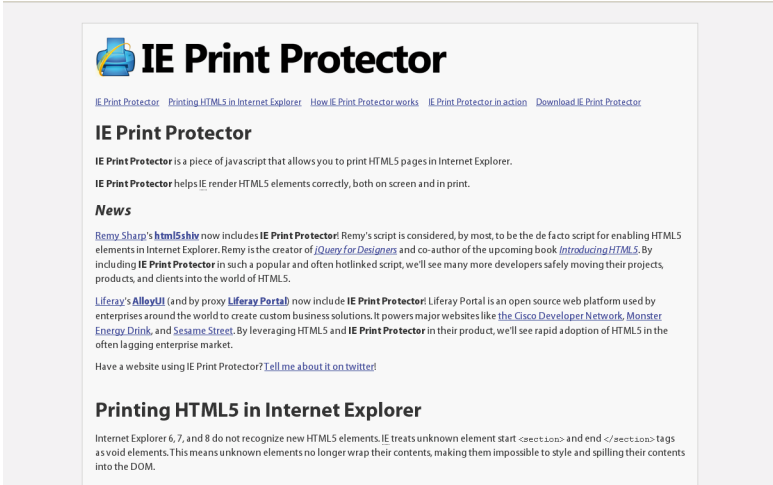
Sous Internet Explorer (version 8 comprise), les balises HTML 5 non encore implémentées comme `<header>`, même si elles sont créées dans le DOM et sont correctement affichées à l'écran, ne prendront tout simplement pas en compte les propriétés destinées à l'impression.

Il est donc inutile de chercher à les styler directement. Il faut passer par JavaScript pour que la CSS d'impression soit prise en compte pour ces nouvelles balises : IE Print Protector est un script permettant l'affichage de ces éléments HTML 5 sur papier (figure 10-4).

▸ http://www.iecss.com/print-protector

Figure 10-4

IE Print Protector



Aller plus loin…

Vous êtes prêt maintenant à concevoir une feuille de styles CSS dédiée au support d'impression. Si vous souhaitez aller plus loin dans ce domaine, voici une liste de liens vers des tutoriels aux techniques plus avancées...

ALLER PLUS LOIN **CSS pour l'impression**

PrintFriendly :

- http://www.printfriendly.com

« Faites bonne impression avec les CSS » :

- http://www.pompage.net/pompe/impression/

« L'impression à votre façon » :

- http://www.pompage.net/pompe/votreimpression/

« Feuille de styles pour une mise en page à l'impression » :

- http://actuel.fr.selfhtml.org/articles/css/mise_en_page_imp/

« Créer une feuille d'impression CSS pour votre site » :

- http://www.skyminds.net/2006/10/10/creer-une-feuille-dimpression-css-pour-votre-site/

11

CSS et les messageries

Parler des messageries et de courrier électronique de nos jours, c'est prendre conscience du retard considérable accumulé par les clients de courrier électronique et les webmails. Suivez-moi dans ce retour à la préhistoire où l'on parlera de tableaux de mise en page, de styles CSS en ligne, de restriction des images, de Flash et autre JavaScript, et de toute autre bonne pratique à suivre pour mieux acheminer vos courriers électroniques.

Standards ? Connais pas !

Les problèmes de compatibilité de CSS 3, des navigateurs mobiles et du support d'impression semblent bien dérisoires comparés à la prise en charge des styles CSS chez les différents clients actuels de messagerie.

Il est pourtant difficile aujourd'hui de comprendre comment certains clients de messagerie ne parviennent pas encore à reconnaître les rudiments des styles CSS 1. Pire, il ne s'agit pas forcément d'un problème générationnel puisque Google Gmail, par exemple, fait partie des moins bien lotis dans ce domaine ! Chaque support traîne son boulet : Internet Explorer 8 pour l'écran, Blackberry pour le Web mobile, Firefox 3.6 pour l'impression... Lotus Notes 7, suivi de près par Gmail pour la messagerie.

Ce chapitre concernant les CSS et les messageries va donc vous sembler terriblement frustrant par certains de ses aspects, notamment lorsque je vais devoir vous annoncer que la technique la plus robuste consistera à retourner à la préhistoire de la conception web et de concevoir ses gabarits à l'aide de tableaux de mise en page.

Avant de parvenir à cette terrible conclusion, nous allons tenter de mieux comprendre ce marché de la messagerie électronique, les outils employés par les clients, les statistiques d'usages et les forces et faiblesses en présence.

Client de courrier électronique ou webmail ?

Selon vos habitudes ou vos besoins, deux solutions s'offrent à vous pour la consultation de vos courriers électroniques : un client de courrier électronique ou un webmail.

Le premier consiste en un logiciel installé sur votre poste de travail, qui télécharge les messages depuis Internet et stocke leur contenu sur votre ordinateur. Les clients les plus connus sont Outlook de Microsoft, Thunderbird de Mozilla, Apple Mail, Lotus Notes, Eudora, Foxmail et Incredimail.

Un *webmail* est un service en ligne, une page web sur laquelle vous vous connectez afin de récupérer votre courrier électronique. Les contenus ne sont pas stockés sur votre disque dur, ils demeurent sur le serveur en ligne. Parmi les webmails célèbres, citons Yahoo! Mail, Hotmail et Google Mail (Gmail).

Afin de bien s'imprégner de ce monde ténébreux qu'est celui d'un concepteur en messageries web, voici une courte présentation des clients de courrier électronique et webmails usuels.

Les logiciels de messagerie courants

Microsoft Outlook

Parmi les clients de messagerie électronique les plus employés dans le monde, celui de Microsoft se positionne souvent en tête de file.

De nos jours, les versions principalement usitées sont Outlook 2003 et Outlook 2007, sachant que le moteur de rendu de ce dernier s'est considérablement dégradé, au point d'être devenu un écueil sérieux pour la conception de courriels en CSS. C'est à croire que Microsoft a volontairement fait un retour dans les années 1990 pour ce qui est de son intégration de CSS au sein de sa messagerie.

Pour commencer, Outlook 2007 (figure 11-1) ne reconnaît pas les éléments HTML suivants : `<frameset>`, `<head>`, `<html>`, `<meta>`, `<xml>` et… `<style>` ! Votre première mission sera donc d'appliquer tous vos styles au sein même de chacune des balises à l'aide de l'attribut HTML `style`.

Les éléments reconnus peuvent pour la plupart bénéficier des quelques styles CSS suivants : `display`, `width`, `height`, `margin`, `padding`, `font`, `color`, `border`, `text-indent`, `list-style`, `line-height`, `letter-spacing`, `text-align`, `border-collapse` et `background-color`.

Certains éléments, tels que `<span>`, ne reconnaissent pas les propriétés `margin`, `padding` et `text-indent` ; d'autres, tels que `<div>` et `<p>`, ne reconnaissent pas `width`, `height`, `padding` et `border`.

Enfin, les différentes propriétés de positionnement hors flux telles que `float` et `position` ne sont aucunement prises en compte.

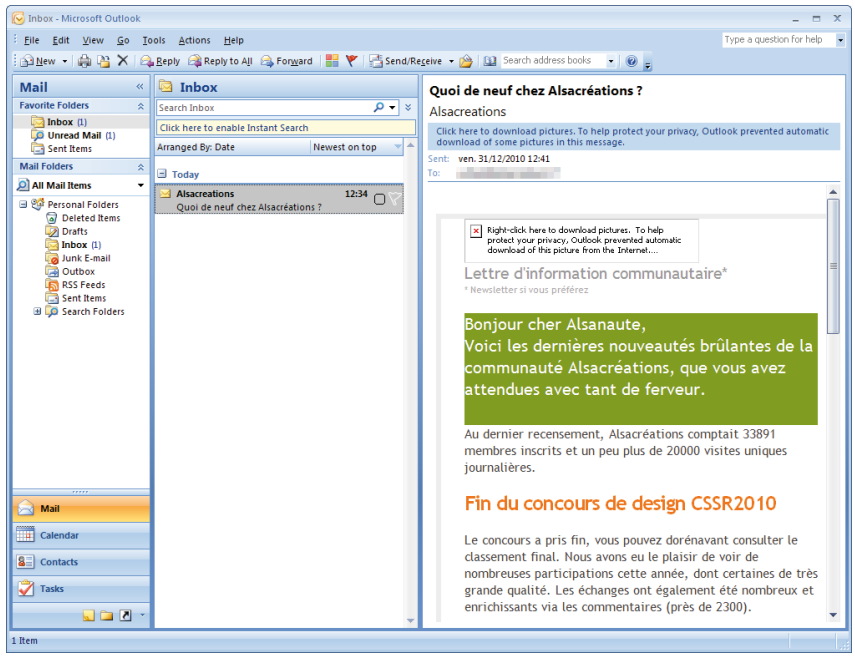


Figure 11-1

Aperçu de Microsoft Outlook 2007

Apple Mail et iPhone Mail

Mail est le logiciel de messagerie d'Apple fourni en standard sur les systèmes d'exploitation Mac OS X (ordinateurs) et iOS (mobiles). La version 4 de Mail est sortie avec Mac OS X 10.61 (Snow Leopard) en 2009.

Qu'il soit porté sur un ordinateur de bureau ou sur un terminal mobile tel que iPhone, iPod ou iPad, Mail est parmi les plus aboutis et les plus respectueux des standards CSS. Sa panoplie compte même un bon nombre de propriétés et valeurs CSS 3 : `border-radius` (avec préfixe `-webkit-`), `opacity`, `RGBa`, `text-shadow` ou encore `word-wrap` sont de la partie.

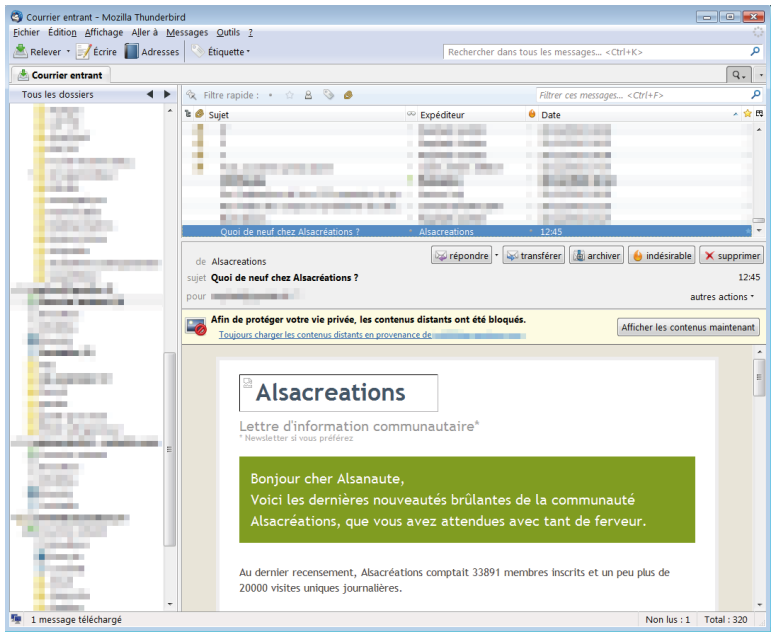
Thunderbird

Thunderbird est un client de messagerie libre et multilingue distribué gratuitement par la Fondation Mozilla et issu du Projet Mozilla (figure 11-2). Tout comme Firefox, Thunderbird est basé sur le moteur Gecko et dispose d'une interface en XUL, ce qui lui permet de fonctionner sur diverses plates-formes.

Actuellement à la version 3 de son logiciel, Thunderbird affiche un tableau de reconnaissance des propriétés CSS assez impressionnant, au même niveau que le client d'Apple, à l'exception des propriétés CSS 3.

Figure 11-2

Aperçu de Thunderbird



Lotus Notes

La messagerie de Lotus, assez peu utilisée en France, compte parmi les retardataires les plus marquants dans le domaine de l'envoi de courriers au format CSS. À titre d'exemple, les versions 6 et 7 ne reconnaissent tout simplement aucune propriété de style, à l'exception de quelques miettes dédiées à la typographie.

Fort heureusement, la version Lotus Notes 8.5 a marqué un virage radical en faveur des standards et se positionne dans le peloton de tête à présent : même certaines propriétés ou valeurs CSS 3 sont comprises.

Les webmails habituels

Yahoo! Mail

L'outil de messagerie de Yahoo!, créé initialement en 1997, propose une bonne prise en charge de CSS en général ; ce n'est pas le client qui vous posera le plus de soucis ! Il est même capable, contrairement à d'autres, d'afficher des images de fond via `background-image`.

Dans le lot de ses quelques faiblesses, comptons tout de même la non-reconnaissance des propriétés CSS 3, ni de `:first-letter` et `position`.

Windows Live Hotmail

Hotmail est un service de messagerie en ligne gratuit proposé par Microsoft que l'on peut considérer comme étant le premier webmail, puisque créé avant 1997.

Son traitement global de CSS se situe dans la moyenne de ses confrères, avec les restrictions auxquelles nous sommes habitués : pas de positionnement (excepté `float`), pas d'images d'arrière-plan ni de propriété `list-style-image`, pas de sélecteurs avancés (enfant, adjacent, attribut, `:first-letter`).

Une déficience pour le moins insolite – et gênante – caractérise toutefois ce webmail : son absence complète de prise en charge de la propriété `margin` !

Google Mail (Gmail)

Gmail est le service de messagerie gratuit lancé par Google en 2004 et ouvert au public en 2006 (figure 11-3). Malgré sa prime jeunesse dans le monde des services de messageries, Gmail accuse un retard impressionnant en ce qui concerne la reconnaissance des CSS.

Par exemple, qu'il soit installé sur un ordinateur ou sur un mobile Android, le client Gmail ne reconnaît tout simplement ni la balise `<style>`, ni la balise `<link>`, ni aucun sélecteur CSS ! En clair, seuls les styles en ligne, directement incorporés aux balises HTML seront pris en compte dans les courriels affichés sur Gmail.

En outre, seules sont acceptées quelques propriétés de typographie (`font`, `text-align`, `letter-spacing`, `text-indent`, `text-decoration`, `vertical-align`), de couleurs (`color`, `background-color`), de boîte (`border`, `width`, `height`, `margin`, `padding`), de positionnement (`display`, `float`, `clear`), de listes (`list-style-type` et `list-style-position`) et de tableaux (`border-collapse`, `border-spacing`, `table-layout`, `empty-cells`, `caption-side`).

Figure 11-3

Aperçu de Gmail

Lacunes des clients de messagerie

Suite à ce parcours de présentation des services de messagerie classiques, voici un résumé non exhaustif des principales lacunes particulières de ces différents outils :

- `<style>` n'est pris en compte ni par Gmail ni par Lotus Notes 7.
- Les sélecteurs CSS ne sont reconnus ni par Gmail ni par Lotus Notes 7.
- `margin` n'est compris ni par Hotmail ni par Lotus Notes 7.
- `width` et `height` ne sont reconnues ni par Outlook 2007/2010, ni par Lotus Notes 7, hormis sur des cellules de tableau.
- `display` n'est pas pris en charge par Outlook 2007.
- `float` n'est reconnu ni par Outlook 2007 ni par Lotus Notes 7.
- Sauf rares exceptions (clients Apple), aucune propriété ou valeur CSS 3 n'est prise en compte par les clients de messagerie et webmails actuels.

Le marché des clients de messagerie

Un peu de statistiques

Deux sites web, CampainMonitor et FingerPrint, dressent les palmarès des clients de messagerie d'une façon que l'on peut considérer comme fiable, puisque effectuée sur une durée de six mois et sur des échantillons de plusieurs centaines de millions de courriels recensés.

Leurs résultats concernent les chiffres mondiaux et sont consultables aux adresses ci-après :

- http://www.campaignmonitor.com/stats/email-clients/
- http://fingerprintapp.com/email-client-stats

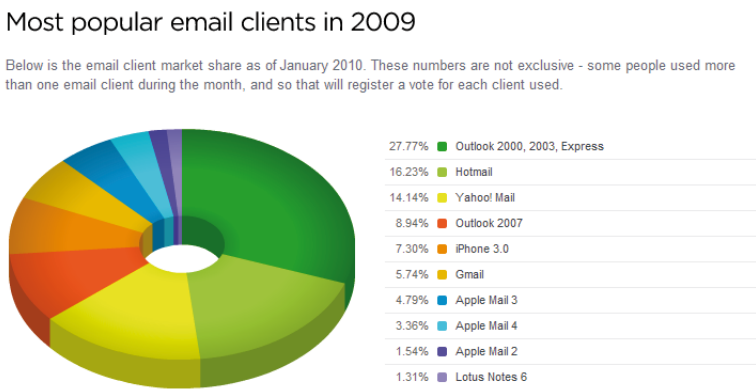
Ils décrivent un marché pour le moins morcelé (figure 11-4) :

- Outlook (toutes versions confondues) domine le marché avec des chiffres situés entre 35 et 45 % d'usagers.
- Hotmail occupe de 15 à 20 % du marché.
- Yahoo! Mail fédère entre 13 et 15 % des utilisateurs.
- iPhone (ainsi que iPod et iPad) sont utilisés par un panel de 5 à 7 % des utilisateurs.
- Gmail représente une part de marché de 5 à 6 %.
- Apple Mail occupe environ 4 % du marché.
- Thunderbird est utilisé par environ 2 % de la population.
- Lotus Notes représente environ 1 % des usages.
- AOL, Windows Live Desktop, Entourage et MobileMe se partagent les restes.
- Une part non négligeable d'environ 10 % des outils de messagerie n'est pas détectable ou reconnaissable.

En dépit de leur faible part de marché, certains de ces outils affichent une progression constante et rapide ces derniers temps. C'est le cas notamment pour l'iPhone qui a gagné quasi 7 % en une année, mais aussi pour Apple Mail (+ 3 %) et Outlook 2007 (+ 2 %). D'autres, tels Outlook 2000/2003 (- 5 %) et Yahoo! Mail (- 2 %), accusent un déclin au cours de l'année 2010.

Figure 11-4

Statistiques de CampaignMonitor



Quelle est votre cible ?

La qualité des services de courriel et leur acquisition des standards CSS est pour le moins disparate, vous en conviendrez avec moi. Les statistiques évoquées précédemment n'offrent qu'une pâle vision de la réalité : l'usage d'Outlook en France n'est pas le même que dans le reste du monde, le public dit *geek* a un penchant pour les services tels que Gmail ou Thunderbird, contrairement aux grosses entreprises, historiquement attachées à Lotus Notes et à Outlook.

En conséquence, la cible que vous souhaitez atteindre par vos envois de courriers électroniques n'est pas forcément identifiée, voire identifiable. Il va falloir faire des choix et contenter une globalité floue.

Dans tous les cas et quelles que soient vos sélections, une certitude devient à présent évidente : il va vous falloir renoncer au monde merveilleux des CSS 3, voire tout simplement oublier les CSS, pour espérer délivrer un contenu lisible et utilisable par vos lecteurs. Soyez courageux, ça se passera bien.

Les bonnes pratiques du publipostage (e-mailing)

Quelle largeur ?

Les clients de messagerie et webmails sont pour la plupart constitués d'un ou plusieurs panneaux d'outils latéraux, en conséquence de quoi la place disponible pour l'affichage du contenu des courriers est souvent bien plus restreinte que l'espace offert par l'écran. Cette règle est d'autant plus vérifiable sur les nouveaux terminaux nomades tels que les téléphones mobiles, les netbooks et l'iPad d'Apple.

Au final, la largeur réelle allouée à la consultation de ses courriels ne dépasse que rarement 600 pixels (vous pouvez d'ailleurs vous en rendre compte en observant les derniers messages publicitaires reçus dans votre boîte de réception).

D'une manière générale, il y a deux façons de procéder :

- proposer un gabarit de largeur fluide, qui s'adaptera à l'espace du client ;
- imposer une mise en page d'une largeur fixe de 600 pixels au maximum, ou fluide (en pourcentage), telles que le préconisent les recommandations en général.

Dans le premier cas, il faudra s'attendre à de grandes disparités d'affichage entre un terminal mobile et un écran 26 pouces. C'est pourquoi je vous suggère de ne pas prendre de risque, de suivre l'usage courant et d'opter pour une largeur fixée à 600 px.

Images

Un grand nombre de services de messagerie bloquent par défaut, ou par choix de l'utilisateur, l'affichage des images HTML au sein des courriers électroniques. Cette restriction est pour le moins contraignante puisque les campagnes de publipostage sont le plus souvent truffées d'illustrations et de visuels marketing.

OUTIL Qui bloque les images ?

CampaignMonitor fait la liste des messageries qui empêchent l'affichage des images par défaut (si vous ne bénéficiez pas du statut d'expéditeur de confiance) à cette adresse :

▸ http://www.campaignmonitor.com/image-blocking/

Par exemple, Outlook, Yahoo!, Windows Live, Thunderbird et Gmail bloquent systématiquement toute image distante par défaut, à la différence de Hotmail, Apple Mail et iPhone. Pire encore, les anciens Blackberry sont incapables d'afficher la moindre image dans un courriel !

Par ailleurs, ce n'est pas au sein de la spécification CSS que nous trouverons notre salut, puisque quasi aucun client ne reconnaît la propriété `background-image`. Seule `background-color` est reconnue la plupart du temps.

La conclusion de ce constat est évidente : n'utilisez en aucun cas des images (`<img>` HTML ou arrière-plans CSS) pour véhiculer des contenus et informations pertinents, notamment des éléments d'action, des liens ou des titres.

Quelques réflexions parallèles viennent toutefois adoucir ce cruel verdict :

- Les images jointes directement au courrier (contrairement aux images distantes) sont généralement acceptées par les différents clients.
- De nombreuses messageries accordent aux relations issues de votre carnet de contacts le statut privilégié d'expéditeur de confiance (*trust sender*) et offrent plus de liberté aux courriers échangés avec ces relations, en garantissant parfois l'affichage par défaut des images. Une bonne habitude est de demander à vos destinataires d'ajouter votre adresse à leur carnet à chaque fois que vous le pouvez.

- La plupart des services de messagerie reconnaissent et interprètent les textes alternatifs (attribut `alt`) apposés aux images HTML. Pensez systématiquement à les renseigner de façon cohérente : leur contenu doit être parfaitement représentatif de l'image associée et le plus concis possible pour ne pas déformer votre mise en page.
- Puisque rien ne vous garantit le bon acheminement de vos images, il est de bon ton de fournir dès le début de votre courriel un lien en texte brut vers la version HTML en ligne du résultat. Ainsi, les destinataires intéressés bénéficieront au moins d'une version maîtrisée sur votre site web.
- N'oubliez pas non plus que les images non chargées n'occuperont pas la surface prévue au sein de votre mise en page. Afin de vous prémunir d'éventuels désagréments de rendu et de déformations, pensez à indiquer les largeurs et hauteurs de chacun des éléments `<img>` de votre document.
- Enfin, dans le cas où plusieurs images sont positionnées côte à côte, certains espaces indésirables peuvent apparaître entre elles. L'astuce consiste alors à leur affecter des déclarations `display: block` ou `float: left` pour éviter ces effets de bord.

Toutes ces précautions mises en application ne vont malheureusement pas vous garantir un affichage parfait des images, mais vous préserver de la plupart des déconvenues causées par leur blocage sur les messageries.

Flash et JavaScript

Dès lors que les environnements de messagerie ont la fâcheuse tendance à bloquer les images statiques, vous ne serez guère surpris d'apprendre que la situation empire lorsqu'il s'agit d'éléments multimédias tels que Flash et JavaScript.

Là encore, la règle est implacable : n'utilisez aucun support multimédia quel qu'il soit au sein de vos courriers électroniques. Flash est quasi inconnu des clients de messagerie (le seul à ne pas empêcher son interprétation est Apple Mail) et, pour des raisons de sécurité, tout code JavaScript sera supprimé ou remplacé par un avertissement sur les postes de vos destinataires.

Désinscription

La CNIL (Commission nationale de l'informatique et des libertés), dans sa quête de protection des internautes, milite activement contre la prospection commerciale abusive (plus connue sous le nom de *spam*). En effet, selon la Loi informatique et libertés, toute personne peut refuser, sans avoir à se justifier, que les données qui la concernent soient utilisées à des fins de prospection commerciale.

C'est pourquoi il est obligatoire de proposer à vos destinataires un moyen de radier leurs informations personnelles de vos bases de données de courriers, généralement via un lien de désinscription ou un moyen de vous contacter directement.

Faites bien attention à ne pas vous soustraire à ce droit, car cela peut vous coûter cher : ainsi les sociétés Cdiscount et Isotherm ont-elles été sanctionnées en 2009 par des amendes respectives de 60 000 et 30 000 euros pour n'avoir pas pris en compte les multiples demandes de désinscription de leurs destinataires.

Encodage des caractères

Vous avez sans doute déjà reçu des courriers électroniques incompréhensibles, envahis de caractères bizarres et non lisibles.

Les environnements actuels (logiciels éditeurs HTML, serveurs, bases de données et de courriels, messagerie) sont généralement capables d'employer un jeu de caractères considéré comme universel, l'UTF-8, une variante Unicode permettant d'encoder la plupart des langues et symboles.

Dans de rares cas, l'un des maillons de la chaîne ne reconnaît pas ce type d'encodage ou est victime d'un conflit d'encodage. Il s'agit généralement d'un courriel mal encodé ou sous un format spécifique : le format ISO, dénué d'accents et encore fréquemment employé dans les pays anglophones.

Pour éviter toute mésaventure, je vous conseille vivement de vérifier l'encodage du fichier et de vos documents, mais aussi d'éviter les blancs et caractères accentués pour les noms de fichiers et URL.

Testez !

Cette liste de bonnes pratiques ne pourrait se conclure sans finir par la plus importante de toutes : testez vos pages sur un maximum de messageries.

De très nombreux clients de courrier électronique et webmails sont gratuits : il vous suffit de vous y inscrire pour bénéficier de leurs services et pouvoir réceptionner vos courriers. Il suffit d'un minimum d'organisation et de retenir tous vos mots de passe pour vérifier vos envois sur les plates-formes les plus diverses :

- parmi les webmails : Yahoo! Mail, Hotmail et Gmail ;
- sur PC : Outlook, Thunderbird et Lotus Notes ;
- sur Mac : Mac Mail, Entourage et Eudora ;
- … et si vous disposez d'un smartphone, n'oubliez pas d'y tester vos documents également.

Méthodologie générale

Abstraction faite des préférences de votre client, il vous est impossible de savoir quel environnement de gestion des courriels vos destinataires utiliseront. En fin de compte, il vous faudra constamment vous fonder sur des suppositions et faire « au mieux » car rien ne pourra vous permettre de cibler les différents services de messagerie.

Étape 1 : retour aux tableaux de mise en page

Rejoignons la dure réalité de la conception de documents courriels : hors tableaux de mise en forme, point de salut et d'interopérabilité.

Encore aujourd'hui, le panel des propriétés des feuilles de styles comprises par les clients de messagerie est extrêmement ténu et c'est encore pire lorsqu'il s'agit d'interpréter les schémas de

positionnement en CSS : presque tous ignorent la propriété `position` et peu nombreux sont ceux qui reconnaissent `float` et `display`.

Former la structure

Faisant fi de toutes les règles d'accessibilité, de performance et de sémantique, la conception du gabarit global d'un document de courrier électronique nécessite l'emploi de tableaux de mise en page et de cellules. Le tout sera saupoudré d'attributs HTML… voire de tableaux imbriqués.

La structure la plus simpliste consiste en une cellule orpheline. La largeur du tableau est conditionnée par l'attribut `width` de cette cellule, cette méthode étant souvent plus fiable que de tenter de fixer une largeur au tableau dans son ensemble :

```
<table>
<tr>
   <td width="600"></td>
</tr>
</table>
```

Par défaut, les tableaux disposent de bordures et de marges internes. Nous allons là encore nous rappeler nos anciens souvenirs d'attributs HTML pour annuler ces propriétés, tout en appliquant une marge interne de 10 pixels au sein de chaque cellule :

```
<table cellspacing="0" cellpadding="10" border="0">
<tr>
   <td width="600"></td>
</tr>
</table>
```

Dans le cas d'un gabarit formé de deux colonnes, il suffit de définir une seconde cellule de tableau dimensionnée :

```
<table cellspacing="0" cellpadding="10" border="0">
<tr>
   <td width="150"></td> <!-- cellule de gauche -->
   <td width="450"></td> <!-- cellule de droite -->
</tr>
</table>
```

L'exercice devient toutefois un peu plus périlleux lorsqu'il s'agit de représenter une structure composée d'une partie d'en-tête, d'une partie centrale avec deux colonnes et d'une partie de pied de page, car la technique la plus robuste consiste à disposer d'un tableau pour chacune des zones, le tout imbriqué dans le tableau global (figure 11-5) !

```
<table cellspacing="0" cellpadding="10" border="0">
<tr>
   <td width="600"> <!-- conteneur global -->
      <table cellspacing="0" cellpadding="10" border="0">
      <tr><td width="600"> en-tête </td> </tr>
      </table>
```

```
      <table cellspacing="0" cellpadding="10" border="0">
      <tr>
         <td width="150"> gauche </td>
         <td width="450"> droite </td>
      </tr>
      </table>
      <table cellspacing="0" cellpadding="10" border="0">
      <tr><td width="600"> pied de page </td> </tr>
      </table>
   </td>
</tr>
</table>
```

C'est un réel plaisir, n'est-ce-pas ?

Figure 11-5

Trois étages ?
Trois tableaux !

Afficher un arrière-plan

Nombreux sont les clients de messagerie qui ignorent les images et les arrière-plans illustrés, nous l'avons évoqué. Saviez-vous toutefois que certains vont même jusqu'à négliger les couleurs de fond sur les éléments, hormis les composants tabulaires ?

Cela signifie que si vous souhaitez appliquer une couleur d'arrière-plan à votre courrier, vous devrez cibler l'une des cellules du tableau à l'aide de l'attribut HTML `bgcolor`, car la teinte sera occultée sur un élément `<div>` et même sur `<body>` :

```
<table cellspacing="0" cellpadding="10" border="0">
<tr>
   <td width="150" bgcolor="#aabbcc">
   <!-- ici le contenu de l'e-mail -->
   </td>
</tr>
</table>
```

La méthode est identique pour l'affichage des images d'arrière-plan, mais rappelez-vous que certains clients de messagerie ne les comprennent pas ; il est donc conseillé de proposer une alternative sous forme de couleur simple :

```
<table cellspacing="0" cellpadding="10" border="0">
<tr>
   <td width="150" background="chemin_de_l_image" bgcolor="#aabbcc">
   <!-- ici le contenu du message -->
```

```
    </td>
</tr>
</table>
```

Caler les éléments

Sans surprise, les attributs HTML de mise en forme – déconseillés dans la plupart des cas de figure – deviennent nos meilleurs alliés pour positionner les contenus au sein du gabarit de message.

Ainsi, vous ne serez pas étonné de rencontrer les exemples suivants :

- `border="0"` : pour annuler les bordures autour d'une cellule ou du tableau entier ;
- `cellpadding="5"` : pour définir la marge interne des cellules à 5 pixels ;
- `cellspacing="10"` : pour régler l'espace entre chaque cellule à 10 pixels ;
- `valign="top"`, `valign="middle"`, `valign="bottom"` : pour aligner verticalement le contenu d'une cellule ;
- `align="left"`, `align="right"`, `align="center"` : pour aligner horizontalement le contenu d'une cellule.

Il devient bien entendu parfaitement envisageable de caler ou de déplacer les éléments à l'aide des marges internes (`padding`) ou externes (`margin`) mais avec les précautions suivantes :

- Les marges externes ne sont pas reconnues par Hotmail ; il sera nécessaire de les « remplacer » par des marges internes.
- Il est parfois risqué de cumuler un positionnement à l'aide des marges CSS et des attributs HTML tels que `cellpadding`.

Étape 2 : styler avec parcimonie

L'architecture générale étant construite à l'aide de tableaux et de cellules, passons aux agréments visuels qui pourront être réalisés via quelques styles CSS employés avec modération.

Styles en ligne et complets

Trop peu de messageries prennent en compte les styles déclarés dans un fichier CSS ou dans l'en-tête du document : toutes les mises en forme devront être apportées en ligne, au sein de la balise via l'attribut HTML `style`.

Ainsi, pour attribuer une taille de police de 20 pixels, une couleur verte et un alignement centré à un titre de niveau 2, vous écrirez le code suivant.

Partie HTML

```
<h2 style="font-size: 20px; color: green; text-align: center">Un titre de niveau 2</h2>
```

Souvenez-vous que les styles relatifs aux polices (couleur, taille, alignement) sont hérités de père en fils. Attribuez autant que possible ces styles aux éléments de cellules plutôt qu'aux parties de contenus (`<h1>`, `<h2>`, `<p>`, `<a>`, etc.) pour des raisons de meilleure prise en charge.

La compréhension des éléments `<div>` et de leurs styles est pour le moins aléatoire : mieux vaut encore réapprendre à manier les tableaux imbriqués pour être sûr de la compatibilité de votre document. A contrario, il semble que les éléments `<span>` bénéficient d'une prise en charge bien plus complète. Ils peuvent sans risque servir à mettre en forme des portions de contenus.

Prenez également note que, dans de nombreux cas, les syntaxes raccourcies des propriétés sont purement et simplement ignorées par les clients de messagerie. Dans cette optique, préférez toujours :

- `margin-right: 10px` à `margin: 0 10px 0 0` ;
- `font-size: 20px` à `font: 20px` ;
- `background-color` à `background` ;
- etc.

Pas de flottants ?

Outlook 2007 et les anciennes versions de Lotus Notes ignorent les propriétés CSS de positionnement et notamment `float`. Pour disposer les éléments tels que les images côte à côte, il est préférable de se rapporter à l'attribut HTML `align` :

```
<img src="image.jpg" align="right" alt="texte alternatif" />
```

Styler les liens

Plusieurs environnements de messagerie écrasent vos mises en forme des liens hypertextes avec leurs propres styles internes.

Il est possible de se soustraire à ce comportement au prix d'une gymnastique peu engageante. Tout d'abord, affectez une couleur à chacun des liens de la sorte :

```
<a href="http://www.alsacreations.com/" style="color:#00ff00">Alsacréations</a>
```

Puis introduisez un élément `<span>` lui-même affublé de la couleur désirée :

```
<a href="http://www.alsacreations.com/" style="color:#00ff00">
<span style="color:#00ff00">Alsacréations</span></a>
```

Cette double sécurité, qui peut paraître superflue, demeure pourtant le meilleur moyen de se prémunir des éventuelles préséances des clients de messagerie.

Astuce : utiliser des gabarits

À présent que vous disposez de tout l'arsenal pour concevoir de superbes mises en page de courriers électroniques, vous n'aurez peut-être qu'une seule envie, compte tenu de la pénibilité de la tâche qui vous attend : tout laisser tomber !

Pour vous dispenser de graves désappointements et de longues heures de tests, je vous invite à découvrir les gabarits existants prémâchés proposés par les sites de références suivants, Mailchimp (figure 11-6) et Campaingmonitor.

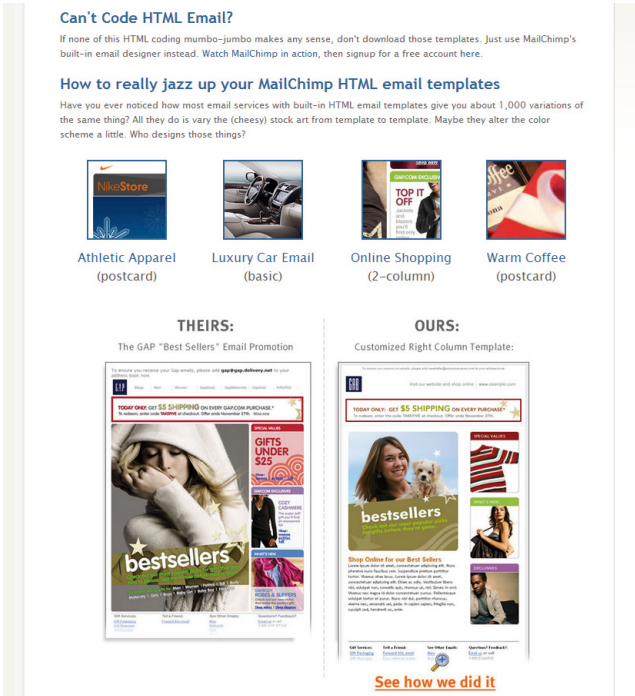
- http://www.mailchimp.com/resources/html_email_templates/
- http://www.campaignmonitor.com/templates/

Ces deux adresses partagent librement et gratuitement des ressources inestimables pour tous les créateurs de courriels : des centaines de kits graphiques (*templates*) déjà fonctionnels et testés sur un vaste panel de clients de messagerie.

Ces gabarits sont présentés par types d'usages (professionnel, *grunge*, simpliste) ou par formats (une ou plusieurs colonnes, type « carte postale ») et n'attendent que vous pour être affinés, personnalisés à votre image et envoyés à vos plus chers clients.

Figure 11-6

Aperçu des templates Mailchimp



Outils pour le créateur de courriels

Parallèlement à l'usage de gabarits graphiques, certains sites en ligne vont s'avérer de redoutables alliés dans votre quête du courriel universel :

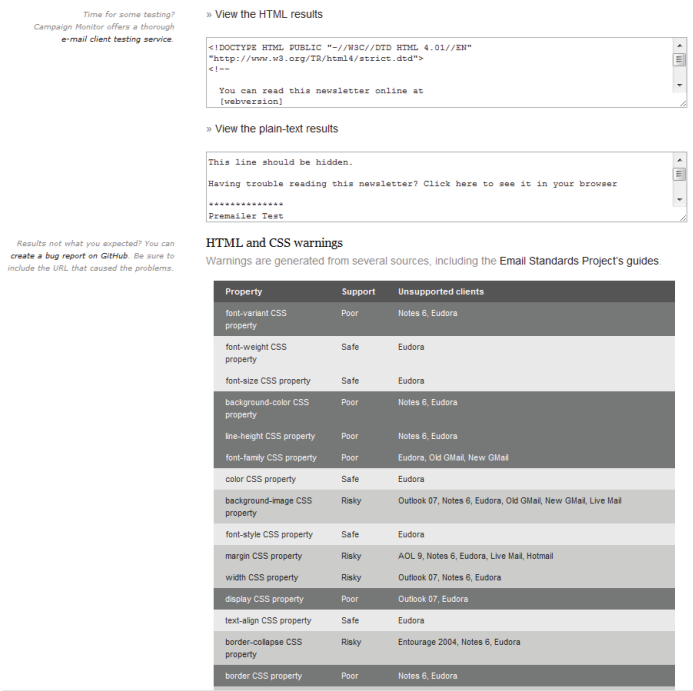
- http://premailer.dialect.ca (figure 11-7)
- http://inlinestyler.torchboxapps.com

Non seulement, ces deux outils vont convertir tous vos styles CSS externes en mises en forme directement en ligne appliquées via des attributs HTML `style`, mais réalisent également d'autres prouesses :

- transformation des URL relatives en URL absolues ;
- déplacement des pseudo-classes telles que `:hover` au sein de la partie `<head>` du document ;
- vérification du résultat sur un large panel d'environnements de messagerie ;
- ajout de texte en préambule tel que « *You can read this newsletter online at* [adresse] » ;
- création d'une version en texte seul pour les clients n'acceptant pas les courriers en HTML.

Figure 11-7

Aperçu de premailer



12

Et les autres périphériques ?

La liste des périphériques gérés selon les spécifications CSS va bien au-delà des quelques aspects détaillés dans les chapitres précédents. Cette dernière partie s'applique à présenter quelques-uns de ces environnements méconnus : `speech`, `projection` et `TV`. Les normes CSS proposent depuis longtemps des solutions et techniques pour ces périphériques destinés à des usages spécifiques, d'autant plus que l'ignorance des navigateurs à leur sujet commence enfin à s'estomper.

Nous venons de traiter trois environnements de restitution de documents HTML, avec une prise en charge malheureusement parfois incomplète chez certains agents utilisateurs : le mobile, l'impression et les clients de messagerie électronique.

La liste des environnements définis par les spécifications W3C va bien au-delà des quelques usages que nous avons développés et analysés. Cette dernière partie de l'ouvrage s'intéresse plus particulièrement à trois types de supports peu connus, notamment parce qu'ils sont encore peu reconnus par les navigateurs :

- le support auditif (`speech`) ;
- la projection sur un support distant (`projection`) ;
- la télévision (`TV`).

Par défaut, une feuille de styles CSS s'applique à tous les supports existants (`all`) et de nombreuses propriétés sont disponibles pour tous les types. Cependant, il ne s'agit pas d'une généralité : certains effets, tels que la taille de la police ou la forme du curseur de la souris, n'ont guère de sens pour une synthèse vocale et les propriétés de sauts de pages ne sont disponibles que sur les supports paginés tels que `print`, `projection` et `TV`.

ATTENTION Un média à la fois !

Il est cohérent de penser qu'un agent utilisateur ne prend en charge qu'un seul environnement à la fois : par exemple, un document ne peut pas être diffusé sur un écran et sur un téléviseur en même temps. Contre toute attente, certains agents utilisateurs transgressent pourtant ce principe. C'est le cas des anciennes versions d'Internet Explorer Mobile qui appliquaient à la fois `screen` et `handheld`, même si vous étiez sur un écran de bureau. C'est aussi le cas de MSN TV, qui accole `screen` à `TV` (voir la section consacrée à la télévision).

Speech : périphériques de restitution vocale

Le support auditif regroupe les recommandations nommées `speech` depuis CSS 2.1 (précédemment `aural`). Il a pour vocation de restituer des styles particuliers (prononciation, voix, pauses, volume, etc.) lorsque la page web est lue par une synthèse vocale, l'un des outils de prédilection des internautes malvoyants ou aveugles.

La mise en application de la feuille de styles dédiée est très simple :

```
<link rel="stylesheet" media="speech" href="styles-audio.css" type="text/css">
```

Un environnement critiqué

Malgré leur souplesse d'utilisation et leur richesse, les propriétés `speech` proposées par le W3C présentent un handicap sérieux : non seulement elles sont mal reconnues par les navigateurs, mais elles sont généralement contrecarrées par les logiciels lecteurs d'écrans eux-mêmes (Jaws ou Home Page Reader, par exemple).

En effet, chaque synthèse vocale détermine ses propres algorithmes d'analyse et de prononciation des contenus d'un document, non basés sur les CSS.

La question de savoir si l'environnement `speech` est à mettre aux mains des concepteurs web se pose également, puisqu'il rassemble des fonctionnalités généralement peu maîtrisées, à moins d'être un expert en accessibilité.

Enfin, les spécialistes en la matière offrent un débat intéressant sur le sujet : et si le support CSS `speech` n'était pas purement et simplement nuisible ?

En effet, les logiciels de lecture d'écrans disposent de préférences naturelles en ce qui concerne les voix, la vitesse d'élocution ou encore les différentes prononciations. Transférer une partie de ces contrôles naturels aux mains d'une surcouche CSS pourrait avoir des répercussions potentiellement gênantes ou même dangereuses.

Le débat sur la technologie à employer demeure ouvert et je vous conseille vivement de bien étudier la question avant d'imposer à vos visiteurs des styles auditifs qui pourraient écraser leurs préférences habituelles.

OUTIL Liste des lecteurs d'écrans

L'adresse suivante propose une liste à jour des différentes synthèses vocales ainsi que des techniques pour les cibler : type de terminal, styles importés, etc.

▸ http://css-discuss.incutio.com/wiki/Screenreader_Visibility

Quelques propriétés de speech

Les spécifications CSS 2.1 et CSS 3 proposent ou redéfinissent un petite trentaine de propriétés liées à l'environnement auditif. Je vous présente les plus caractéristiques en guise d'introduction et vous suggère de lire les documents officiels du W3C si vous souhaitez explorer plus avant ce thème :

- **`volume`** : cette propriété permet de définir le réglage du volume sonore, soit à l'aide d'un nombre situé entre 0 (minimum audible) et 100 (maximum de confort), soit via des mots-clés tels que `silent`, `soft`, `médium`, `loud`, `x-loud`…
- **`speak`** : cette propriété spécifie si un texte aura un rendu auditif et, le cas échéant, comment il sera rendu. La propriété `speak` autorise les valeurs suivantes : `none` (aucun rendu auditif), `normal` ou `spell-out` (lecture lettre par lettre).
- **`pause`** : les propriétés de pause (`pause-before` et `pause-after`) spécifient la pause à observer avant que le contenu d'un élément ne soit lu, ou après qu'il l'ait été.
- **`cue`** : les propriétés de signal (`cue-before` et `cue-after`) permettent de jouer des sons avant ou après un élément.
- **`voice-family`** : correspond à des types de voix tels que `male`, `female` ou `child`. Les valeurs sont séparées par des virgules et par ordre de priorité (comme `font-family`).
- **`pitch`** : correspond à la fréquence de la voix, qui est de 120 Hz en moyenne pour un homme et 210 Hz chez une femme.

Prise en charge de speech

La prise en charge du support auditif est actuellement quasi inexistante. Parmi les navigateurs communs, seul Opera sort du lot en exploitant à la fois les propriétés CSS 2.1 et un bon nombre des propositions CSS 3.

Voici la liste des différents agents utilisateurs reconnaissant cet environnement particulier :

- Opera ;
- FireVox : une extension pour Mozilla Firefox ;
- Emacspeak : un logiciel audio développé pour GNU/Linux ;
- pwWebSpeak : un navigateur textuel (pas mis à jour depuis 2001) ;
- Fonix SpeakThis : un service de reconnaissance vocale.

Projection : restitution sur grand écran

Le périphérique `projection` s'adresse aux documents destinés à être projetés sur grand écran et bénéficie de l'ensemble des fonctionnalités prévues pour les supports paginés (imprimante et télévision).

Il se déclare de la même manière que les autres supports :

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

Il est également possible d'appeler le fichier en CSS basé sur la règle `@import` :

```
@import url(projection.css) projection;
```

On peut enfin incorporer directement les styles en ligne au sein d'un fichier CSS commun à plusieurs environnements de restitution :

```
@media projection {
body {
  font-size: 130%;
  color: black;
  background: white;
  }
}
```

Dans la pratique, il sera parfois utile d'adresser le même fichier de mise en forme à plusieurs types de supports :

Une feuille de styles pour écran, projection et TV

```
<link rel="stylesheet" media="screen, projection, tv" href="styles.css" type="text/css">
```

Quel usage ?

Le principal intérêt de ce périphérique réside dans les ajustements que l'on peut apporter lorsqu'un site web ou un document est destiné à être projeté dans des conditions qui ne sont pas forcément maîtrisées (faible contraste, lumière gênante, surface non unie…).

Il devient alors possible de renforcer les contrastes (couleur de fond blanche, couleur de texte noire) et les tailles des contenus afin de mieux s'adapter au contexte de projection.

Ce dispositif permet également de bénéficier d'un affichage différent lorsque la présentation est à la fois projetée sur un mur et sur l'écran de l'orateur : les aides et commentaires personnels s'afficheront alors uniquement sur l'écran du conférencier.

```
@media screen {
  .personal {display: block;} /* notes affichées à l'écran */
}
@media projection {
  .personal {display: none;} /* … mais masquées à l'auditoire */
}
```

Compatibilité

Actuellement, seul le navigateur Opera prend en charge ce format, que l'on peut actionner en affichant la page web en plein écran (dans le menu Page>Plein écran ou via la touche F11).

OUTILS Comment faire sans Opera ?

Si vous n'êtes pas en mesure de projeter votre présentation à l'aide du navigateur Opera, il vous reste la possibilité de profiter de quelques outils basés sur JavaScript et CSS conçus par des développeurs astucieux :

- Eric Meyer a déployé son format de présentation S5 basé sur OperaShow.

▸ http://meyerweb.com/eric/tools/s5/structure-ref.html

- Paul Rouget a développé DZ Slides avec des technologies HTML 5 et CSS 3, fonctionnant sur les navigateurs récents tels que Firefox 4 et WebKit.

▸ http://paulrouget.com/dzslides/

TV : environnements télévisuels

Le périphérique TV a été conçu pour les environnements télévisuels au format couleur, en basse résolution (les formats PAL/SECAM affichent généralement une définition de 720 × 576 pixels) et sonorisés.

Compatibilité

Une fois n'est pas coutume, seuls quelques moteurs reconnaissent le format TV dans des cas très particuliers :

- Opera (Presto), proposé sur la plate-forme de jeu Wii de Nintendo (wii.opera.com) ;
- Android (WebKit) sur Google TV, FreeBox v6 et certaines télévisions « Web » telles que Samsung ;
- Netfront sur PlayStation 3 ;
- Internet Explorer sur XBOX 360.

D'autres produits plus exotiques et confidentiels permettent cependant de tirer profit de ce support, à l'exemple de MSN TV (ex-WebTV), racheté par Microsoft et offrant un environnement Internet ainsi qu'un client de messagerie à un poste de télévision classique.

Le format télévisuel

À sa création vers 1998, le périphérique TV fut adapté à un contexte technologique en pleine évolution où la résolution d'un écran de télévision était – au mieux – comparable à celle d'un ordinateur de l'époque, à savoir 640 ou 800 pixels de large. L'arrivée de la télévision haute définition (HD) est en passe de changer complètement la donne et de rendre quasi obsolète cette détection du terminal.

Certaines spécificités de ce support demeurent toutefois encore parfaitement d'actualité aujourd'hui. C'est le cas par exemple des formats d'affichage en 4/3 et 16/9 que l'on peut définir à l'aide des styles CSS.

Il est également recommandé d'adapter les contrastes, dont la gestion est très particulière sur un écran de téléviseur : contrairement à d'autres supports, il est conseillé d'éviter les contrastes trop intenses ou les larges surfaces d'une couleur unie telle que le rouge vif.

Le navigateur Opera sur Wii

Le navigateur Internet de la console de jeu Wii est basé sur le moteur de rendu d'Opera 9 et bénéficie, à quelques différences près, des mêmes aptitudes : HTML, CSS, JavaScript (y compris XMLHttpRequest et Ajax) et Adobe Flash.

Opera sur Wii reconnaît parfaitement le périphérique `TV` et l'applique s'il est sollicité. Cependant, pour s'aligner au même niveau que tous les autres navigateurs ne reconnaissant pas ce format, il choisit les styles dédiés au support `screen` si aucune mise en forme n'est prévue spécifiquement pour la télévision.

De plus, conformément aux spécifications, Opera n'applique qu'un seul environnement à la fois : si des styles sont prévus pour `screen` et `TV`, seul le dernier est pris en compte.

La télécommande Wii (*Wii remote*) est considérée comme un gestionnaire de souris classique et dispose des fonctionnalités équivalentes : par exemple, appuyer sur les boutons de la télécommande correspond aux clics réalisables sur la souris ou aux défilements des ascenseurs.

Il est possible d'exploiter tous les formats d'image reconnus par Opera, à savoir JPEG, GIF, PNG (avec transparence alpha), mais aussi SVG et `<canvas>`. En revanche, seules deux fontes sont disponibles sur Wii : une police générique sans empattements nommée Wii NTLG PGothic et une police de type monospace, Wii NTLG Gothic. Il est inutile par conséquent de tenter de faire apparaître d'autres familles de polices sur l'écran.

La console Wii peut être affichée en format classique (4/3) ou en mode panoramique (16/9).

Fort heureusement, Opera sur Wii prend en charge les Media Queries CSS 3 et s'adapte aux différents formats d'affichage. En voici un exemple :

Adaptation au mode d'affichage et à la résolution

```
@media tv and (device-aspect-ratio: 16/9) and (max-width: 800px){
  /* ici les styles adaptés */
}
```

Annexes

A

Liste de toutes les propriétés CSS (CSS 1, CSS 2, CSS 3)

Ce tableau liste l'ensemble des propriétés CSS actuelles ou ayant existé. Ce travail de fourmi a été réalisé par Jens O. Meiert, que je remercie vivement d'avoir accepté de figurer au sein de cet ouvrage.

▸ http://meiert.com

Propriété	CSS 1	CSS 2	CSS 2.1	CSS 3	Valeur initiale
`alignment-adjust`				x	`auto`
`alignment-baseline`				x	`baseline`
`animation`				x	selon propriétés individuelles
`animation-delay`				x	0
`animation-direction`				x	`normal`
`animation-duration`				x	0
`animation-iteration-count`				x	1
`animation-name`				x	`none`
`animation-play-state`				x	`running`
`animation-timing-function`				x	`ease`
`appearance`				x	`normal`

Propriété	CSS 1	CSS 2	CSS 2.1	CSS 3	Valeur initiale
azimuth		x	x	?	center
backface-visibility				x	visible
background	x	x	x	x	selon propriétés individuelles
background-attachment	x	x	x	x	scroll
background-break				x	continuous
background-clip				x	border
background-color	x	x	x	x	transparent
background-image	x	x	x	x	none
background-origin				x	padding
background-position	x	x	x	x	0% 0%
background-repeat	x	x	x	x	repeat
background-size				x	auto
baseline-shift				x	baseline
binding				x	none
bookmark-label				x	content
bookmark-level				x	none
bookmark-target				x	self
border	x	x	x	x	selon propriétés individuelles
border-bottom	x	x	x	x	selon propriétés individuelles
border-bottom-color		x	x	x	couleur courante
border-bottom-left-radius				x	0
border-bottom-right-radius				x	0
border-bottom-style		x	x	x	none
border-bottom-width	x	x	x	x	medium
border-collapse		x	x	?	separate
border-color	x	x	x	x	selon propriétés individuelles
border-image				x	none
border-image-outset				x	0
border-image-repeat				x	stretch
border-image-slice				x	100,00%
border-image-source				x	none

Propriété	CSS 1	CSS 2	CSS 2.1	CSS 3	Valeur initiale
border-image-width				x	1
border-left	x	x	x	x	selon propriétés individuelles
border-left-color		x	x	x	couleur courante
border-left-style		x	x	x	none
border-left-width	x	x	x	x	medium
border-length				x	auto
border-radius				x	0
border-right	x	x	x	x	selon propriétés individuelles
border-right-color		x	x	x	couleur courante
border-right-style		x	x	x	none
border-right-width	x	x	x	x	medium
border-spacing		x	x	?	0
border-style	x	x	x	x	selon propriétés individuelles
border-top	x	x	x	x	selon propriétés individuelles
border-top-color		x	x	x	couleur courante
border-top-left-radius				x	0
border-top-right-radius				x	0
border-top-style		x	x	x	none
border-top-width	x	x	x	x	medium
border-width	x	x	x	x	selon propriétés individuelles
bottom		x	x	?	auto
box-align				x	stretch
box-decoration-break				x	slice
box-direction				x	normal
box-flex				x	0.0
box-flex-group				x	1
box-lines				x	single
box-ordinal-group				x	1
box-orient				x	inline-axis
box-pack				x	start
box-shadow				x	none

Propriété	CSS 1	CSS 2	CSS 2.1	CSS 3	Valeur initiale
box-sizing				x	content-box
caption-side		x	x	?	top
clear	x	x	x	x	none
clip		x	x	?	auto
color	x	x	x	x	selon propriétés système
color-profile				x	auto
column-break-after				x	auto
column-break-before				x	auto
column-count				x	auto
column-fill				x	balance
column-gap				x	normal
column-rule				x	selon propriétés individuelles
column-rule-color				x	couleur courante
column-rule-style				x	medium
column-rule-width				x	medium
column-span				x	1
column-width				x	auto
columns				x	selon propriétés individuelles
content		x	x	x	normal
counter-increment		x	x	x	none
counter-reset		x	x	x	none
crop				x	auto
cue		x	x	x	selon propriétés individuelles
cue-after		x	x	x	none
cue-before		x	x	x	none
cursor		x	x	x	auto
direction		x	x	?	ltr
display	x	x	x	x	inline
display-model				x	?
display-role				x	?
dominant-baseline				x	auto

Propriété	CSS 1	CSS 2	CSS 2.1	CSS 3	Valeur initiale
drop-initial-after-adjust				x	text-after-edge
drop-initial-after-align				x	baseline
drop-initial-before-adjust				x	text-before-edge
drop-initial-before-align				x	caps-height
drop-initial-size				x	auto
drop-initial-value				x	initial
elevation		x	x	?	level
empty-cells		x	x	?	show
fit				x	fill
fit-position				x	0% 0%
float	x	x	x	x	none
float-offset				x	0 0
font	x	x	x	x	selon propriétés individuelles
font-family	x	x	x	x	selon propriétés système
font-size	x	x	x	x	medium
font-size-adjust		x		x	none
font-stretch				x	normal
font-style	x	x	x	x	normal
font-variant	x	x	x	x	normal
font-weight	x	x	x	x	normal
grid-columns				x	none
grid-rows				x	none
hanging-punctuation				x	none
height	x	x	x	x	auto
hyphenate-after				x	auto
hyphenate-before				x	auto
hyphenate-character				x	auto
hyphenate-lines				x	no-limit
hyphenate-resource				x	none
hyphens				x	manual
icon				x	auto

Propriété	CSS 1	CSS 2	CSS 2.1	CSS 3	Valeur initiale
image-orientation				x	auto
image-resolution				x	normal
inline-box-align				x	last
left		x	x	?	auto
letter-spacing	x	x	x	x	normal
line-height	x	x	x	x	normal
line-stacking				x	selon propriétés individuelles
line-stacking-ruby				x	exclude-ruby
line-stacking-shift				x	consider-shifts
line-stacking-strategy				x	inline-line-height
list-style	x	x	x	x	selon propriétés individuelles
list-style-image	x	x	x	x	none
list-style-position	x	x	x	x	outside
list-style-type	x	x	x	x	disc
margin	x	x	x	x	selon propriétés individuelles
margin-bottom	x	x	x	x	0
margin-left	x	x	x	x	0
margin-right	x	x	x	x	0
margin-top	x	x	x	x	0
mark				x	selon propriétés individuelles
mark-after				x	none
mark-before				x	none
marks		x		x	none
marquee-direction				x	forward
marquee-play-count				x	1
marquee-speed				x	normal
marquee-style				x	scroll
max-height		x	x	x	none
max-width		x	x	x	none
min-height		x	x	x	0
min-width		x	x	x	0

Propriété	CSS 1	CSS 2	CSS 2.1	CSS 3	Valeur initiale
move-to				x	normal
nav-down				x	auto
nav-index				x	auto
nav-left				x	auto
nav-right				x	auto
nav-up				x	auto
opacity				x	1
orphans		x	x	x	2
outline		x	x	x	selon propriétés individuelles
outline-color		x	x	x	invert
outline-offset				x	0
outline-style		x	x	x	none
outline-width		x	x	x	medium
overflow		x	x	x	selon propriétés individuelles
overflow-style				x	auto
overflow-x				x	visible
overflow-y				x	visible
padding	x	x	x	x	selon propriétés individuelles
padding-bottom	x	x	x	x	0
padding-left	x	x	x	x	0
padding-right	x	x	x	x	0
padding-top	x	x	x	x	0
page		x		x	auto
page-break-after		x	x	x	auto
page-break-before		x	x	x	auto
page-break-inside		x	x	x	auto
page-policy				x	start
pause		x	x	x	selon propriétés système
pause-after		x	x	x	selon propriétés système
pause-before		x	x	x	selon propriétés système
perspective				x	none

Propriété	CSS 1	CSS 2	CSS 2.1	CSS 3	Valeur initiale
perspective-origin				x	50% 50%
phonemes				x	selon propriétés système
pitch		x	x	?	medium
pitch-range		x	x	?	50
play-during		x	x	?	auto
position		x	x	?	static
presentation-level				x	0
punctuation-trim				x	none
quotes		x	x	x	selon propriétés système
rendering-intent				x	auto
resize				x	none
rest				x	selon propriétés système
rest-after				x	selon propriétés système
rest-before				x	selon propriétés système
richness		x	x	?	50
right		x	x	?	auto
rotation				x	0
rotation-point				x	50% 50%
ruby-align				x	auto
ruby-overhang				x	none
ruby-position				x	before
ruby-span				x	none
size		x		x	auto
speak		x	x	x	normal
speak-header		x	x	?	once
speak-numeral		x	x	?	continuous
speak-punctuation		x	x	?	none
speech-rate		x	x	?	medium
stress		x	x	?	50
string-set				x	none
tab-side				x	top

Propriété	CSS 1	CSS 2	CSS 2.1	CSS 3	Valeur initiale
table-layout		x	x	?	auto
target				x	selon propriétés individuelles
target-name				x	current
target-new				x	window
target-position				x	above
text-align	x	x	x	x	start
text-align-last				x	start
text-decoration	x	x	x	?	none
text-emphasis				x	none
text-height				x	auto
text-indent	x	x	x	x	0
text-justify				x	auto
text-outline				x	none
text-replace				x	none
text-shadow		x		x	none
text-transform	x	x	x	?	none
text-wrap				x	normal
top		x	x	?	auto
transform				x	none
transform-origin				x	50% 50% 0
transform-style				x	flat
transition				x	selon propriétés individuelles
transition-delay				x	0
transition-duration				x	0
transition-property				x	all
transition-timing-function				x	ease
unicode-bidi		x	x	?	normal
vertical-align	x	x	x	x	baseline
visibility		x	x	x	visible
voice-balance				x	center
voice-duration				x	selon propriétés système

Propriété	CSS 1	CSS 2	CSS 2.1	CSS 3	Valeur initiale
voice-family		x	x	x	selon propriétés système
voice-pitch				x	medium
voice-pitch-range				x	selon propriétés système
voice-rate				x	selon propriétés système
voice-stress				x	moderate
voice-volume				x	medium
volume		x	x	?	medium
white-space	x	x	x	x	normal
white-space-collapse				x	collapse
widows		x	x	x	2
width	x	x	x	x	auto
word-break				x	normal
word-spacing	x	x	x	x	normal
word-wrap				x	normal
z-index		x	x	?	auto
Total (sur 246)	**53**	**120**	**115**	**246**	

B

Prise en charge de HTML 5 et CSS 3

Voici un tableau montrant la prise en charge des différentes propriétés CSS 3 et applications HTML 5 par les différents navigateurs actuels. Cette ressource a été concoctée et mise à disposition par Jim Morrison, de l'agence web *Deep Blue Sky*. Je le remercie pour son partage.

▸ http://findmebyip.com/litmus

Propriétés CSS 3

Système d'exploitation	Mac OS						Windows											% Prise en charge [échelon]
Navigateur	Chrome		Firefox	Opera	Safari		Chrome			Firefox		Opera	Safari	IE				
Version	5	6	3,6	10,6	5	4	5	6	7	3,6	4,02	10,6	5	6	7	8	9	
RGBa	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	82 % [excellent]
HSLa	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	N	81 % [excellent]
Background Size	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	78 % [excellent]
Backgrounds Multiples	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	78 % [excellent]
Border Image	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	N	80 % [excellent]
Border Radius	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	82 % [excellent]
Box Shadow	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	81 % [excellent]
Opacity	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	82 % [excellent]
CSS Animations	O	O	N	N	O	O	O	O	O	N	N	N	O	N	N	N	N	36 % [limité]
CSS Columns	O	O	O	N	O	O	O	O	O	O	O	N	O	N	N	N	N	77 % [excellent]
CSS Gradients	O	O	O	N	O	O	O	O	O	O	O	N	O	N	N	N	N	73 % [bon]
CSS Reflections	O	O	N	N	O	O	O	O	O	N	N	N	O	N	N	N	N	36 % [limité]
CSS Transforms	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	N	80 % [excellent]
CSS Transforms 3D	N	N	N	N	O	N	N	N	N	N	N	N	O	N	N	N	N	11 % [faible]
CSS Transitions	O	O	N	O	O	O	O	O	O	N	O	O	O	N	N	N	N	42 % [limité]
CSS @font-face	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	95 % [excellent]

Sélecteurs CSS 3

Système d'exploitation	Mac OS						Windows											% Prise en charge [échelon]
Navigateur	Chrome		Firefox	Opera	Safari		Chrome			Firefox		Opera	Safari	IE				
Version	5	6	3,6	10,6	5	4	5	6	7	3,6	4,02	10,6	5	6	7	8	9	
CSS3 [^=attribut]	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	98 % [excellent]
CSS3 [$=attribut]	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	98 % [excellent]
CSS3 [*=attribut]	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	98 % [excellent]
CSS3: `root`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	83 % [excellent]
CSS3 `:nth-child`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	82 % [excellent]
CSS3 `:nth-last-child`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	82 % [excellent]
CSS3 `:nth-of-type`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	82 % [excellent]
CSS3 `:nth-last-of-type`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	82 % [excellent]
CSS3 `:last-child`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	83 % [excellent]
CSS3 `:first-of-type`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	82 % [excellent]
CSS3 `:last-of-type`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	82 % [excellent]
CSS3 `:only-child`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	83 % [excellent]
CSS3 `:only-of-type`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	82 % [excellent]
CSS3 `:empty`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	83 % [excellent]
CSS3 `:target`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	82 % [excellent]
CSS3 `:enabled`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	83 % [excellent]
CSS3 `:disabled`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	83 % [excellent]
CSS3 `:checked`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	83 % [excellent]
CSS3 `:not`	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	83 % [excellent]
CSS3 E ~ F	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	98 % [excellent]

Applications web HTML 5

Système d'exploitation	Mac OS						Windows											% Prise en charge [échelon]
Navigateur	Chrome		Firefox	Opera	Safari		Chrome			Firefox		Opera	Safari	IE				
Version	5	6	3,6	10,6	5	4	5	6	7	3,6	4,02	10,6	5	6	7	8	9	
Local Storage	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	O	O	92 % [excellent]
Session Storage	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	O	O	93 % [excellent]
Post Message	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	O	O	94 % [excellent]
Offline Applications	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	N	80 % [excellent]
Workers	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	N	77 % [excellent]
Query Selector	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	85 % [excellent]
WebSQL Database	O	O	N	O	O	O	O	O	O	N	N	O	O	N	N	N	N	38 % [limité]
IndexDB Database	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	
Drag and Drop	O	O	O	N	O	O	O	O	O	O	O	N	O	O	O	O	O	93 % [excellent]
Hash Change (Event)	O	O	O	N	O	N	O	O	O	O	O	N	O	N	N	O	O	80 % [excellent]
History Management	O	O	N	N	O	N	O	O	O	N	O	N	O	N	N	N	N	36 % [limité]
Web Sockets	O	O	N	N	O	N	O	O	O	N	O	N	O	N	N	N	N	35 % [limité]
SMIL	O	O	N	O	O	O	O	O	O	N	O	O	O	N	N	N	O	43 % [limité]
SVG	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	81 % [excellent]
SVG Clipping Paths	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	81 % [excellent]
GeoLocation	O	O	O	O	O	N	O	O	O	O	O	O	O	N	N	N	N	79 % [excellent]

Contenu multimédia HTML 5

Système d'exploitation	Mac OS						Windows											% Prise en charge [échelon]
Navigateur	Chrome		Firefox	Opera	Safari		Chrome			Firefox		Opera	Safari	IE				
Version	5	6	3,6	10,6	5	4	5	6	7	3,6	4,02	10,6	5	6	7	8	9	
Canvas	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	83 % [excellent]
Canvas Text	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	81 % [excellent]
Audio	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	80 % [excellent]
Video	O	O	O	O	O	O	O	O	O	O	O	O	O	N	N	N	O	80 % [excellent]

Codecs audio HTML 5

Système d'exploitation	Mac OS						Windows											% Prise en charge [échelon]
Navigateur	Chrome		Firefox	Opera	Safari		Chrome			Firefox		Opera	Safari	IE				
Version	5	6	3,6	10,6	5	4	5	6	7	3,6	4,02	10,6	5	6	7	8	9	
Audio: ogg	O	O	O	O	N	N	O	O	O	O	O	O	N	N	N	N	N	68 % [bon]
Audio: mp3	O	O	N	N	O	O	O	O	O	N	N	N	O	N	N	N	O	35 % [limité]
Audio: wav	N	N	O	O	O	O	N	N	N	O	O	O	O	N	N	N	N	54 % [bon]
Audio: AAC	O	O	N	N	O	O	O	O	O	N	N	N	O	N	N	N	O	35 % [limité]

Codecs vidéo HTML 5

Système d'exploitation	Mac OS						Windows											% Prise en charge [échelon]
Navigateur	Chrome		Firefox	Opera	Safari		Chrome			Firefox		Opera	Safari	IE				
Version	5	6	3,6	10,6	5	4	5	6	7	3,6	4,02	10,6	5	6	7	8	9	
Video: ogg	O	O	O	O	N	N	O	O	O	O	O	O	N	N	N	N	N	68 % [bon]
Video: H.264	O	O	N	N	O	O	O	O	O	N	N	N	O	N	N	N	O	34 % [limité]
Video: WebM	N	O	N	O	N	N	N	O	O	N	O	O	N	N	N	N	N	18 % [faible]

Types de champs de formulaires HTML 5

Système d'exploitation	Mac OS						Windows											% Prise en charge [échelon]
Navigateur	Chrome		Firefox	Opera	Safari		Chrome			Firefox		Opera	Safari	IE				
Version	5	6	3,6	10,6	5	4	5	6	7	3,6	4,02	10,6	5	6	7	8	9	
Form: Search	O	O	N	N	O	O	O	O	O	N	O	N	O	N	N	N	N	39 % [limité]
Form: Phone	O	O	N	N	O	O	O	O	O	N	O	N	O	N	N	N	N	38 % [limité]
Form: URL	O	O	N	O	O	N	O	O	O	N	N	O	O	N	N	N	N	38 % [limité]
Form: Email	O	O	N	O	O	N	O	O	O	N	N	O	O	N	N	N	N	38 % [limité]
Form: DateTime	N	N	N	O	N	N	N	N	N	N	N	O	N	N	N	N	N	4 % [faible]
Form: Date	N	N	N	O	N	N	N	N	N	N	N	O	N	N	N	N	N	4 % [faible]
Form: Month	N	N	N	O	N	N	N	N	N	N	N	O	N	N	N	N	N	4 % [faible]
Form: Week	N	N	N	O	N	N	N	N	N	N	N	O	N	N	N	N	N	4 % [faible]
Form: Time	N	N	N	O	N	N	N	N	N	N	N	O	N	N	N	N	N	4 % [faible]
Form: LocalTime	N	N	N	O	N	N	N	N	N	N	N	O	N	N	N	N	N	4 % [faible]
Form: Number	N	N	N	O	N	N	N	N	N	N	N	O	N	N	N	N	N	4 % [faible]
Form: Range	O	O	N	O	O	N	O	O	O	N	N	O	O	N	N	N	N	37 % [limité]
Form: Colour	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	

Attributs de formulaires HTML 5

Système d'exploitation	Mac OS						Windows											% Prise en charge [échelon]
Navigateur	Chrome		Firefox	Opera	Safari		Chrome			Firefox		Opera	Safari	IE				
Version	5	6	3,6	10,6	5	4	5	6	7	3,6	4,02	10,6	5	6	7	8	9	
Form: Autocomplete	N	N	N	O	N	N	N	N	N	N	N	O	N	N	N	N	N	4 % [faible]
Form: Autofocus	O	O	N	O	O	O	O	O	O	N	O	O	O	N	N	N	N	42 % [limité]
Form: List	N	N	N	O	O	N	N	N	N	N	N	O	O	N	N	N	N	13 % [faible]
Form: Placeholder	O	O	N	N	O	O	O	O	O	N	O	N	O	N	N	N	N	38 % [limité]
Form: Min	O	O	N	O	O	N	O	O	O	N	N	O	O	N	N	N	N	38 % [limité]
Form: Max	O	O	N	O	O	N	O	O	O	N	N	O	O	N	N	N	N	38 % [limité]
Form: Multiple	O	O	O	N	O	O	O	O	O	O	O	N	O	N	N	N	N	72 % [bon]
Form: Pattern	O	O	N	O	O	N	O	O	O	N	N	O	O	N	N	N	N	39 % [limité]
Form: Required	O	O	N	O	O	N	O	O	O	N	N	O	O	N	N	N	N	39 % [limité]
Form: Step	O	O	N	O	O	N	O	O	O	N	N	O	O	N	N	N	N	38 % [limité]

C

Ressources

Événements et conférences

Paris Web : ce cycle de conférences annuel est généralement organisé au mois d'octobre. Principalement francophone, ses sujets de prédilection tournent autour des bonnes pratiques du Web, des standards et de l'accessibilité. On y rencontre les grandes figures (Dave Shea en 2010) qui rendent notre métier universel et abordable. On peut y rencontrer Daniel Glazman, Tristan Nitot, Éric Daspet, Karl Dubost, Nicole Sullivan, Christian Heilmann, Matt May, Molly Holzschlag. Incontournable ! (figure C-1)

- http://www.paris-web.fr

LeWeb : autre événement annuel parisien, plus orienté affaires, marketing et « Web 2.0 ».

- http://www.leweb.net

Confoo : séminaire de conférences se déroulant en mars à Montréal et originellement axé sur les technologies PHP, Python, Ruby, Java et .NET, il s'ouvre peu à peu aux standards web.

- http://www.confoo.ca

@media : c'est la plus grande conférence londonienne dédiée à la conception web et au mobile. De grands orateurs s'y retrouvent : Andy Clarke, John Resig, Steve Souders, Bruce Lawson.

▸ http://atmedia.webdirections.org

Front-Trends : cette conférence internationale se tient à Varsovie en Pologne (fin octobre) et regroupe de grands noms de la conception web *front-end*.

▸ http://front-trends.com

An Event Apart : il s'agit d'un ensemble d'événements annuels, organisés en moyenne une fois tous les deux mois aux États-Unis (Boston, Atlanta, Minneapolis, Washington) et très orientés « technologies d'avant-garde ».

▸ http://aneventapart.com

Future Of Web Design : cette autre conférence américaine (New York) est consacrée à la conception web et à son futur. Vous y croiserez peut-être Cameron Moll, Molly Holzschlag ou encore Doug Bowman.

▸ http://futureofwebdesign.com

Et… la **Kiwi Party** ! Cette petite rencontre strasbourgeoise printanière sans prétention que nous organisons entre quelques « Alsanautes » se trouve être une bonne occasion pour discuter et boire à la santé des standards web. L'édition 2011, prévue en avril, rassemblera plus de 100 spectateurs.

▸ www.kiwiparty.fr

Figure C-1

Paris Web

Ressources en lignes

Ressources francophones

Généralités

Openweb : cette référence historique francophone sur les standards du Web a été créée en 2002. Même si son rythme s'est quelque peu essoufflé depuis, les contenus demeurent extrêmement fiables.

> http://openweb.eu.org

Pompage : ce site de traduction en français d'articles sur la conception web faisant référence dans le monde est un véritable *Reader's Digest* de ce qui se fait de mieux autour de nous (figure C-2).

> http://www.pompage.net

Le Site du Zéro : c'est une excellente ressource pour les débutants complets dans des domaines aussi variés que HTML et CSS, mais également C++, PHP, LaTex, Linux ou encore Blender et Maya.

> http://www.siteduzero.com

Figure C-2

Pompage

CSS, CSS 3

Mammouthland : ces cours en ligne, gratuits et libres, pour débuter avec le langage CSS, sont écrits avec passion et justesse par une sémillante enseignante, Pascale Lambert-Charreteur (figure C-3).

> http://css.mammouthland.net

CSS 2 sur Yoyodesign : vous y trouverez la documentation officielle des spécifications CSS 2.1 du W3C traduite en français.

> http://www.yoyodesign.org/doc/w3c/css2/cover.html#minitoc

Validateur CSS : le Validateur CSS officiel du W3C.

> http://jigsaw.w3.org/css-validator

Figure C-3

Mammouthland

HTML 5

HTML 5 par l'exemple : un tableau comparatif et interactif sur la prise en charge de HTML 5 par les navigateurs. Très à jour !

> http://w3c.html5.free.fr

Docteur HTML 5 : voici la version traduite en français du site éponyme de référence sur l'apprentissage de HTML 5 (figure C-4).

> http://docteurhtml5.com

HTML 5 aujourd'hui : traduction d'un article du WHATWG (*Web Hypertext Application Technology Work Group*) : « *Presentation : How HTML5 can be used today* ».

> http://blog.whatwg.org/html5-aujourd-hui

Figure C-4

Docteur HTML 5

Accessibilité

Introduction à l'accessibilité par Openweb

> http://openweb.eu.org/articles/intro_accessibilite/

Plongez dans l'accessibilité

> http://www.la-grange.net/accessibilite/

Règles pour l'accessibilité des contenus Web WCAG2

- http://www.w3.org/Translations/WCAG20-fr/

Une vidéo nommée « l'ordinateur des aveugles »

- http://www.pyrat.net/Film-L-ordinateur-des-aveugles.html

L'accessibilité pour les personnes malvoyantes

- http://membres.lycos.fr/dbarzin/

Une base de données des aides techniques pour handicapés

- http://www.handicat.com

Toute l'actualité dans le domaine de l'accessibilité

- http://planete-accessibilite.com

Web mobile

Design de poche : porter votre site web au petit écran

- http://www.pompage.net/pompe/petitsecrans/

Comment faire un site web mobile ?

- http://www.zdnet.fr/blogs/2006/12/07/mobile-ready-mobi-comment-faire-un-site-web-mobile/

Résumé des Web mobile Best Practices 1.0 (figure C-5)

- http://www.w3.org/2007/02/mwbp_flip_cards.html.fr

Figure C-5

Les « flipcards » webmobile du W3C

Ressources anglophones

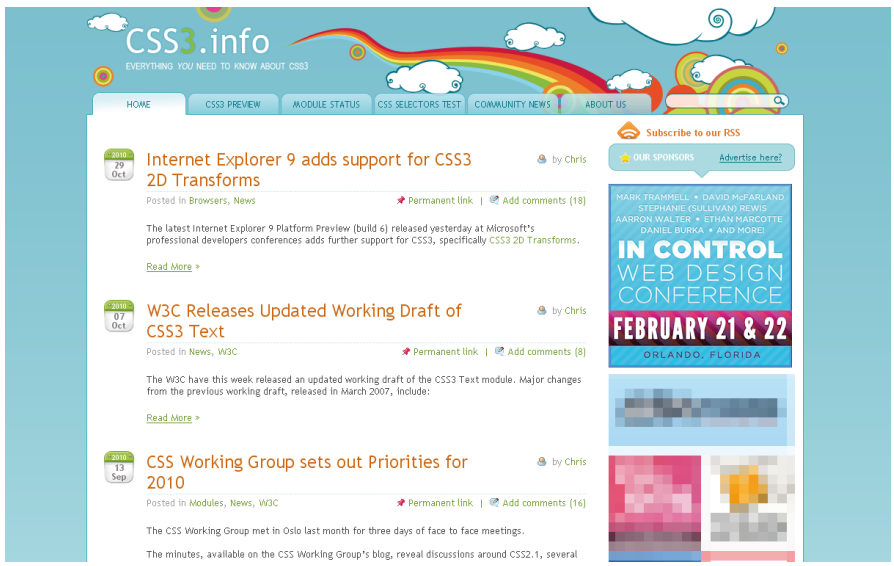
Généralités

CSS3.info : toute l'actualité des spécifications CSS 3 au jour le jour (figure C-6).

> http://www.css3.info

Figure C-6

CSS3.info



Can I Use ? : un récapitulatif complet de toutes les compatibilités des navigateurs envers HTML 5, CSS 3, SVG et autres standards web.

- http://caniuse.com

Position Is Everything : une exceptionnelle ressource sur le thème des compatibilités et erreurs des navigateurs.

- http://www.positioniseverything.net

HTML5gallery, **CSS3gallery** et **CSSmania** : des sélections de sites web méritants, généralement en HTML 5 et CSS 3.

- http://html5gallery.com
- http://css3gallery.net
- http://www.cssmania.com

Semantic Checker : extension Firefox destinée à mettre en exergue la présence (ou l'absence) d'éléments sémantiques HTML et les microformats.

- http://www.semantic-checker.com

Find me by IP : prise en charge HTML 5 et CSS 3 (voir annexe B).

- http://findmebyip.com/litmus

CSS 3

CSS3Pie : prise en charge de plusieurs propriétés décoratives CSS 3 pour Internet Explorer 6 à 8 et Opera.

- http://css3pie.com

Css3maker : très bon outil interactif de tests en direct de plusieurs propriétés CSS 3 classiques.

- http://css3maker.com

Fontsquirrel fontface generator : générateur de polices pour `@font-face` multinavigateurs.

> http://www.fontsquirrel.com/fontface/generator

Selectivizr : émulation des sélecteurs CSS 3 pour Internet Explorer 6 à 8 (figure C-7).

> http://selectivizr.com

Figure C-7

Selectivizr

Ultimate CSS Gradient Generator : un éditeur de dégradés CSS 3 en ligne très intuitif.

> http://www.colorzilla.com/gradient-editor/

IE7nomore : mon « bac à sable » personnel pour tester HTML 5 et les techniques avancées de CSS 2.1 et CSS 3. Vous y trouverez une large partie des exercices composant cet ouvrage.

> http://ie7nomore.com

IE9.js de Dean Edwards : une librairie JavaScript permettant d'améliorer la reconnaissance des standards sur les différentes versions d'Internet Explorer.

> http://code.google.com/p/ie7-js/

CSS-properties : une liste de toutes les propriétés CSS (CSS 1, CSS 2, CSS 3) (voir annexe A).

> http://meiert.com/en/indices/css-properties/

Impressive webs : tableau de prise en charge CSS 3 sur IE9.

> http://www.impressivewebs.com/css3-support-ie9/

HTML 5

Dive Into HTML 5 : pour apprendre HTML 5 comme dans un livre.

> http://diveintohtml5.org

HTML 5 Boilerplate : un gabarit HTML 5 et CSS 3 contenant toutes les bonnes pratiques du moment en termes de performances et de compatibilité entre navigateurs (figure C-8).

> http://www.html5boilerplate.com

Figure C-8

HTML 5 Boilerplate

HTML 5 Reset : une feuille de styles par défaut pour bien commencer vos documents HTML 5.

> http://html5reset.org

HTML 5 Rocks : une démonstration interactive des possibilités technologiques de HTML 5 réalisée par Google.

```
http://www.html5rocks.com
```

HTML 5 Readiness : un graphique représentant visuellement la reconnaissance de HTML 5 par les navigateurs depuis 2008.

```
http://html5readiness.com
```

Web mobile

Mobile Web Initiative du W3C

```
http://www.w3.org/Mobile/
```

XHTML Basic 1.1 Specification

```
http://www.w3.org/TR/xhtml-basic/
```

SVG Tiny 1.2 Specification

```
http://www.w3.org/TR/SVGMobile12/
```

MobileOK, le validateur mobile du W3C

```
http://validator.w3.org/mobile/
```

Mobile Flipcards

```
http://www.w3.org/2007/02/mwbp_flip_cards
```

Tableau comparatif de la prise en charge CSS des mobiles

```
http://www.w3.org/2007/03/mth/results?ts=cssmedia
```

Support d'impression

IE Print Protector : un script permettant l'affichage des éléments HTML 5 sur papier pour Internet Explorer.

▸ http://www.iecss.com/print-protector

Comptes Twitter

@alsacreations : « Communauté d'apprentissage web et agence exotique. »

@css3 : « Everything you need to know about CSS3. News, previews, tutorials and more. »

@html5 : « Notifications of changes to HTML5 drafts. »

@w3c : « The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. »

@nitot : « Mozilla Europe founder. Free Software, Open Web evangelist. »

@glazou : « CSS Working Group Co-chairman, entrepreneur, software engineer, geek, father of two, polyglot, duck lover. »

@fvsch : « Web professional. »

@deaxon : « I do apps. »

@meyerweb : « Web standards | (X)HTML | CSS | microformats | community | writing | speaking | signing man. »

@ie : « The official Twitter page for Internet Explorer. »

@docteurhtml5 : « Pour vous aider à en apprendre plus sur le HTML5. Traduction du site http://html5doctor.com. »

@smashingmag : « Get smashed with Vitaly Friedman, editor-in-chief of Smashing Magazine, an online magazine for designers and web developers. »

Livres

CSS, CSS 3

Rachel Andrew, Kevin Yank , *Everything You Know About CSS Is Wrong !*, Sitepoint, octobre 2008

Dan Cederholm, *Handcrafted CSS,* New Riders, 2009

Andy Clarke, *Hardboiled Web Design*, Five Simple Steps, fin 2010

Steve Souders, *High Performance Web Sites*, O'Reilly, 2007

Christopher Murphy, *Beginning HTML5 and CSS3 : Next Generation Web Standards*, Apress, août 2010

Jason Cranford Teague, *CSS3 : Visual QuickStart Guide (5th Edition)*, Peachpit Press, août 2010

Dan Cederholm, *CSS 3 for Web designers*, A Book Apart, novembre 2010 – traduit en français : *CSS 3 pour les Web designers*, Eyrolles, 2011

Zoe Mickley Gillenwater, *Stunning CSS 3,* New Riders, décembre 2010

HTML, HTML 5

Jeremy Keith, *HTML5 for Web Designers*, A Book Apart, juin 2010 – traduit en français : *HTML 5 pour les Web designers*, Eyrolles, 2010

Bruce Lawson, Remy Sharp, *Introducing HTML5*, Voices That Matter, juin 2010

Mark Pilgrim, *HTML5 : Up and Running*, O'Reilly, juin 2010

Matthew David, *HTML5 : Visualizing the Web*, Focal Press, juillet 2010

Web mobile

François Daoust, Dominique Hazaël-Massieux, *Relever le défi du Web mobile*, Eyrolles, 2011

Éric Sarrion, *XHTML/CSS et JavaScript pour le web mobile*, Eyrolles, 2010

Jonathan Stark, *Applications iPhone avec HTML, CSS et JavaScript*, Eyrolles, 2010

Damien Guignard, Julien Chable, Emmanuel Robles, Nicolas Sorel, *Programmation Android*, Eyrolles, 2010

Index

Symboles

A

P

Q

R

S

T

U

V

W

X

Y

Z